MASC: A Linear Method for Multiple Nucleotide Sequence Alignment on Spark Parallel Framework

Journal:	*Journal of Computational Biology*
Manuscript ID	JCB-2017-0040.R1
Manuscript Type:	Method Article
Keyword:	MULTIPLE ALIGNMENT
Manuscript Keywords (Search Terms):	Multiple Sequence Alignment, Suffix Tree, Center-Star Strategy, Spark
Abstract:	Multiple sequence alignment (MSA) is an essential prerequisite and dominant method to deduce the biological facts from a set of molecular biological sequences. It refers to a series of algorithmic solution for the alignment of evolutionarily related sequences while taking into account evolutionary events such as mutations, insertions, deletions, and re-arrangements under certain conditions, etc. These methods can be applied to DNA, RNA or protein sequences. In this work, we take advantage of a center-star strategy to reduce the MSA problem to pairwise alignments, and we use a suffix tree to match identical substrings between two pairwise sequences. MASC can accomplish MSA in O(mn), which is linear time complexity, where m is the number of sequences, and n is the average length of sequences. Furthermore, we execute our method on the Spark distributed parallel framework to deal with ever-increasing massive datasets. Our method is significantly faster than previous techniques, with no loss in accuracy for highly similar nucleotide sequences like homologous sequences, which we experimentally demonstrate. Comparing with mainstream MSA tools (e.g., MAFFT), MASC could finish the alignment of 67200 sequences, longer than 10 thousand bps, in 9 minutes which takes MAFFT over 3.5 days.

MASC: A Linear Method for Multiple Nucleotide Sequence Alignment on Spark Parallel Framework

Wenhe SU, School of Computer Science and Technology, National University of Defense Technology, Changsha, China

Quan ZOU, School of Computer Science and Technology, Tianjin University, Tianjin, China

Xiangke LIAO, School of Computer Science and Technology, National University of Defense Technology, Changsha, China

Yutong LU, School of Computer Science and Technology, National University of Defense Technology, Changsha, China

Shaoliang PENG*, School of Computer Science and Technology, National University of Defense Technology, Changsha, China

*Corresponding author: E-mail: pengshaoliang@nudt.edu.cn

Connect Information:

Name	Telephone	E-mail
Wenhe SU	+86 13272094630	suwenhecn@gmail.com
Quan ZOU	+86 17092261008	zouquan@nclab.net
Xiangke LIAO	+0731 84576322	xkliao@nudt.edu.cn
Yutong LU	+0731 84576322	ytlu@nudt.edu.cn
Shaoliang PENG	+86 13574817196	pengshaoliang@nudt.edu.cn

Abstract

Multiple sequence alignment (MSA) is an essential prerequisite and dominant method to deduce the biological facts from a set of molecular biological sequences. It refers to a series of algorithmic solution for the alignment of evolutionarily related sequences while taking into account evolutionary events such as mutations, insertions, deletions, and re-arrangements under certain conditions, etc. These methods can be applied to DNA, RNA or protein sequences. In this work, we take advantage of a center-star strategy to reduce the MSA problem to pairwise alignments, and we use a suffix tree to match identical substrings between two pairwise sequences. MASC can accomplish MSA in O(*mn*), which is linear time complexity, where *m* is the number of sequences, and *n* is the average length of sequences. Furthermore, we execute our method on the Spark distributed parallel framework to deal with ever-increasing massive datasets. Our method is significantly faster than previous techniques, with no loss in accuracy for highly similar nucleotide sequences like homologous sequences, which we experimentally demonstrate. Comparing with mainstream MSA tools (e.g., MAFFT), MASC could finish the alignment of 67200 sequences, longer than 10 thousand bps, in 9 minutes which takes MAFFT over 3.5 days.

Key words: Multiple Sequence Alignment, Suffix Tree, Center-Star Strategy, Spark

1. Introduction

Multiple sequence alignment (MSA) is the alignment of more than two molecular biological sequences, aiming to discover maximally similar (or identical) amino acid or identical nitrogenous base positions across the aligned query set of sequences. MSA is an essential preprocess to many bioinformatics analyses, including homology modeling, secondary structure prediction, phylogenetic reconstruction, and protein structure and function prediction. An MSA is visualized as a two-dimensional matrix in which the rows are the individual sequences and the columns are maximally similar or identical amino acid or nitrogenous base positions arranged to correspond by inserting gap characters in appropriate positions (known as indels). An MSA can provide a wealth of information about the structure/function relationships within a set of sequences, such as the evolutionary conservation of functionally or structurally important sites, and conserved hydrophobicity patterns in precise regions.

Although dynamic programming like Needleman–Wunsch algorithm (Needleman and Wunsch, 1970) can be generalized in theory to produce alignments for any number of sequences, unfortunately, this leads to an explosive increase in computer time and memory requirements as the number of sequences increases (Taylor, 1990). MSA remains under continuous development, and is regarded as one of the most challenging problems in the field of bioinformatics and computational biology. (Chatzou *et al.*, 2016) Furthermore, the computation of an accurate MSA has long been known to be an NP-complete problem (Wang and Jiang, 1994), a situation that explains why over 100 alternative methods have been developed these last three decades, and many mainstream methods use heuristic and combinatorial optimization algorithms for obtaining approximation of optimal solution, imposed by the NP-complete nature of the problem. (Chatzou *et al.*, 2016)

We develop a new method that performs pairwise alignment in O(*n*) time between two highly similar sequences based on a suffix tree structure, which is well-known for its string processing power. Here, we take two sequences which have at least 56.8 percent nucleotides in common as highly similar sequences. The number of 56.8% is calculated in the experiment sector. We also design a novel MSA method called MASC (Multiple sequence Alignment based on a Suffix tree and Center-star strategy), which has extremely high performance and accuracy with our new pairwise alignment algorithm. Furthermore, we implement MASC on the Spark parallel distributed framework for use case for massive scale sequence data and accelerate the method by using multiple computing node parallel programming.

2. Related Work

Multiple Sequence Alignment to be one of the most widely used modeling methods in biology is indeed an important modeling tool whose development has required addressing a very complex combination of computational and biological problems. (Chatzou *et al.*, 2016) (Centre and Regulation, 2015)

The most commonly used heuristic methods involve progressive-alignment strategy (Hogeweg and Hesper, 1984), iterative-alignment strategy (Wallace *et al.*, 2005), or center-star strategy (Zou *et al.*, 2009). ClustalW (Thompson *et al.*, 1994) is the most widely used implementation of the progressive-alignment strategy. MAFFT (Lounkine *et al.*, 2012) is quite fast, using a fast Fourier transform algorithm along with a progressive-alignment strategy.

