BIOINFORMATICS

Vol. 00 no. 00 201X
Pages 1–8

Supplementary Document for: K_2 and K_2^*: efficient alignment-free sequence similarity measurements based on the Kendall statistics

Jie Lin [1], Donald A. Adjeroh [2], Bing-Hua Jiang [3] and Yue Jiang [1*]

[1] College of Mathematics and Informatics, Fujian Normal University, Fuzhou 350108, China.
[2] Department of Computer Science & Elect. Engineering, West Virginia University, Morgantown, WV 26506, USA.
[3] Department of Pathology, Carver College of Medicine, the University of Iowa, Iowa city, IA 52242, USA.

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXX

1 INTRODUCTION

This document is the supplementary data for the paper: K_2 and K_2^*: efficient alignment-free sequence similarity measurements based on the Kendall statistics.

2 USAGE

2.1 Packages and data

The package is available: `http://community.wvu.edu/~daadjeroh/projects/K2/K2_1.0.tar.gz`.
The experiments codes are available: `http://community.wvu.edu/~daadjeroh/projects/K2/K2Experiment.R`.
The related data are available: `http://community.wvu.edu/~daadjeroh/projects/K2/Data.tar.gz`.
The entire supplementary material (including this document) is available in one archive file at: `http://community.wvu.edu/~daadjeroh/projects/K2/K2_1.0Files.tar.gz`

2.2 Usage

1. Installation

```
R> install.packages("K2_1.0.tar.gz")
```

2. Functions
   See the help file for the "K_2" package.
3. Examples
   The following shows an example on how to use the proposed K_2 and K_2^* algorithms to calculate similarity between two sequences:

```
R>library(K2)
R>seq.fa= readFasta("./Data/CaoDNA.txt")
R>K2similarity(seq.fa$seq[1],
+                 seq.fa$seq[2], k=7)
R>K2star(seq.fa$seq[1],
+          seq.fa$seq[2], SigmaSize=4)
```

The following are codes to calculate similarity among sequences in the fasta format by using K_2 method.

```
R> K2=matrix(0, nrow=seq.fa$total,
+          ncol=seq.fa$total)
R> for (i in 1:seq.fa$total)
+ {
+    for (j in 1:seq.fa$total)
+    {
+        K2[i,j]= K2similarity(
+                       seq.fa$seq[i],
+                       seq.fa$seq[j], k=7)
+    }
+ }
```

The following are codes to calculate similarity among sequences in fasta format using the K_2^* method.

```
R> K2s=matrix(0, nrow=seq.fa$total,
+          ncol=seq.fa$total)
R> for (i in 1:seq.fa$total)
+ {
+    for (j in 1:seq.fa$total)
+    {
+        K2s[i,j]= K2star(
+                       seq.fa$seq[i],
+                       seq.fa$seq[j], 4)
+    }
+ }
```

*to whom correspondence should be addressed (yueljiang@163.com)

3 DETAILED METHOD

3.1 Optimized computation of Kendall statistics

The time cost to compute $\hat{\tau}$, the approximation to the Kendall correlation statistic is $O(n^2)$, including time to compare each pair between (X_i, X_j) and (Y_i, Y_j), $i \neq j$, where n is the number of pairs in X and Y. Christensen (Christensen, 2005) showed an algorithm to calculate $\hat{\tau}$ in $O(n \log n)$ time complexity. It was implemented in Pascal. We propose a new algorithm to compute $\hat{\tau}$ which also runs in $O(n \log n)$ time, but uses a different approach. We then apply the algorithm to analyze similarity between a given pair of sequences. A related problem of computing the weighted Kendall correlation was addressed in (Lin et al., 2017).

3.1.1 Calculation of $\hat{\tau}$ Given a set of random variables X and Y, their corresponding pair (X, Y) is preprocessed by ordering X variable first and then by Y variable in increasing order. After sorting, we have $X_1 \leq X_2 \leq ...X_i... \leq X_n$. For some X_i's which have the same value: $X_i = X_{i+1} = ... = X_{i+k}$, we must have $Y_i \leq Y_{i+1} \leq ... \leq Y_{i+k}$.

For all pairs that have the same values of $\{X_i, Y_i\}$, we organize them into a group. For each $\{X, Y\}$ group, if the group starts at s and ends at e, then we must have $X_s = X_{s+1}... = X_e$, and $Y_s = Y_{s+1}... = Y_e$. In the calculation step, a group is regarded as an individual element. If all pairs inside $\{X, Y\}$ have different X_i or Y_i values, this means that we will have n individual groups. The group number is in the range of 1 to n.

The pairs are processed one by one from left to right, in increasing order of their X values. For a given current position k, $1 \leq k \leq n$, if $1 < i < k < j < n$, it means that $\{X_i, Y_i\}$ is a processed element and $\{X_j, Y_j\}$ is a pending element.

Before we dig into the detailed steps for calculating $\hat{\tau}$, we first introduce four auxiliary array data structures that we will use in the calculations. Each array is computed before hand in $O(n)$ time during preprocessing.

1. RX
   $RX[k]$: Given a current position k and corresponding X_k, $RX[k]$ contains the number of X_t whose values are less than or equal to X_k. That is, $RX[k] = |\{X_t | X_t \leq X_k, 1 \leq t \leq k\}|$. It can be precomputed in $O(n)$ time, for all k.
2. RY
   $RY[k]$: Given a current position k, $RY[k]$ is the number of Y_t whose values are less than Y_k. That is, $RY[k] = |\{Y_t | Y_t < Y_k, 1 \leq t \leq n\}|$. It is the number of pairs which has Y_t variable less than Y_k. It can be precomputed in $O(n)$ time, for all k.
3. TY
   $TY[k]$: Given a position k, $TY[k]$ is the number of pairs of (X_t, Y_t) that has the same Y_t value as Y_k. That is, $TY[k] = |\{(X_t, Y_t) | Y_t = Y_k, 1 \leq t \leq n\}|$. It can be precomputed in $O(n)$ time.
4. PY
   $PY[k]$: Given a current position k, $PY[k]$ is the number of pairs within the already processed pairs that have Y_t values as the same as Y_k. That is, $PY[k] = |\{(X_t, Y_t) | Y_t = Y_k, 1 \leq t \leq k\}|$. In the computation process, $PY[k]$ is initialized to 0 first. When scanning the list, it will be updated accordingly.

   We note the difference between $TY[k]$ and $PY[k]$: $TY[k]$ is a precomputed variable, while $PY[k]$ is a variable that is updated while scanning the sequence. At the end of scanning, both will have the same value.

To compute $\hat{\tau}$, we use :

$$\hat{\tau} = \frac{n_c - n_d}{\frac{n \times (n-1)}{2}} \tag{1}$$

From Eqn 1, we can see that the most important part is to calculate n_d (the number of discordant pairs), and n_c (the number of concordant pairs) within the $\{X, Y\}$ pairs. In the calculation process, the sorted $\{X, Y\}$ pairs are processed one by one from left to right – from the pairs with the smallest X values to those with the largest. Given a position k and its corresponding group of pairs $\{X_k, Y_k\}$, for element k, the number of discordant pairs (N_{dk}) and of concordant pairs (N_{ck}) are each calculated one by one, from left to right. The sum of these group-wise discordant and concordant numbers will give n_d and n_c respectively: $n_d = \sum_k N_{dk}$; $n_c = \sum_k N_{ck}$.

1. **Computing N_{dk}, the number of discordant pairs for an element at position k.**
   For a current position k and $\{X_k, Y_k\}$, the discordance occurs when an element has $Y_k > Y_j$ where $k < j \leq n$ and $X_k < X_j$, denoted as $|\{Y_j | Y_j < Y_k, 1 \leq k < j \leq n\}|$. The discordance number can be calculated from $N_{dk} = RY[k] - less$, where $RY[k]$ is the number of pairs inside $\{X, Y\}$ that have Y variable less than Y_k, and $less$ is the number of elements in the left side of k that have variable Y_i less than Y_k. $RY[k]$ is preprocessed as introduced previously. We can access $RY[k]$ in constant time. The value of $less$ can be calculated by using the following steps.

   - $less = |\{Y_i | Y_i < Y_k, 1 < i < k\}|$, variable $less$ contains the number of processed elements (in the left side of k) which has Y_i less than the current Y_k. It can be calculated from $less = |\{Y_i | Y_i \leq Y_k\}| - PY[k]$, where $PY[k]$ is the number of processed elements Y_i in the left side of Y_k which have $Y_i = Y_k$. Calculating $|\{Y_i | Y_i \leq Y_k\}|$ uses the binary index tree (BIT) data structure (Fenwick, 1994), which allows the value to be retrieved in $O(\log n)$ time.

2. **Computing N_{ck}, the number of concordant pairs for an element at position k.**
   This can be done using the relation: $N_{ck} = large0 - large1 - large2$. We now describe how we compute each of the three variables involved.

   - $large0$: This records the number of elements whose Y_t is greater than Y_k. That is, $large0 = |\{Y_t | Y_t > Y_k, 1 \leq t \leq n\}|$, It can be precomputed by using $large0 = n - RY[k] - TY[k]$. $RY[k]$ and $TY[k]$ are computed during preprocessing as mentioned earlier. $RY[k]$ is the number of all pairs whose $Y_k < Y_i$. Hence, $n - RY[k]$ is the number of all pairs whose $Y_k \geq Y_i$. $TY[k]$ contains the number of ties to the current pair.

- $large1$: This records the number of pairs of Y_i, where $|\{Y_i|Y_i > Y_k\}|$, $1 \leq i < k$. This is computed using the following: $large1 = k - leeq$, where, $leeq = |\{Y_i|Y_i \leq Y_k\}|$ is calculated using the BIT array. The variable $leeq$ denotes the number of processed elements that have Y_i less than and equal to Y_k. It can be calculated by using the BIT data structure in $O(\log n)$ time complexity.
- $large2$: This records the number of unprocessed elements whose X_j are equal to the current X_k. That is, $|\{X_j|j > k,\ \&\ X_j = X_k, Y_j > Y_k\}|$. We compute, $large2$ using the relation: $large2 = RX[k] - k$.

In the calculation step, we scan from left to right in $O(n)$ times. When calculating $leeq$, we use the BIT data structure (Fenwick, 1994), in $O(\log n)$ time. The other steps require constant time access to the precomputed auxiliary arrays. Thus, the total time complexity will be in $O(n \log n)$.

3.1.2 Algorithm analysis In the proposed algorithm, first, we sort the pairs of input variables X, Y in increasing order, to generate the ranked pairs. Next, to compute the number of concordant pairs, and number of discordant pairs, we check the ranked pairs one by one in the same order. At the step of processing each pair, the algorithm uses the Binary Interval Tree (BIT), also called a binary indexed tree, which was first introduced in (Fenwick, 1994). This is a key to improved efficiency in our approach. Briefly, the BIT is an indexed data structure to compute prefix sums in a series of numbers, and also to update the elements in the table. Traditionally, for a table with n elements represented using an ordinary array, we will need $O(n)$ time to compute the prefix sum, and the same $O(n)$ time to update elements in the table. With the BIT, the numbers in the table are represented in a tree, such that the value stored at a given node in the tree corresponds to the sum of the values stored at its child-nodes. Thus, this essentially stores the cumulative counts from the subtree rooted at the node. The tree structure of the BIT thus allows both the element update operation, and the computation of the cumulative (prefix) sum to be performed in $O(\log n)$ time for a given node. The BIT uses arrays to implement the tree data structure. Here, the data structure uses an array to keep track of the index of each element, where the index can be frequency of occurrence, cumulative sum, etc. Using the BIT, we can quickly compute the frequencies(/sums) of an interval from element i to element j. The time for this operation is logarithmic, requiring $O(\log n)$ time in the worst case. Thus, for n pairs, the algorithm runs in $O(n \log n)$ time complexity. This can be compared with the $O(n^2)$ time required by the original method for Kendall statistic.

3.2 Comparative complexity analysis of K_2 and K_2^*

Some variants of the D_2 statistic require time that is potentially quadratic or cubic in terms of the length of the sequences, or in terms of the size of the k-gram(/k-mer) (Song et al., 2014; Bonham-Carter et al., 2014).

Given $S = T\$P\$$ and $\mathbf{w}$, the time cost of the match function $\texttt{match}(S, i, k, \mathbf{w})$ to check if one k-mer $\mathbf{w}$ occurred at a given position in S is $O(k)$. To identify and count all the occurrences of this single k-mer will require $O(|S|k)$ time. There are potentially $O(|\Sigma|^k)$ possible substrings of size k, given the alphabet Σ. Thus, a naïve computation of the statistics will require time in $O(k|S||\Sigma|^k)$. A simple improvement can be made to avoid this exponential complexity, for instance, by observing that only those k-grams that actually occurred in the sequences (T or P) will impact the result of the statistics. Thus, the actual number of k-grams that will be matched should not exceed ($|S|$). This leads to an overall time complexity of $O(k|S|^2)$. In theory, k can be in $O(|S|)$, making this a cubic time complexity, although in practice k is relatively small, when compared with $|S|$.

Our approach relies on sophisticated data structures such as suffix arrays, and the binary interval tree. The time required to construct the suffix array structures, including suffix array (SA) and Longest Common Prefix (LCP) array, is in $O(|S|)$ time. Also, with the suffix arrays, we can count *all* the occurrences of *each* unique k-length substring $\mathbf{w}$ in S in a total time of $O(|S|)$ with the SA and LCP arrays (Manber and Myers, 1993; Gusfield, 1997; Adjeroh et al., 2008). The improved Kendall statistic uses the counts independently, and is computed in $O(|S| \log |S|)$ time via the binary interval tree (Fenwick, 1994). Thus, the overall time for our algorithm is in $O(|S| \log |S|)$. This is a significant improvement in complexity, when compared with the $O(k|S||\Sigma|^k)$ required for computing the D_2 and other related statistics, or even with the observed improvement that reduces the time to $O(k|S|^2)$. K_2^* requires just a one-time run of K_2, using the automatically computed k-parameter. This will be practially faster than using K_2, however, the time complexity still remains the same $O(|S| \log |S|)$ as in K_2.