Progressive-alignment generates a quasi-phylogenetic guide tree among the sequences and gradually builds up the alignment in a pairwise fashion, following the order provided by the tree. Although successful for a wide variety of cases, the main caveat of the progressive alignment approach is the existence of local minima, e.g. the early computation of the first pairwise alignment may prevent the computation of a globally optimal MSA. Errors made in initial alignments cannot be rectified later as the remainder of the sequences are added (Notredame, C., Higgins, D. G., & Heringa *et al.*, 2000). Apart from this, progressive alignment is time-consuming when dealing with large-scale datasets, due to its nonlinear time complexity. (Zou *et al.*, 2012)

The iterative strategy(Wallace *et al.*, 2005) is an interesting alternative method, which is based on tree-based progressive strategy and involves re-estimating trees and alignments until both converge. Iterative strategies do not provide any guarantee of an optimal solution, but are reasonably robust, and are much less sensitive to the number of sequences than their deterministic counterparts (Gotoh, 1996). PRRN (Gotoh, 1996) and MUSCLE (Edgar, 2004) employ an iterative-alignment strategy.

The center-star alignment strategy, which is utilized in our project, is a fast method for solving MSA, and suitable for highly similar sequences. Assume that it takes O(t) to run a pairwise alignment between two sequences. The complexity of center star strategy is O($m^2t + mt$) where m is the number of sequences.

The main approach underlying the center star method is to transform MSA into pairwise alignments based on a 'center sequence'. Center sequence is selected using a similarity matrix, a symmetric matrix that stores the similarity scores of each of the two sequences. The similarity scores are calculated by pairwise alignments, which indicate the similarity of two sequences. The construction of this matrix takes O(m^2t) time. When center sequence is selected, other sequences are pairwise aligned to the center sequence. Then, all of the inserted spaces are summed to obtain the final MSA result. (Zou *et al.*, 2015) This process takes O(mt) time, t is the time cost of a pairwise alignment.

Regardless of which heuristic method used, the main simplification idea common to all is to reduce the MSA to a series of pairwise sequence alignments. Consequently, pairwise alignment is a dominant component of all MSA techniques. Traditional dynamic programming algorithms, such as Smith–Waterman (Smith and Waterman, 1981) and Needleman–Wunsch (Needleman and Wunsch, 1970), require O(n^2) temporal and spatial complexity to perform pairwise alignment, where *n* is the maximum length of two sequences. Other, faster algorithms have been developed, such as MAFFT (Lounkine *et al.*, 2012), which is O(*n*log*n*). These algorithms work extremely well on conventional tasks with multiple single protein sequences, cDNA sequences, or relatively short genomic DNA sequences containing a single gene and simple intron interruptions. Furthermore, when performing the pairwise alignment between two very long sequences, such as mtDNA (Iborra *et al.*, 2004), or whole genomes, many algorithms either run out of memory or take too long to complete (Delcher *et al.*, 1999). To the best of our knowledge, there is no efficient and effective pairwise alignment method that requires O(n) time complexity.

3. Algorithms

We contribute three novel points for MASC. First, suffix tree (Ukkonen, 1995) is employed for pairwise alignment, which can reduce the time complexity to O(n) for similar DNA sequences. Here n is the sequence length. Second, we improve the center star multiple sequence alignment strategy by selecting randomly center star sequence (Zou *et al.*, 2012). Experiments in Section 4.2 prove that center star sequence selection would not influence the performance for similar DNA sequences. Finally, MASC is implemented on the Spark distributed parallel framework pursuing further acceleration by parallel computing (Zaharia *et al.*, 2010).

3.1. Suffix Tree Pairwise Alignment

The basis of our method is a data structure known as a suffix tree which is a compressed trie (Aho and Corasick, 1975) containing all the suffixes of a given text as keys, and positions in the text as values. Suffix trees allow particularly fast implementations of many important string operations (Baeza-Yates and Gonnet, 1996). The suffix tree for the string S of length n is defined as follows:

Definition:

The tree has exactly *n+1* leaves numbered from 0 to *n*. Except for the root, every internal node has at least two children. Each edge is labeled with a non-empty substring of S. No two edges starting out of a node can have string-labels beginning with the same character. The string obtained dby concatenating all the string-labels found on the path from the root to leaf *i* spells out a suffix, a simple substring that begins at any position in S and extends to the end of S.

In Figure 2, a complete suffix tree of string "agctggcc$" is shown.

The efficient algorithms given by Ukkonen can construct a suffix tree in linear time with O(*n*) computational space from a string, where *n* is the length of the string. (Ukkonen, 1995) Ukkonen's algorithm also reduces suffix tree construction to O(*n*) time, for constant-size alphabets, and O(*n*log*n*) in general. Within a bioinformatics context, the tree alphabets consist of {A, G, C, T} or {A, G, C, U} for DNA or RNA, respectively, or the 20 amino acid symbols for proteins. Therefore, the time cost is linear for all molecular biological sequences (Barsky *et al.*, 2008).

The input to the pairwise alignment are two sequences. We call them sequence S1 and sequence S2 for convenience, it is assumed that the sequences to be compared are highly similar. Therefore, there are many common substrings in both S1 and S2. The alignment process consists of four steps, which are shown in Figure 1.

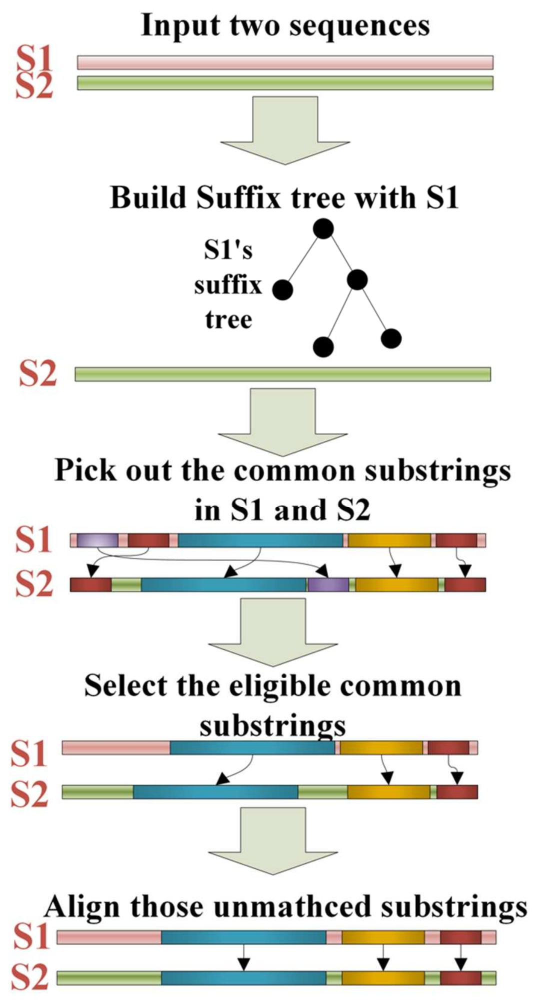

Input two sequences
S1
S2
Build Suffix tree with S1
S1's suffix tree
S2
Pick out the common substrings in S1 and S2
S1
S2
Select the eligible common substrings
S1
S2
Align those unmathced substrings
S1
S2

Figure 1 Pairwise alignment process using a suffix tree

(1) The first step of our method is to build a suffix tree from one sequence. Here S1 is assumed as the chosen one, and the tree's name is tree-S1. The tree construction consumes O(n) time with Ukkonen's algorithm, n is the length of S1.