4 RESULTS AND ANALYSIS

4.1 Correlation with the edit distance

The edit distance between two strings is defined as the minimum number of operations required to transform one string into the other. The edit distance is the basic standard used to compare two strings (Gusfield, 1997; Adjeroh et al., 2008). However, it has a quadratic time complexity with respect to the length of the strings. Thus, finding alternative distance measures that have a good correlation with the edit distance has become important in various applications in genomics.

To compare the methods, we compute the Pearson correlation coefficient between the results from each method and the standard edit distance. With the Pearson correlation coefficient, a value of 0 indicates no correlation; a value of 1 indicates positive correlation, while a value of -1 indicates negative correlation. For a comparison method, a value close to 1 or -1 indicates its ability in measuring the similarity (/dissimilarity) between sequences. A value close to 0 shows an inability to measure the similarity(/dissimilarity) between the given sequences.

4.1.1 mtDNA20 data set

Table 1 shows the Pearson correlation coefficient between the standard edit distance and the distance/similarity measure reported by each of alignment-free methods using the mtDNA20 dataset. Here, the k parameter was varied from 2 to 9. First, consider the methods that use different values for k. In the table, the best performance methods are indicated in bold for a given k. We can observe that, in general, the correlation with the edit distance increases with increasing values of k. This implies an improvement in the detection power with increasing k. Here, detection power

is measured by the absolute value of the correlation coefficient. This result is in line with those reported in (Reinert et al., 2009) where a similar increase in detection power was observed. For this performance measure, D_2^{sh} and K_2 were the best methods, with the highest correlation amongst the methods tested. This result is also in agreement with those of (Song et al., 2014) who demonstrated that D_2^{sh} (denoted D_2^s in their work) was the best D_2 statistic in their experiments. Table 1 demonstrates that K_2 and D_2^{sh} are better than the other methods in estimating the similarity(/dissimilarity) between sequences. The K_2 method is quite competitive, and comparable with D_2^{sh} in performance under this measure (Each of them outperformed the others in four or five cases respectively).

Now, consider the K_2^* method. It has a correlation coefficient of 0.94, and it outperformed the other methods in all 8 cases except K_2 with $k = 9$. Comparing K_2^* to K_2, K_2^* performed better than K_2 when k is varied from 2 to 8 (7 cases). K_2 did slightly better at $k = 9$ ($\rho = 0.95$). A key advantage of the K_2^* method is that it is able to select parameter k automatically and quickly. However, if we consider that K_2 will need to try all possible k values to determine the best k (9 in this case), the slight performance disadvantage ($\rho = 0.94$ vs. $\rho = 0.95$) of K_2^* becomes less significant, especially with increasing data volumes.

Table 1. Pearson correlation coefficient between the similarity/distance measure from different alignment-free statistical methods and the edit distance. Reports are for the mtDNA20 dataset. K_2^* with automatically determined k value, DV and Shi without varied k parameter, are all reported in the last row for brevity. D_2^z generated an error at $k = 9$.

k	D_2	D_2^*	D_2^{sh}	D_2^z	K_2	DMk	CPF	WFV
2	-0.45	-0.51	-0.55	0.02	-0.56	**0.67**	0.62	0.57
3	-0.48	-0.60	**-0.74**	0.10	-0.73	0.68	0.66	0.62
4	-0.53	-0.71	**-0.86**	-0.74	-0.82	0.70	0.71	0.63
5	-0.61	-0.79	**-0.91**	-0.81	-0.89	0.78	0.77	0.72
6	-0.77	-0.87	**-0.92**	-0.83	**-0.92**	0.84	0.86	0.68
7	-0.87	-0.91	**-0.92**	-0.84	**-0.92**	0.87	0.89	0.68
8	-0.90	-0.92	-0.91	-0.84	**-0.93**	0.85	0.89	0.66
9	-0.91	-0.91	-0.91	—	**-0.95**	0.85	0.87	0.67
	K_2^*	**-0.94**		DV	0.70		Shi	0.68

4.1.2 Fish23 dataset

Table 2 shows the Pearson correlation coefficients between the similarity measurements from the different alignment-free methods and the edit distance, using the Fish23 dataset. From the table, we see similar trends with the observations from the mtDNA20 dataset: (1) In general, the correlation with the edit distance increases with increasing values of k, which indicates an improvement in the detection power with increasing k. (2) The K_2 method is quite competitive when compared with the other methods. Especially when $k \geq 4$, K_2 demonstrates overwhelming superiority. (3) K_2^* is the best method, with respect to correlation of the distance measure with the edit distance.

4.2 Practical running time using Fish23 dataset

Table 3 and Supplementary Figure S3 show the execution times for all methods, using the second dataset (Fish23 dataset) from (Fischer et al., 2013). $K_2^* = 6.64s$, $DV = 2.57s$ and $Shi = 1.80s$ are shown in the last row of the table. Although DV and Shi are faster than K_2^*, they, however, have relatively low accuracy. From the table, we can make several observations, similar to the case with mtDNA20 dataset. (1) The respective running times for D_2^{sh}, DMk, WFV and D_2^z increase exponentially with increasing k. (2) WFV is the fastest when $k \leq 6$; (3) K_2 is faster than D_2^{sh}, D_2^*, D_2^z, DMk and WFV, when $k \geq 7$, showing a high correlation with the edit distance; (4) Combining accuracy and running time, K_2^* is the best choice, especially with huge data volumes.

Table 2. Pearson correlation coefficient between the similarity/distance measure from different alignment-free statistical methods and the edit distance, using Fish23 dataset. Results for K_2^* with automatically determined k values, DV and Shi with fixed k values, are reported in the last row. D_2^z generated an error at $k = 9$.

k	D_2	D_2^*	D_2^{sh}	D_2^z	K_2	DMk	CPF	WFV
2	-0.43	-0.56	-0.47	0.11	-0.47	**0.69**	0.66	0.49
3	-0.48	-0.66	-0.56	0.06	-0.55	**0.70**	0.67	0.67
4	-0.56	-0.69	-0.66	-0.28	**-0.81**	0.70	0.70	0.79
5	-0.68	-0.82	-0.78	-0.52	**-0.92**	0.84	0.83	0.80
6	-0.83	-0.88	-0.87	-0.63	**-0.94**	0.93	0.91	0.62
7	-0.92	-0.93	-0.92	-0.65	**-0.95**	0.93	0.94	0.60
8	-0.94	-0.94	-0.94	-0.68	**-0.95**	0.92	0.90	0.61
9	-0.94	-0.93	-0.93	—	**-0.95**	0.92	0.91	0.60
	K_2^*	**-0.96**		DV	0.66		Shi	0.60