(2) Then we use a process to pick out common strings. The pseudocode of the process is shown below:

Input: S1，S2: two Strings;tree-S1: the suffix tree of S1;

Output: result-list: a list of substrings' information, whose element is composed of tuples like (a substring's start position in S1, a substring's start position in S2, the length of substring)

```
List<Tuple> pick_out_common_strings(String S1, S2, Suffix tree tree-S1)
    index=0;
    while (index<S2's length){
        /*Select the longest common prefix of one of S1's suffix and the substring of S2 starting at index, return the start-position and length */
        start-position, length=tree-S1.select_prefix(S2,index);
        if(start-position != -1){
            /*A common prefix(substring) is found*/
            Record the start-postion of the prefix in both S1 and S2 in result-list;
            Record the length of the prefix in result-list;
            index+=length;
        }
        else {
            /*Common prefix is not found*/
            index++;
        }
    }
```

```
        return result-list;

}
```

The function named select_prefix, which is a member function of suffix tree data structure, is applied to find the longest common prefix between a given input string and a suffix in the tree. A prefix is simply a subsequence that begins at the beginning of the sequence and extend to any position of the sequence.

When we search common substrings between a string and a suffix tree, the select_prefix function is used to find the longest common prefix, then the common prefix is skipped and process reuses select_prefix to find another common prefix. This would be repeated until all characters in S2 are scanned. Thus, in a single scan of string S2, all common substrings can be identified.

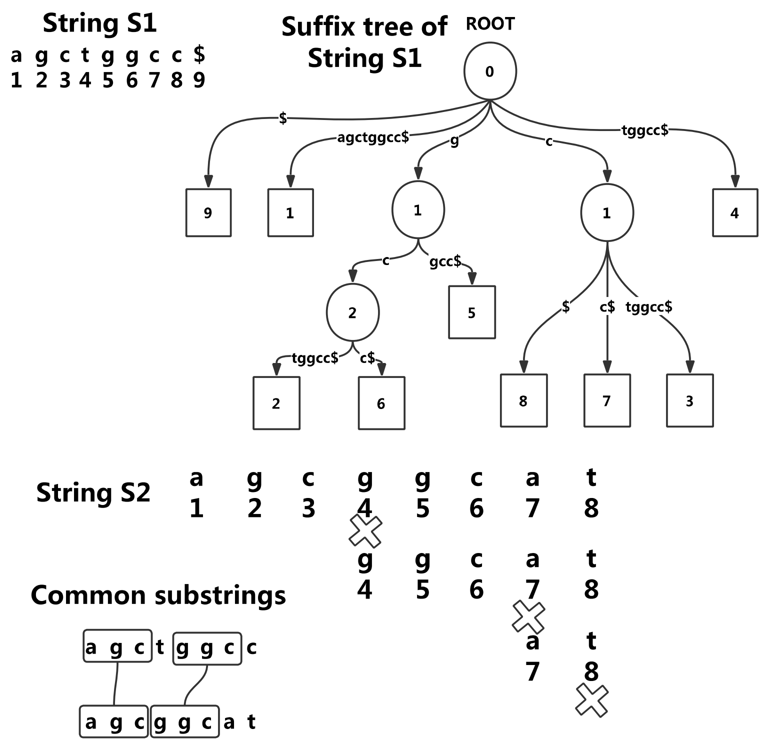


Figure 2 Suffix tree of a string and the example of selection of common substrings between "agctggcc" and "agcggcat".

In Figure 2, S1 has 9 suffixes and they are all stored in its suffix tree. The square nodes are leaf nodes and represent suffixes. They are labeled by the start position of a suffix. The inner nodes are labeled by the length of the prefix of a suffix.

The process of picking out common substrings is shown step by step. At first, the string is "agcggcat", S2 itself. The process traverses S1's suffix tree until mismatch occurs at the fourth character 'g'. Then "agcg" is skipped and the string is updated to "ggcat", the process traverses the tree for the second time until mismatch occurs at the seventh character ‘a’. Then “ggc” is skipped and the string is updated to “at”. The process repeats again and again until all characters in S2 are scanned. The selected common substrings are shown. ‘a’ and ‘t’ are omitted, although they are also selected.

(3) The process of picking out common strings in S1 and S2 with sequence S2 and tree-S1 is run to form two common substring sets. The next step is dropping out ineligible substrings. Our standard for what constitutes eligible common substrings is described below.

The criterion is the starting positions of two matching substrings cannot be too far away. A pair of matching substrings is kept if the difference between their start positions is less than their length, otherwise they will be discarded. For highly similar sequences, most part of the sequences are the same. The matching of remote substrings would enlarge the areas which need to be aligned by dynamic programming algorithm so that the time of alignment is extended.

(4) Next, the process aligns the remaining unmatched substrings between S1 and S2. Finally, all substrings are concatenated to form the aligned sequences.

The construction of the suffix tree can be accomplished in O(*n*) time by Ukkonen algorithm where *n* is the length of the sequence. Picking all common substrings can be done in a single scan of string S2 which can be accomplished in O(*n*) time where n is the length of S2. The alignments among unmatched substrings can be finished in O(km^2), where *k* is the number of substrings and *m* is the maximal length of these unmatched substrings. For highly similar sequences, $k \times m$ is far less than *n*, so the total time complexity is O(*n*).

3.2. Center-Star Strategy

The center-star strategy (Zou *et al.*, 2009) is a method that can solve the MSA problem with extremely high performance. And, the strategy's performance improves with increasingly similar sequences. For a set of sequences S = {s_1, s_2,..., s_{n-1}, s_n}, the center sequence S_{center} can satisfy the following formula:

$$S_{center} = \arg\max_{s_i}\left(\sum_{j=1, j\neq i}^{n} score(s_i, s_j)\right)$$

The formula would help find out the sequence which is similar to all others. Here $score(s_i, s_j)$ is the similarity of two sequences si and sj. A higher score means that two sequences are more similar. The results of pairwise alignments between any two sequence in S are used to calculate the similarity between sequences. The scores are stored in similar matrix, which is a symmetric matrix. Then the matrix is applied to compute the sum of scores in a row. The sequence which has maximum sum of scores is selected as the center sequence.