Table 3. Practical running time (in seconds) for different methods using Fish23 dataset. Results for K_2^* with automatically determined k values, DV and Shi with fixed k values, are reported in the last row. D_2^z generated an error at $k = 9$.

k	D_2	D_2^*	D_2^{sh}	D_2^z	K_2	DMk	CPF	WFV
2	0.01	0.01	0.05	2.16	1.51	4.17	16.67	**0.004**
3	0.02	0.02	0.05	3.77	1.52	4.96	17.05	**0.008**
4	0.04	0.04	0.05	3.96	1.66	5.95	18.96	**0.020**
5	0.24	0.65	0.96	4.28	2.08	8.06	19.57	**0.080**
6	1.08	2.02	3.67	5.61	3.73	11.20	19.88	**0.722**
7	**2.81**	7.37	14.53	11.85	6.16	20.74	20.10	6.800
8	7.08	18.31	53.14	40.2	**6.43**	42.57	21.85	35.1
9	15.02	30.76	92.54	-	**7.85**	90.98	21.64	323.6
	K_2^*	6.64		DV	2.57		Shi	1.80

4.3 Number of Hops

Another way to evaluate the phylogenetic trees generated by different methods is to compare the number of hops betweem two identified nodes using a given tree, againts those observed using a reference tree. Table 4 shows the number of hops between two pairs of species.

The table suggests that the proposed methods K_2 and K_2^* are quite competitive, with the number of hops generally similar to, or bertter than, the other methods on the mtDNA20 dataset.

Similar to the analysis for the trees generated on mtDNA20 dataset, we can observe that, once again, the methods did generally

Table 4. Number of hops needed to move from one species (leaf node) to another using the different phylogenetic trees. Results shown are for two sample pairs of species, using the trees in the Supplementary Figure S1. D_2^z, WFV, Shi, and DV are omitted, given space constraint, since they did not perform as well as the other methods.

Description	Ref. Tree	D_2	D_2^*	D_2^{sh}	K_2	K_2^*	DMk	CPF
horse to cat	4	5	5	5	6	5	5	5
horse to cow	5	7	6	4	4	4	4	6

well with respect to pairwise similarity between species (i.e, the lowest level of nodes, nodes 1 to 10 in the reference tree, see the Supplementary Figure S2 (a)). But higher levels become more challenging for the methods. For instance, check the positions for *T.polylepis* and *P.flagellifer* for each tree. Again, we can use the number of hops for a quick appraisal of the performance of the methods. Similar to the Table 4, Table 5 shows the corresponding number of hops between two pairs of species, using the phylogenetic trees for the Fish23 dataset. These results suggest that K_2 and K_2^* are superior to the D_2 family of alignment-free methods on the Fish23 dataset.

Table 5. Number of hops needed to move from one species (leaf node) to another using the different phylogenetic trees. Results shown are for two sample pairs of species, using the trees in the Supplementary Figure S2. D_2^z, WFV, Shi, and DV are omitted since they did not perform as well as the other methods.

Description	Ref. Tree	D_2	D_2^*	D_2^{sh}	K_2	K_2^*	DMk	CPF
P. poll. to *C. aggr.*	7	9	8	8	8	8	10	8
P. poll. to *T. poly.*	5	6	7	6	6	6	6	6

5 DATA SET AND EXPERIMENTAL CODE

5.1 Data set

The first data set, called "mtDNA20", used in this study is the complete mtDNA sequences from (Cao et al. (1998)) with 20 eutherian species of 12 proteins encoded in the H strand of mtDNA. The gene database accession numbers are listed for reference:

```
Bos taurus (cow; database accession number V00654)
Balaenoptera physalus (fin whale; X61145),
Balaenoptera musculus (blue whale; X72204),
Phoca vitulina (harbor seal; X63726),
Halichoerus grypus (gray seal; X72004),
Felis catus (cat; U20753),
Equus caballus (horse; X79547),
Rhinoceros unicornis (Indian rhinoceros; X97336),
Mus musculus (mouse; V00711),
Rattus norvegicus (rat; X14848),
Homo sapiens (human; D38112),
Pan troglodytes (chimpanzee; D38113),
Pan paniscus (bonobo; D38116),
Gorilla gorilla (gorilla; D38114),
Pongo pygmaeus p. (Bornean orangutan; D38115),
Pongo pygmaeus abelii (Sumatran orangutan; X97707),
Hylobates lar (common gibbon; X99256),
Didelphis virginiana (Virginia opossum; Z29573),
Macropus robustus (wallaroo; Y10524),
Ornithorhynchus anatinus (platypus; X83427)
```

The second dataset contains 23 whole mitochondrial DNA genomes from different Eukaryotic fish species of the suborder Labroidei which come from (Fischer et al., 2013). We could not find sequences of two species, namely, *P. trewavasae* and *T. moorii* in NCBI gene database. Thus, though the original work in (Fischer et al., 2013) used 25 species, our dataset contained only 23 of the 25 species. We call this dataset the "Fish23" dataset.

The "CRM185" dataset contains (*cis-regulatory modules*) (CRMs), which are the same data used in (Kantorovitz et al. (2007)). The data can be downloaded from `http://veda.cs.uiuc.edu/d2z/publicdata.tar.gz`. The CRM185 dataset contains seven data sets. For completeness, below, we provide a brief description of this dataset, adapted from that provided in (Kantorovitz et al. (2007)).

- FLY_BLASTODERM : 82 non-overlapping CRMs that have expression in the blastoderm-stage embryo of fruitfly, *D. melanogaster*.
- FLY_PNS : 23 experimentally validated CRMs (average length 998 bp) with expression in the peripheral nervous system of fruitfly.
- FLY_TRACHEAL : 9 CRMs (average length 1220 bp) involved in regulation of fruitfly tracheal system.
- FLY_EYE : 17 CRMs (average length 894 bp) that express in the eye of fruitfly.
- HUMAN_MUSCLE : 28 human CRMs (average length 450 bp) involved in regulating muscle-specific gene expression.
- HUMAN_LIVER : 9 human CRMs (average length 201 bp)involved in expressions specific to the human liver.
- HUMAN_HBB : 17 CRMs (mean length 453 bp) involved in regulating the HBB complex (a set of developmentally regulated, erythroid-specific genes encoding β-globin in human).