After that, pairwise alignments are carried out between the center and the other sequences, one by one, each pairwise alignment gives two consequent sequences, the results of pairwise alignment. All the inserted spaces on center sequence are summed to obtain a aligned center sequence. At last, pairwise alignments between aligned center sequence and every sequence in S are carried out. The center-star strategy is shown in following algorithm outline:

```
Input: sequence set S= {$s_1$, $s_2$, $s_3$…$s_{n-1}$, $s_n$}
Output: aligned sequence set S''= {$s_1''$, $s_2''$, $s_3''$…$s_{n-1}''$, $s_n''$}

1. Run pairwise alignment between every two sequences, and construct similarity
   matrix W, which is a symmetric matrix that stores the similarity scores of each of
   the two sequences in set S.
2. Select $S_{center}$ using W.
3. Run pairwise alignments between the center and the other sequences.
4. Sum all the inserted spaces on center sequences to get the aligned center
   named.
5. Run pairwise alignment between aligned center and the other sequences again
   to get the result of the MSA.
6. Output the result.
```

When performing MSA among a set of similar sequences, it takes O(n) to finish a pairwise alignment with our suffix tree method previously illustrated. The construction of similarity matrix W can be accomplished in O(m^2n), where m is the sequence number of a set. Pairwise alignments among the center and the other sequences can be finished in O(mn) time. The total time cost is no more than O($nm^2 + mn$). The process of finding the center sequence takes most of the time.

However, when aligning similar sequences, the elements in similarity matrix are just the same. That means every sequence can be regarded as an average one. Each sequence can be selected as a center. Selecting a sequence randomly as center is a good choice which is justified by Section 4.2. As a result, the process of construction of similarity matrix and selection of center sequence can be omitted. In this way, the time complexity of MASC is reduced to O(mn).

3.3 Distributed and Parallel Implementation with Spark

As MASC can perform MSA in linear time with highly similar sequences, it has enormous potential for very-high-throughput cases. However, in reality an ordinary computer cannot keep so many long sequences in memory, possibly leading to a memory crash, or wasting excessive time shifting between physical and virtual memory. Consequentially, we need to solve the memory storage problem, and parallelize our method using distributed parallel frameworks.

There are many frameworks available for handling large scale data. MapReduce (Dean and Ghemawat, 2004) and Spark (Zaharia *et al.*, 2010) are the most powerful and popular distributed computing frameworks. Spark is more suitable for this work, due to its memory computation characteristics.

The serial version of our method is illustrated in Figure 3.

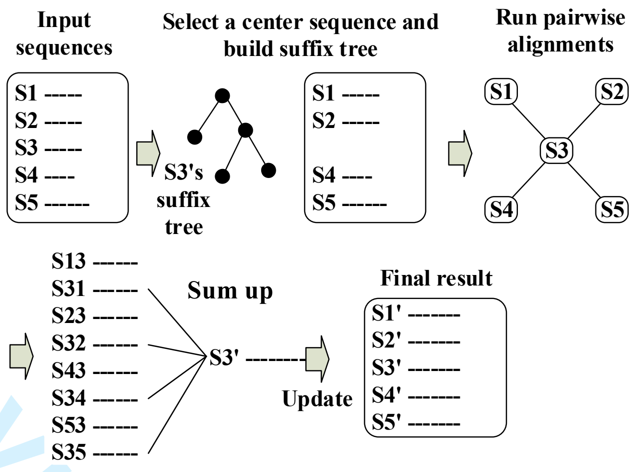


Figure 3 MASC flowchart

After inputting the data, the first task is to select a sequence as center and build its suffix tree. Then, the pairwise alignments between the center and all other sequences are run. Next, the process of summing up the spaces among sequences is carried out. Finally, the result is output.

We analyze the process to find 'hot spots' within the program, the process of "pairwise alignment and sum up spaces", which occupies most of the computational time, as shown in Figure 4.

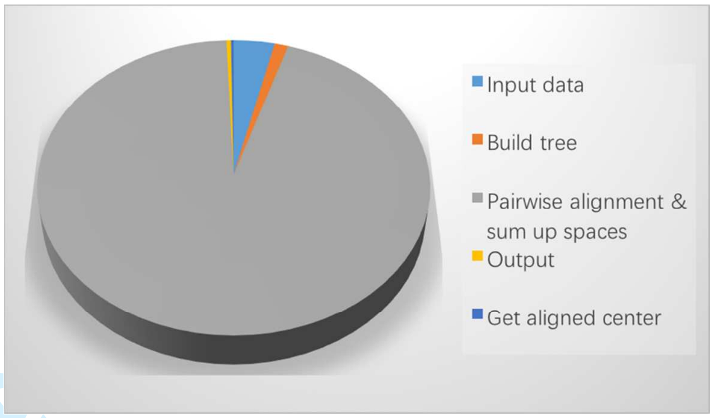


Figure 4 Time cost of MASC steps

Fortunately, in MASC the process of alignment between the center and the other sequences can be parallelized, because there isn't too much data dependency.

According to Amdahl's law (Moncrieff *et al.*, 1996) :

$$T = T_s + T_p$$

$$Speedup = \frac{T_s + T_p}{T_s + \frac{T_p}{p}}$$

T is the total time that a program consumes, which consists of T_s and T_p. T_p is the time consumed by the part of program that can be parallelized. T_s is the time consumed by the part of program that must be run in serial. And p is the number of processors. So the speedup of our method is:

$$\lim_{n\to\infty} \text{Speedup} = \frac{6\% + 94\%}{6\% + \frac{94\%}{n}} = 16$$

The limitation of the speedup of our program is 16 in theory. MASC was therefore developed within the Spark framework, and the process is shown in **Supplementary Method**. The Spark version of MASC is more than 8 times faster than original MASC in practice. The result of parallelization is shown in Section 4.5

4. Experiments

4.1. Datasets and Measurements

Balibase (Thompson *et al.*, 2005) is regarded as the golden benchmark for most MSA research. However, the database is relatively small, and is only suitable for protein sequence alignment. Because there is no benchmark dataset for addressing the large-scale nucleotide MSA problem, we employ human mitochondrial genomes (mt genomes) and 16S rRNAs as the test data in our experiment. Human mitochondrion genome is associated with Alzheimer's Disease, Parkinson's Disease, and Type 2 Diabetes (Tanaka *et al.*, 2004). MASC may help analyze the function of mt genome. Our human mt genome dataset contains 672 highly similar mt genome sequences, with a maximal length of 16,579 bp, and a minimal length of 16,556 bp. With the aim of testing the performance of our program with large-scale data, we duplicate the mt genomes 20 times, 50 times, and 100 times to enlarge the test set.

Table 1 Detailed information on the experimental DNA dataset

Dataset	Max length	Min length	Average length	Sequence number	File size
mt genome(1x)	16579	16556	16569.7	672	10MB
mt genome(20x)				13440	213MB
mt genome(50x)				33600	532MB
mt genome(100x)				67200	1.1GB

Our main project purpose is to accelerate the MSA process, and improve the capacity to handle massive data. Therefore, we focus our attention on running time and throughput. We choose the sum-of-pairs (SP) value (Zou *et al.*, 2009), which is a measurement of the quality of an MSA, for measuring alignment performance. The SP value is an integer representing the sum of every pairwise alignment score from an MSA. For our purposes, in a pairwise nucleotide sequence alignment, if two nucleotides from the same column are different, one is added to the SP value, while if a space is inserted, two is added to the score, otherwise, if the two nucleotides are the same, the score remains unchanged. Thus, the SP value will be a positive integer, and a lower SP value indicates better quality of MSA. However, SP values are not suited for massive MSAs because the score may become too large and exceeds the computer's limitations. Therefore, we employ the average SP value instead, which is the SP value divided by the number of sequences (Zou et al., 2015).