5.2 Experimental code

Here, we explain the usage of experimental codes in this study. The file named "K2Experiment.R" contains the algorithms and the corresponding functions. First, the algorithms need to be loaded in R environment using the R command "source" as follows.

```
R> source("K2Experiment.R")
```

The following is an example code used to generate Phylogenetic trees (result is stored in a file named DNA.pdf at the current directory) and RF distances with parameter k from 2 to 9. A Phylogenetic tree generated by the K_2^* algorithm is also inside the file named DNA.pdf.

```
R> DNA.data= readFasta("./Data/CaoDNA.txt")
R> DNAexperiment(DNA.data)
```

The following codes use Fish Data set and generate Phylogenetic trees (named Fish.pdf file at current directory) and RF distances with parameter k from 2 to 9. A Phylogenetic tree generated by using the K_2^* algorithm is also inside the Fish.pdf file.

```
R> dat=readFas("./Data/FishData.fas")
R> FishExperiment(dat)
```

The evaluation of functionally related regulatory sequences using the same parameters with the original paper (Kantorovitz et al. (2007)) can be conducted as follows.

```
R> seq=read.table("./Data/CRMs.lst"
+   ,stringsAsFactors = F)
R> CRMexperiment(seq)
```

6 BACKGROUND AND RELATED WORK

Various approaches have been proposed for sequence similarity evaluation (see (Gusfield, 1997; Adjeroh et al., 2008)). Most methods are based on some form of sequence alignment (Gusfield, 1997; Wallace et al., 2005; Notredame, 2007; Altschul et al., 1990), or dynamic programming (Gregor and Thomason, 1993). The high computational complexity for sequence alignment is, however, still a major problem (Wallace et al., 2005; Notredame, 2007). Alignment-based methods therefore are faced with serious challenges, as the problem is significantly compounded by the rapid growth in the number, diversity, and size of available biological sequence data, for instance, complete genomes of various organisms, and data from next generation sequencing experiments needed to compile such complete genomes. Alignment-free methods with reduced computational cost have thus been proposed.

Alignment-free sequence comparison methods can be classified into four general types, namely, D_2 statistic family of methods (Song et al., 2014; Bonham-Carter et al., 2014), base-base correlations (BBC) (Liu et al., 2008), feature frequency profiles (FFPs) (Dai et al., 2011), and compositional vectors (CVs) (Wu et al., 2006). One approach here is computing the longest common subsequence (LCS) between two strings (Gusfield, 1997; Adjeroh et al., 2008). However, this is still generally time consuming, requiring time that is quadratic in the sequence lengths. Another approach is in computing the complexity profile for the sequences (Troyanskaya et al., 2002; Nan and Adjeroh, 2004), and then comparing the sequences based on their complexity profiles.

Perhaps, the most popular approach to alignment-free sequence comparison is by using the k-grams (also called k-mers, short patterns of length k) contained in the two or more sequences. The basis of this approach is the assumption that, if two sequences are closely related, they should also share a significant proportion of k-grams between them. In this work, we propose the K_2 statistic, a novel approach for analyzing such k-gram substrings for use in efficient similarity measurement. Based on the K_2 statistic, we propose an improved version, called K_2^*, which is able to determine the suitable k value automatically, without degrading the performance.

With the emergence of massive genomic and proteomic data sets, especially from next-generation sequencing, comparison among a huge number of biological sequences has become a critical procedure in bioinformatics. Alignment-based methods, for example, BLAST (Altschul et al., 1990) and dynamic programming (Gregor and Thomason, 1993), are not feasible with very large number of sequences, or very long sequences, due to their time-consuming nature. Given the huge computational cost of alignment-based sequence similarity measurement, alignment-free methods with reduced time and space complexity have been studied, and applied to many sequence analysis problems (Bonham-Carter et al., 2014; Song et al., 2014; Vinga and Almeida, 2003; Vinga, 2007). Compared to traditional alignment-based methods, alignment-free sequence comparison methods often use statistical approaches which are typically faster, with reduced resource requirements (Bonham-Carter et al., 2014). This makes them very popular in sequence comparison for various applications.

6.1 D_2 family

Our approach is more closely related to the D_2 family of statistics. Thus, we describe these in some more detail below, to provide a context for our contribution. To formalize the description, we first introduce some notations needed for general string/sequence analysis, which will be used to describe some popular algorithms for alignment-free sequence analysis. Let $T = T[1 \dots n]$ be a given string of length n, over an alphabet Σ. Let $T = \alpha\beta\gamma$, for some strings α, β, and γ (α and γ could be empty). The string β is called a *substring* of T, α is called a *prefix* of T, while γ is called a *suffix* of T. We will use $T[i]$ to denote the i-th symbol in T, and $T[i \dots j]$ to denote a substring that starts at position i in T and ends at position j (inclusive). We let $P = P[1 \dots m]$ be another string of length m over Σ that needs to be compared with T for similarity. A k-mer in T (also called a k-gram (Gusfield, 1997; Adjeroh et al., 2008), k-tuple (Reinert et al., 2009), or k-word (Lippert et al., 2002)) is simply a substring $T[i \dots j]$ of T with length $k = j - i + 1$.

To describe the members of the D_2 family, we will follow the notations in (Reinert et al., 2009). Let $\bar{n} = n-k+1, \bar{m} = m-k+1$. Let $\mathbf{w} = w_1 w_2 \dots w_k \in \Sigma^k$ be any possible k-mer that can be formed from the sequence alphabet Σ. Let $X_{\mathbf{w}}$ record the number of times the k-mer $\mathbf{w}$ occurred in T. Define the match function $\texttt{match}(T, i, k, \mathbf{w})$ as follows:

$$\texttt{match}(T, i, k, \mathbf{w}) = \begin{cases} 1 & : \quad T[i \dots i+k-1] = \mathbf{w} \\ 0 & : \quad \texttt{otherwise} \end{cases} \tag{2}$$

Then, $X_{\mathbf{w}}$ is defined as follows:

$$X_{\mathbf{w}} = \sum_{i=1}^{\bar{n}} \texttt{match}(T, i, k, \mathbf{w}), \tag{3}$$

Similarly, we use $Y_{\mathbf{w}}$ to count the number of times $\mathbf{w}$ occurred in P, the second sequence:

$$Y_{\mathbf{w}} = \sum_{i=1}^{\bar{m}} \texttt{match}(P, i, k, \mathbf{w}), \tag{4}$$

The D_2 word count statistic (Lippert et al., 2002; Reinert et al., 2009) is then defined as follows:

$$D_2 = \sum_{\mathbf{w} \in \Sigma^k} X_{\mathbf{w}} Y_{\mathbf{w}}. \tag{5}$$

The asymptotic properties of the D_2 distribution are studied in (Lippert et al., 2002; Reinert et al., 2009), where they established conditions for normal approximations for D_2. The D_2 statistic is known to suffer from the potential problem of being dominated by single-sequence background noise, for the case of non-uniform symbol distributions (Lippert et al., 2002; Reinert et al., 2009). Reinert et al. further argued that the D_2 is not a desirable static for measuring sequence similarity, especially given that its capability (or power) to detect similarity or relatedness between sequences decreases with increasing length of the sequences, or as the sequences become more similar.