4.2. Center sequence selection

The section is intended to prove it is feasible to select a center sequence randomly when align highly similar sequences. A dataset contains 600 homologous sequences selected randomly from mt genome dataset is used to execute center-star MSA 600 times, which use every sequence as center. Then the results of these experiments are quantitatively assessed by SP score. Then SP score is normalized and transformed into z-score. The formula is as following:

Given,

$$sp(s) = \text{the SP score of the MSA's result which use } \boldsymbol{s} \text{ as center}$$

$$\overline{sp} = \sum_{every\ s} sp(s) = 15775.38 \text{ [Sample SP score mean]}$$

$$\sigma = \sqrt{\frac{1}{599}\sum_{i=1}^{600}(sp(s_i) - \overline{sp})^2} = 322.7115 \text{ [Sample SP score std. dev]}$$

Then,

$$Z(s) = \frac{sp(s) - \overline{sp}}{\sigma}$$

We collect 600 z-scores and analyze the distribution of these data. The distribution is shown in Figure 5.

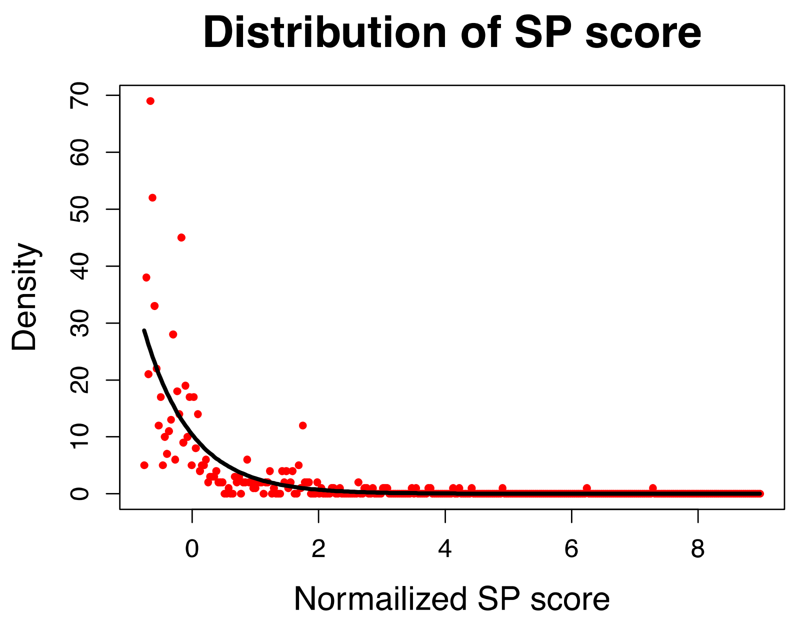


Figure 5 The distribution of normalized SP score

From the fitting model, the probability that z-score less than x is:

$$P(z < x) = \int_{-0.76}^{x} 0.484e^{-1.34t}\,dt = 1 - 0.36e^{-1.34x}$$

When a MSA is carried out with a random selected center sequence, the probability of Z-score less than 4, which means the SP score is less than 17063, is 0.9983. The SP of the result of MAFFT and KAlign is between 16500 and 17000. That means random selection of center sequence is feasible.

4.3. Sequence similarity analysis

MASC is a method to do multiple sequence alignment among highly similar sequences. In order to find out what kind of sequences are suitable to be aligned by MASC, an experiment is designed to quantify the similarity of sequences. A 1000bp long sequence is taken as basic sequence in the experiment. Then variations are carried on the copies of basic sequence. The dataset is formed by basic sequence and variated copies. The variation is to change one residue of basic sequence to one element of {A, G, C, T, -} where '-' means to delete the residue. The quotient of the number of residues which are changed divided by length of basic sequence is called mutation rate. In the experiment, mutation rate ranges from 0 to 100%. The result shows that MASC works when the mutation rate is no more that 54%.

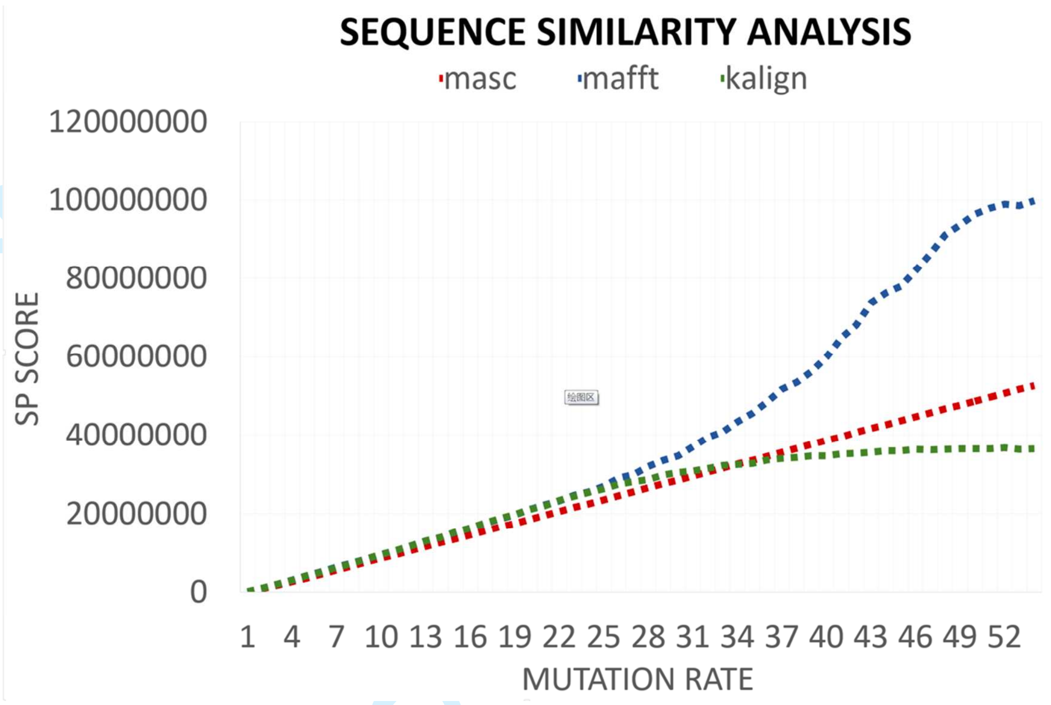


Figure 6 SP value of MASC, MAFFT and KAlign with mutation rate changed

In Figure 6, less SP value is on behalf of better accuracy. Before mutation rate gets to 33%, MASC has a better accuracy than KAlign, and the SP score and mutation rate are linear related. When the mutation rate is over 33%, KAlign has the best accuracy. MASC is more accurate than MAFFT when the mutation rate is no more than 54%. Due to the variation is selected from {A, G, C, T, -} in random, five elements have the same probability to be chosen, so that the residue has 20% probability to be unchanged (e.g. when mutate a residue 'A', it has 20% probability to choose 'A'). It means that the MASC works when there are more than 56.8% ((100%-54%)+54%/5=56.8%) residues are same between sequences that need to be aligned.

4.4. Comparison with State-of-Art Tools

Most of the available state-of-the-art MSA software tools cannot address large-scale data. Therefore, we only perform comparisons with MAFFT and KAlign. KAlign and MAFFT are run on single nodes, without any parallel operations. Therefore, for fairness, we carry out all the experiments on the same server (Intel® Xeon® CPUE7-8890v3 @ 2.5 GHz with 2 TB total memory, and the Red Hat Enterprise Linux Server release 7.1 operating system). Table 2 shows time consumption for the various human mitochondrial genome datasets.

Table 2 Time consumption for different MSA tools using human mitochondrial genome datasets of different multiplicities

	672(1x)	13440 (20x)	33600 (50x)	67200 (100x)
MASC-serial	35s	10m54s	26m14s	51m51s
MASC-spark	7s	1m50s	5m11s	8m44s
MAFFT	1m59s	3h52m14s	21h54m18s	3d12h41m42s
KAlign	1h27m10s	---	---	---

MSA is run with three software on different datasets which contains different number of sequences. From Table 2 we can see that MAFFT and KAlign take an extremely long time to finish the MSA among long sequences, even with relatively small files. Furthermore, KAlign cannot even handle datasets has more than 13440 sequences. However, MASC (serial version) is compatible with massive files of long sequences, and the parallelized version runs extremely faster than all other programs. The parallelization analysis will be shown in Section 4.5.

Table 3 Accuracy of different methods

	MASC	MAFFT	KAlign
10M(1x)	14276	15202	14809

Table 3 shows a comparison of the average SP values among the different programs. As we have described previously, a lower SP value means better accuracy. Because the human mitochondrial genome dataset sequences are highly similar, the different programs perform similarly. However, the data in Table 3 shows that our method is somewhat more accurate than the other two methods with this dataset.

MASC can be used for any file, no matter how large it is. In comparison, MAFFT cannot be used for files larger than 1 GB (67200 sequences), and KAlign cannot be used for files larger than 10MB (672 sequences). MASC is clearly superior for processing massive MSAs.

4.5. Speedup of Spark

We demonstrate the speedup and scalability of the Spark version of MASC in this section. The method we use is introduced in **Supplement Method.** We perform our experiments on a server with 16 CPUs and 2 TB of memory (same specifications as previous).

We test the 1×, 20×, 50×, and 100× human mt genome datasets with different modes that run various numbers of executors. Figure 7 shows the results of those experiments. Each combination of executors number (Zaharia *et al.*, 2010) and dataset was run over different 50 times. Here executors mean threads that do pairwise alignments.

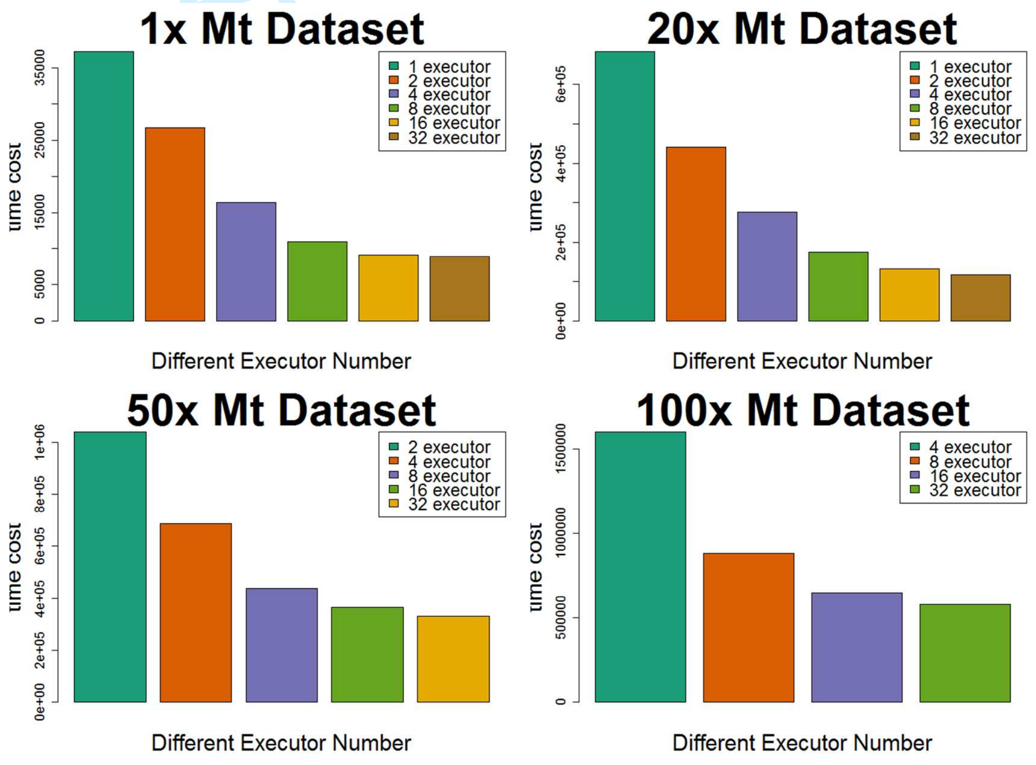


Figure 7 Running times for different mt genome datasets with different modes

In Figure 7, each bar plot shows the performance of MASC on different datasets. Each column shows the average time cost of the program running in different environments (which can be regarded as different executors on the cluster). The unit of y-axis is second. We do not test the one and two executor modes for the 100×

dataset, and the one executor mode for the 50× dataset, because it is too difficult for one node keeping such a huge copy of massive data in memory. We see a remarkable speedup by Spark parallelization when we use no more than 32 executors at the same time. The speedup can be over ten sometimes, with an average total value of eight. The scalability is well-maintained, and efficiency remains constant, while we enlarge the dataset and the number of executors at the same time.