To improve the performance of D_2, Kantorovitz et al. (Kantorovitz et al., 2007) modified the static by using the z-scores of the D_2 values, rather than the direct value, viz:

$$D_2^z = \frac{D_2 - \mathcal{E}(D_2)}{sd(D_2)}, \tag{6}$$

where, $\mathcal{E}(x)$ is the expected value of x, and $sd(x)$ is the standard deviation. D_2^z was shown to improve the performance of D_2 statistic on the problem of clustering biologically-related sequences (Kantorovitz et al., 2007). But as pointed out in (Reinert et al., 2009), the problem of dominance of single-sequence background noise for the case of non-uniform symbol distributions is still an issue for this method. Reinert et al. (Reinert et al., 2009) thus proposed and analyzed two other modifications to the D_2 statistic. The first was a self-standardized version, defined as follows: Given $\mathbf{w}$, let $p_{\mathbf{w}} = \Pi_{i=1}^{k} p_{w_i}$ be the probability of occurrence of the k-mer $\mathbf{w}$, where p_{w_i} is the probability of symbol w_i, ($w_i \in \Sigma$), the i-th symbol in the k-mer $\mathbf{w}$. Define, the respective centralized counts for T and P as follows: $\tilde{X}_{\mathbf{w}} = X_{\mathbf{w}} - \bar{n}p_{\mathbf{w}}$; $\tilde{Y}_{\mathbf{w}} = Y_{\mathbf{w}} - \bar{m}p_{\mathbf{w}}$; Then, using these centralized frequencies, the new statistic is given as:

$$D_2^{sh} = \sum_{\mathbf{w} \in \Sigma^k} \frac{\tilde{X}_{\mathbf{w}} \tilde{Y}_{\mathbf{w}}}{\sqrt{\tilde{X}_{\mathbf{w}}^2 + \tilde{Y}_{\mathbf{w}}^2}}, \tag{7}$$

It was shown that, under certain standard assumptions, the D_2^{sh}-statistic (also called D_2^s-statistic) is generally normally distributed. The statistic is related to the earlier observations by Shepp (Shepp, 1964) on the normality of the ratio of the product and square root of the sum of squares of independent zero-mean normally distributed variables.

The second modification is a variation of the D_2^{sh}-statistic to accommodate the fact that, in most cases, only estimates of the symbol probabilities will be available, rather than the exact probabilities. Thus, the exact symbol probability $p_\sigma, \sigma \in \Sigma$ is replaced with its estimate $\hat{p}_\sigma, \sigma \in \Sigma$. The new statistic is then defined by changing the denominator of the previous equation:

$$D_2^* = \sum_{\mathbf{w} \in \Sigma^k} \frac{\tilde{X}_{\mathbf{w}} \tilde{Y}_{\mathbf{w}}}{\sqrt{\bar{n}\bar{m}\hat{p_{\mathbf{w}}}}} \tag{8}$$

In (Reinert et al., 2009), it was shown that D_2^* provided a better measure for sequence similarity, exhibiting superior power for detection of relatedness between sequences, when compared with D_2, or D_2^S. They also performed detailed simulations to study the distributions of the different k-mer-based statistics for sequence similarity measurement, using both the *common motif* model and the *pattern transfer* model between sequences, under the null hypothesis that the two sequences are independent and identically distributed. Overall, the observation is that the power of the different statistics generally increases with increasing k, and increasing sequence lengths n and m, except for D_2. It was also noted that this increase in power comes at a greater computational requirement.

6.2 Robinson-Foulds distance

To compare the similarity/dissimilarity between two trees, we use the Robinson-Foulds(RF) distance (Robinson and Foulds, 1981). The Robinson-Foulds distance (also called the symmetric difference metric) is a well-known approach for measuring the similarity between two trees. (See for example, (Bansal et al., 2010; Lu et al., 2017). The Robinson-Foulds distance measures the topological distance between two labeled trees essentially by counting the minimum number of elementary operations needed to transform one tree to the other. Given two trees and their node labels (some node labels could be empty), the Robinson-Foulds distance between them is computed by using two elementary operations: the contraction operation that merges multiple nodes into one, and the decontraction (split) operation that splits a node into multiple nodes, as needed to transform one tree into another. The distance is given by the total number of contraction and decontraction operations needed for the transformation. This is akin to the traditional edit distance between two strings, which computes the minimum number of edit operations needed to transform one string to another. Under the Robinson-Foulds model, two trees are said to be the same if the trees are isomorphic, and the labels in the trees are preserved by the isormorphism. In general, a smaller value of the Robinson-Foulds distance implies more similarity between the trees.

Thus, to compare the phylogenetic trees generated by various alignment-free sequence comparison techniques, we compute the Robinson-Foulds distance between the golden reference phylogenetic tree and the phylogenetic tree generated by a given alignment-free algorithm. The computed distances can thus be used to rank the performance of the algorithms, with the one that generated the minimum Robinson-Foulds distance taken as the best performing algorithm. Various algorithms for efficient computation of the Robinson-Foulds distance have been proposed (Day, 1985; Pattengale et al., 2007). For this work, we use the Analyses of Phylogenetics and Evolution (APE) package (Paradis et al., 2004) to calculate the Robinson-Foulds distance. This package imports the trees in the Newick format. It also can read the distance matrix to construct the phylogenetic tree. The APE package computes the

topological distance between two trees and returns the Robinson-Foulds distance between them.

REFERENCES

Adjeroh, D., Bell, T., and Mukherjee, A. (2008). The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching. Springer Publishing Company, Berlin, German, 1 edition.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. Journal of Molecular Biology, **215**(3), 403–410.

Bansal, M. S., Burleigh, J. G., Eulenstein, O., and Fernndezbaca, D. (2010). Robinson Foulds Supertrees. Algorithms for Molecular Biology, **5**(1), 1–12.

Bonham-Carter, O., Steele, J., and Bastola, D. (2014). Alignment-free genetic sequence comparisons: a review of recent approaches by word analysis. Briefings in Bioinformatics, **15**(6), 890–905.

Cao, Y., Janke, A., Waddell, P. J., and Westerman, M. e. a. (1998). Conflict among individual mitochondrial proteins in resolving the phylogeny of eutherian orders. Journal of Molecular Evolution, **47**(3), 307–322.

Christensen, D. (2005). Fast algorithms for the calculation of Kendall's tau. Computational Statistics, **20**(1), 51–62.

Dai, Q., Li, L., Liu, X., Yao, Y., Zhao, F., and Zhang, M. (2011). Integrating overlapping structures and background information of words significantly improves biological sequence comparison. PLoS One, **6**(11), e26779.

Day, W. H. E. (1985). Optimal algorithms for comparing trees with labeled leaves. Journal of Classification, **2**(1), 7–28.

Fenwick, P. M. (1994). A new data structure for cumulative frequency tables. Software-Practice and Experience, **24**(3), 327–336.

Fischer, C., Koblmller, S., Glly, C., Schltterer, C., Sturmbauer, C., and Thallinger, G. G. (2013). Complete mitochondrial DNA sequences of the threadfin cichlid (petrochromis trewavasae) and the blunthead cichlid (tropheus moorii) and patterns of mitochondrial genome evolution in cichlid fishes. PLoS One, **8**(6), e67048.

Gregor, J. and Thomason, M. G. (1993). Dynamic programming alignment of sequences representing cyclic patterns. IEEE Trans. Pattern Anal. Mach. Intell., **15**(2), 129–135.