5. Conclusion

Multiple Sequence Alignment is one of the most widely used modeling methods in biology, which is the basis of many analyses. These analyses include domain analysis, phylogenetic reconstruction, motif finding, etc. Recently the development of aligner includes the need of upscaling under the high-throughput sequencing pressure and the need for more complex sequence descriptors including Non Coding RNA (ncRNA) or non-transcribed genomic sequences. Likewise, the explosion of available genomic data has put a lot of pressure on the development of a new aligners. Nowadays, parallelization is regarded as an important strategy to accelerate the process of MSA. (Chatzou *et al.*, 2016)

We propose MASC in this study, which can perform MSA in O(*mn*) time among highly similar sequences, where *m* is the number of sequences in the dataset, and *n* is the average length of the sequences. MASC has high accuracy and performance. The core idea of our method is to accelerate the MSA process using three steps:

We speed up the process of pairwise alignment based on suffix tree, which is a powerful data structure for handling strings. Time complexity is O(*n*) at the most, when aligning pairs of similar sequences based on suffix tree. A center-star strategy is then employed as a heuristic to reduce MSA to pairwise alignments. MASC can be accomplished in O(*mn*) time when the sequences are highly similar and there is little loss in accuracy. Along with MASC's extremely high performance, we used the distributed parallel computing framework Spark to enlarge the memory of the system, and, in this way, the throughput of our program is substantially increased.

Extensive experiments with MASC were then performed. First, the accuracy and performance of our method was tested compared with other state-of-art tools. The scope of the comparison was quite limited, because most available tools are not optimized for performance nor efficiency. MAFFT and KAlign are two optimized tools available that were selected for comparison. The results of our experiments show that we have made some progress, and our method has better accuracy than the other two methods with our sample datasets, as indicated by lower average SP values.

MASC has been implemented on Spark and HDFS to handle the increasingly expansive data in the field of biology and bioinformatics. The method is very suitable for parallelization because the pairwise alignments between sequences are independent. In practice, the Spark version of MASC has great speedup and scalability. Both the single thread and the parallel tools are developed using Java, which works on multiple operating systems. Java 1.8 and Spark 2.0 are prerequisites for its operation. The codes and tools are accessible free of charge at https://github.com/suwenhecn/MASC .

Although MASC has a good performance and accuracy, it still has some unsolved defeats. The ideal scenario for our method is aligning datasets without complex variation. The performance of MASC decreases a lot when it has to handle complex variations. As the sequences are not similar, suffix tree pairwise alignment get less efficient. Because there are fewer common substrings and more large areas of unmatched pairs need to be aligned by Needleman Wunsch algorithm. Apart from this, our improved center-star strategy could lead to an inaccurate result because it selects a center sequence randomly. For these reasons, we intend to develop MASC to adapt complex variations in our future work. Though the scope of application of MASC is limited in theory, it is still quite useful in many research. In order to solve these tough problems, an algorithm with better robustness need to be develop. MASC need further development for more general applications.

Acknowledgement

The work was supported by the Natural Science Foundation of China (No. 61370010, 61272056, U1435222, 61133005) and Guangdong Provincial Department of Science and Technology under grant No. 2016B090918122. Thanks to Ms Dong’s help and her companion.

Reference

Aho,A. V. and Corasick,M.J. (1975) Efficient string matching: an aid to bibliographic search. *Commun. ACM*, **18**, 333–340.

Baeza-Yates,R. a. and Gonnet,G.H. (1996) Fast text searching for regular expressions or automaton searching on tries. *J. ACM*, **43**, 915–936.

Barsky,M. *et al.* (2008) A new method for indexing genomes using on-disk suffix trees. *Proceeding 17th ACM Conf. Inf. Knowl. Manag.*, 649–658.

Centre,C.M. and Regulation,G. (2015) Multiple sequence alignment modeling : methods and applications Multiple Sequence Alignment Modeling : Methods and Applications. *Brief. Bioinform.*

Chatzou,M. *et al.* (2016) Multiple sequence alignment modeling: methods and applications. *Brief. Bioinform.*, **17**, 1009–1023.

Dean,J. and Ghemawat,S. (2004) MapReduce: Simplified Data Processing on Large Clusters. *Proc. OSDI - Symp. Oper. Syst. Des. Implement.*, 137–149.

Delcher,A.L. *et al.* (1999) Alignment of whole genomes. *Nucleic Acids Res.*, **27**, 2369–2376.

Edgar,R.C. (2004) MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, **5**, 113.

Gotoh,O. (1996) Significant Improvement in Accuracy of Multiple Protein Sequence Alignments by Iterative Refinement as Assessed by Reference to Structural Alignments. *J. Mol. Biol.*, **264**, 823–838.

Hogeweg,P. and Hesper,B. (1984) The alignment of sets of sequences and the construction of phyletic trees: an integrated method. *J. Mol. Evol.*, **20**, 175–186.

Iborra,F.J. *et al.* (2004) The functional organization of mitochondrial genomes in human cells. *BMC Biol.*, **2**, 1–14.

Lounkine,E. *et al.* (2012) Large-scale prediction and testing of drug activity on side-effect targets. *Nature*, **486**, 361–7.

Moncrieff,D. *et al.* (1996) Heterogeneous computing machines and Amdahl's law. *Parallel Comput.*, **22**, 407–413.

Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similiarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.

Notredame, C., Higgins, D. G., & Heringa,J. *et al.* (2000) T-coffee: a novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, **302**, 205–217.

Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.

Tanaka,M. *et al.* (2004) Mitochondrial Genome Variation in Eastern Asia and the Peopling of Japan Mitochondrial Genome Variation in Eastern Asia and the Peopling of Japan. *Genome Res.*, 1832–1850.

Taylor,W.R. (1990) Hierarchical method to align large numbers of biological sequences. *Methods Enzymol.*, **183**, 456–474.

Thompson,J.D. *et al.* (2005) BAliBASE 3.0: Latest developments of the multiple sequence alignment benchmark. *Proteins Struct. Funct. Genet.*, **61**, 127–136.

Thompson,J.D. *et al.* (1994) CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Ukkonen,E. (1995) On-line construction of suffix trees. *Algorithmica*, **14**, 249–260.

Wallace,I.M. *et al.* (2005) Evaluation of iterative alignment algorithms for multiple alignment. *Bioinformatics*, **21**, 1408–1414.

Wang,L. and Jiang,T. (1994) On the complexity of multiple sequence alignment. *J Comput Biol*, **1**, 337–348.

Zaharia,M. *et al.* (2010) Spark : Cluster Computing with Working Sets. *HotCloud'10 Proc. 2nd USENIX Conf. Hot Top. cloud Comput.*, 10.

Zou,Q. *et al.* (2012) A Novel Center Star Multiple Sequence Alignment Algorithm Based on Affine Gap Penalty and K-Band. *Phys. Procedia*, **33**, 322–327.