Gusfield, D. (1997). Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge, England.

Kantorovitz, M., Robinson, G., and Sinha, S. (2007). A statistical method for alignment-free comparison of regulatory sequences. Bioinformatics, **23**, i249–i255.

Lin, J., Adjeroh, D. A., Jiang, B.-H., and Jiang, Y. (2017). fastwkendall: an efficient algorithm for weighted Kendall correlation. accepted by Computational Statistics.

Lippert, R. A., Huang, H., and Waterman, M. S. (2002). Distributional regimes for the number of k-word matches between two random sequences. PNAS, **99**(22), 13980–13989.

Liu, Z., Meng, J., and Sun, X. (2008). A novel feature-based method for whole genome phylogenetic analysis without alignment: Application to HEV genotyping and subtyping. Biochemical and Biophysical Research Communications, **368**(2), 223 – 230.

Lu, B., Zhang, L., and Leong, H. W. (2017). A program to compute the soft Robinson-Foulds distance between phylogenetic networks. BMC Genomics, **18**(2), 111.

Manber, U. and Myers, G. (1993). Suffix arrays: A new method for on-line string searches. SIAM Journal of Computing, **22**, 935–938.

Nan, F. and Adjeroh, D. A. (2004). On complexity measures for biological sequences. In 3rd IEEE CSB'04, Stanford, CA, USA, Aug. 16-19, 2004, pages 522–526.

Notredame, C. (2007). Recent evolutions of multiple sequence alignment algorithms. PLoS Comput Biol, **3**(8), 1–4.

Paradis, E., Claude, J., and Strimmer, K. (2004). APE: Analyses of phylogenetics and evolution in R language. Bioinformatics, **20**(2), 289–290.

Pattengale, N. D., Gottlieb, E. J., and Moret, B. M. (2007). Efficiently computing the robinson-foulds metric. Journal of Computational Biology A Journal of Computational Molecular Cell Biology, **14**(6), 724–735.

Reinert, G., Chew, D., Sun, F., and Ms., W. (2009). Alignment-free sequence comparison (I): statistics and power. Journal of Computational Biology, **16**(12), 1615–1634.

Robinson, D. F. and Foulds, L. R. (1981). Comparison of phylogenetic trees. Mathematical Biosciences, **53**(1), 131–147.

Shepp, L. (1964). Normal functions of normal random variables. SIAM Review, **6**(4), 459–460.

Song, K., Ren, J., Reinert, G., Deng, M., Waterman, M. S., and Sun, F. (2014). New developments of alignment-free sequence comparison: measures, statistics and next-generation sequencing. Briefings in Bioinformatics, **15**(3), 343 – 353.

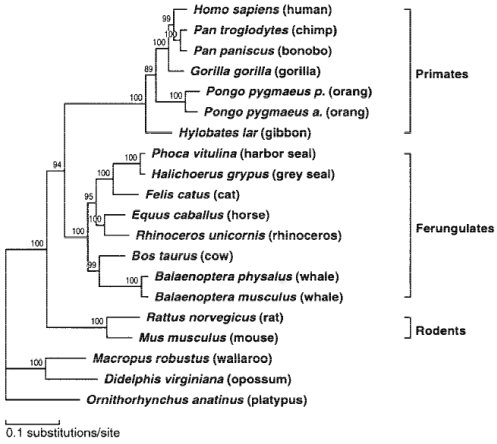
Troyanskaya, O. G., Arbell, O., Koren, Y., Landau, G. M., and Bolshoy, A. (2002). Sequence complexity profiles of prokaryotic genomic sequences: A fast algorithm for calculating linguistic complexity. Bioinformatics, **18**(5), 679–688.

Vinga, S. (2007). Biological sequence analysis by vector-valued functions: revisiting alignment-free methodologies for DNA and protein classification. Iranian Journal of Medical Physics, pages 71–107.

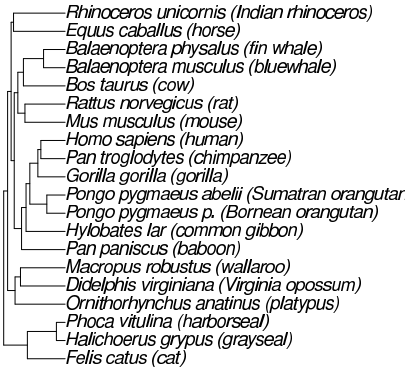
Vinga, S. and Almeida, J. (2003). Alignment-free sequence comparison: a review. Bioinformatics, **19**(4), 513–523.

Wallace, I. M., Blackshields, G., and Higgins, D. G. (2005). Multiple sequence alignments. Current Opinion in Structural Biology, **15**(3), 261 – 266.

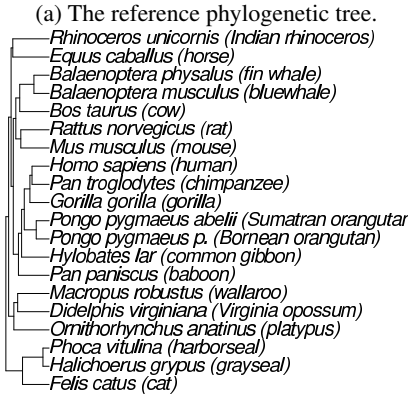
Wu, X., Wan, X.-F., Wu, G., Xu, D., and Lin, G. (2006). Phylogenetic analysis using complete signature information of whole genomes and clustered neighbour-joining method. International Journal of Bioinformatics Research and Applications, **2**(3), 219–248.



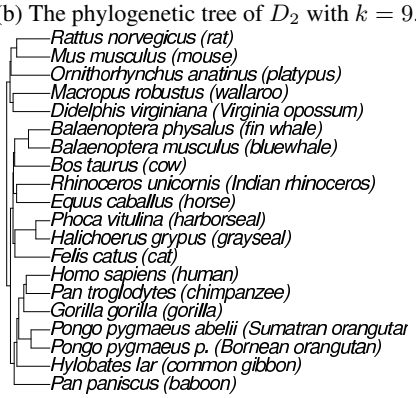
(a) The reference phylogenetic tree.



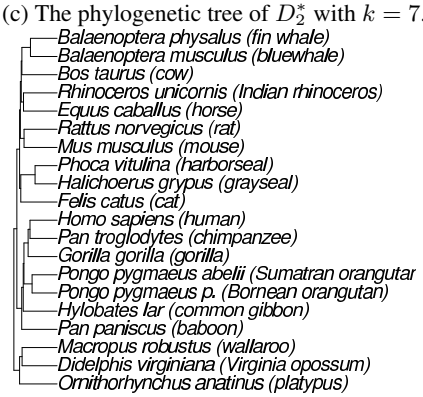
(b) The phylogenetic tree of D_2 with $k = 9$.



(c) The phylogenetic tree of D_2^* with $k = 7$.

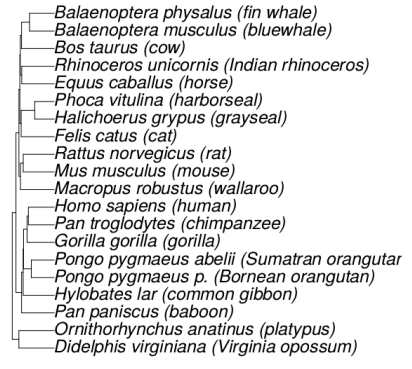


(d) The phylogenetic tree of D_2^{sh} with $k = 7$.



(e) The phylogenetic tree of K_2 with $k = 7$.

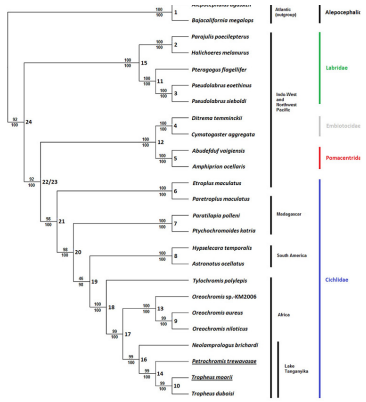
(f) The phylogenetic tree of K_2^* .



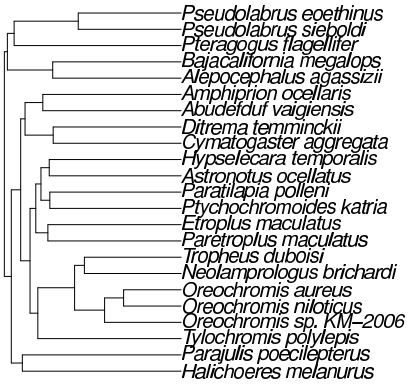
(g) The phylogenetic tree of DMk with $k = 7$.

(h) The phylogenetic tree of CPF with $k = 7$.

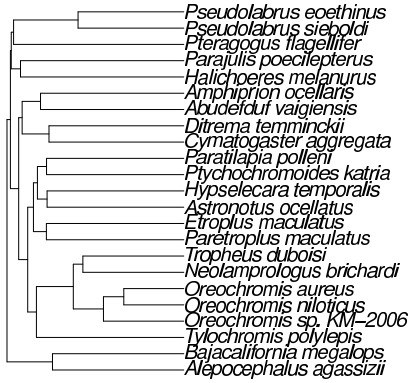
Supplementary Figure S1: Phylogenetic trees generated by different alignment-free sequence comparison methods, using the mtDNA20 dataset.



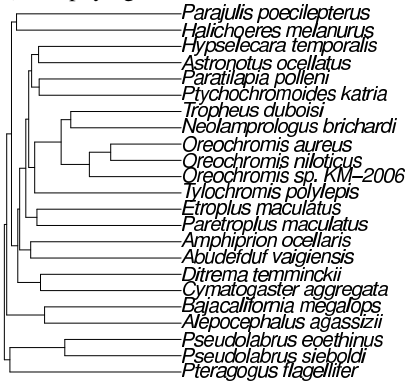
(a) The reference phylogenetic tree.



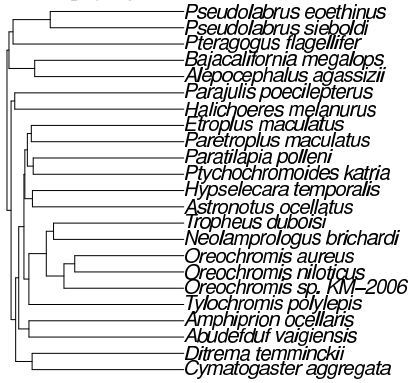
(b) The phylogenetic tree of D_2 with $k = 8$.



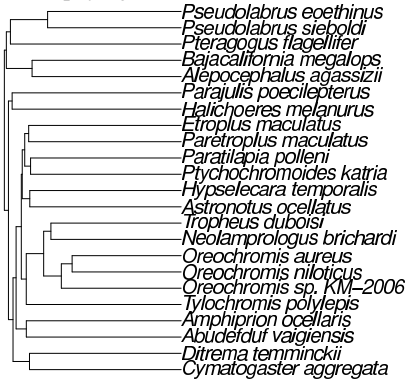
(c) The phylogenetic tree of D_2^* with $k = 8$.



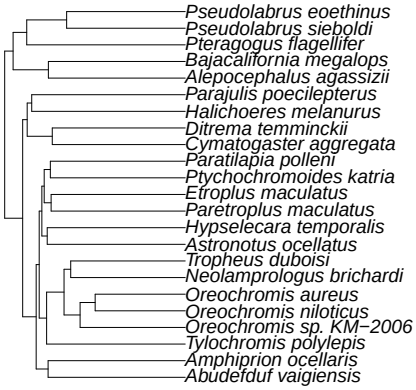
(d) The phylogenetic tree of D_2^{sh} with $k = 8$.



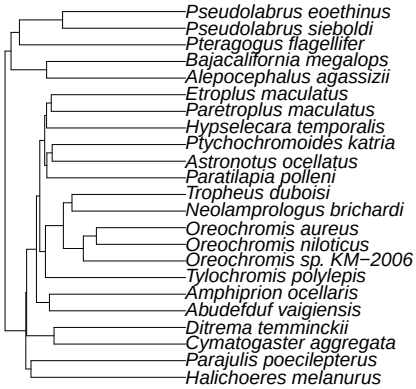
(e) The phylogenetic tree of K_2 with $k = 8$.



(f) The phylogenetic tree of K_2^* .

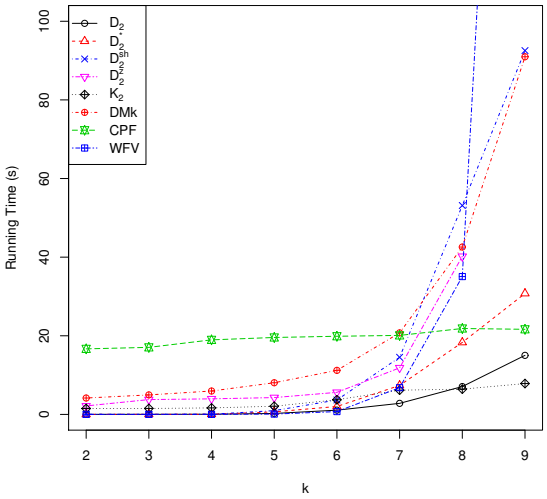


(g) The phylogenetic tree of DMk with $k = 6$.



(h) The phylogenetic tree of CPF with $k = 7$.

Supplementary Figure S2: Phylogenetic trees generated by different alignment-free sequence comparison methods, using the Fish23 dataset.



Supplementary Figure S3: Time cost comparison for D_2, D_2^*, D_2^{sh}, D_2^z, DMk, CPF, WFV and K_2 with parameter k varying from 2 to 9 on on the Fish32 dataset. Results for K_2^*=6.64s, DV=2.57s and Shi=1.80s are not shown in the figure for clarity.