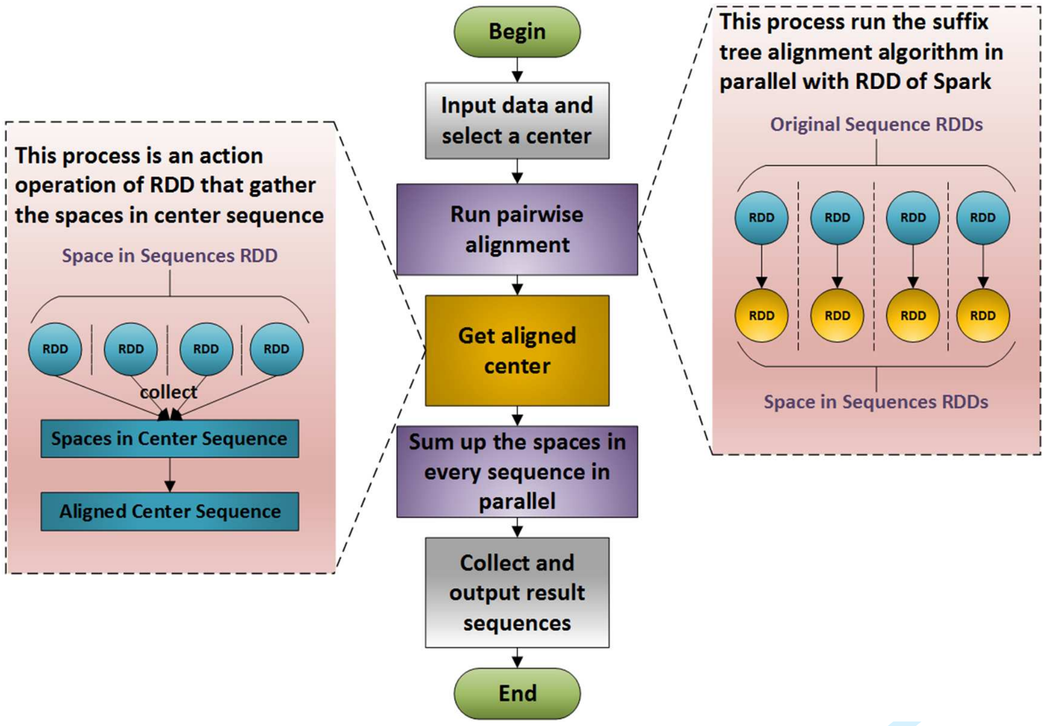
Zou,Q. *et al.* (2009) An Algorithm for DNA Multiple Sequence Alignment Based on Center Star Method and Keyword Tree. *Acta Electron. Sin.*, **38**, 1746–1750.

Zou,Q. *et al.* (2015) HAlign: Fast multiple similar DNA/RNA sequence alignment based on the centre star strategy. *Bioinformatics*, **31**, 2475–2481.

Supplementary Method

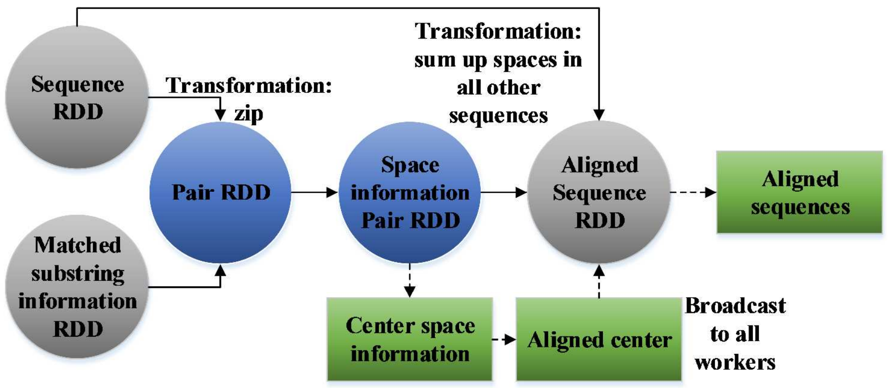
Spark Spark is a MapReduce type framework which can carry on all computation in memory, without saving intermediate results. The main abstraction Spark provides is resilient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated in parallel. RDDs are created by starting with a file in the file system, or an existing collection in the driver program, and transforming it. The collection means a set of data which is distributed on different computers in a cluster.

Design of Spark version of MASC Our Spark center-star MSA has two stages which is shown in Supplementary Figure 1.


Begin
Input data and select a center
Run pairwise alignment
Get aligned center
Sum up the spaces in every sequence in parallel
Collect and output result sequences
End
This process is an action operation of RDD that gather the spaces in center sequence
Space in Sequences RDD
RDD
collect
Spaces in Center Sequence
Aligned Center Sequence
This process run the suffix tree alignment algorithm in parallel with RDD of Spark
Original Sequence RDDs
Space in Sequences RDDs

Supplementary Figure 1 Implementation of MASC on Spark

Initially, data is input from the local file system or an Apache Hadoop Distributed File System (HDFS), and a format examination is performed. Next, the sequence array is parallelized, which converts the string lists to sequence string RDDs. Concurrently, the program needs to choose a center sequence, which, as previously explained, is randomly chosen. This random selected sequence is then used to construct a center sequence suffix tree, and pairwise alignments are run between the tree and all other sequences. In this step, the suffix tree is used to get matching substrings in each sequence serially, because wasting memory can be avoided by keeping a single copy of the center suffix tree. When the information regarding all the matching substrings is obtained, a parallel process of aligning the unmatched substrings is run by passing the function to the Spark transformation, which implements the Needleman–Wunsch algorithm. In the previous step, the sequence string RDDs are transformed into pair RDDs of spaces in the center sequence and spaces in all other sequences. Next, the RDDs of spaces in the center sequence are collected to make the center aligned. Then the aligned center sequence is broadcast to all executors, and align all other sequences. In this step the pair RDDs of spaces are transformed into string RDDs of aligned sequences, and the results are stored. Finally, the aligned string RDDs are collected and are output to the local file system. The data flow and operations are shown in Supplementary Figure 2.



Supplementary Figure 2 Data flow and operations

In Supplementary Figure 2, all the nodes represent data and the edges represent the operations. The cycle elements are RDDs in Spark, which are distributed in the executors, the rectangle elements are datasets in the driver's memory. The operations signed by solid lines represent the transformations in Spark that transfer RDDs into subsequent RDDs and the dotted line edges represent the action operations that convert RDDs into dataset in driver memory.

Table 1 Detailed information on the experimental DNA dataset

Dataset	Max length	Min length	Average length	Sequence number	File size
mt genome(1x)	16579	16556	16569.7	672	10MB
mt genome(20x)				13440	213MB
mt genome(50x)				33600	532MB
mt genome(100x)				67200	1.1GB

Table 2 Time consumption for different MSA tools using human mitochondrial

genome datasets of different multiplicities

	672(1x)	13440(20x)	33600(50x)	67200(100x)
MASC-serial	35s	10m54s	26m14s	51m51s
MASC-spark	7s	1m50s	5m11s	8m44s
MAFFT	1m59s	3h52m14s	21h54m18s	3d12h41m42s
KAlign	1h27m10s	---	---	---

Table 3 Accuracy of different methods

	MASC	MAFFT	KAlign
10M(1x)	14276	15202	14809

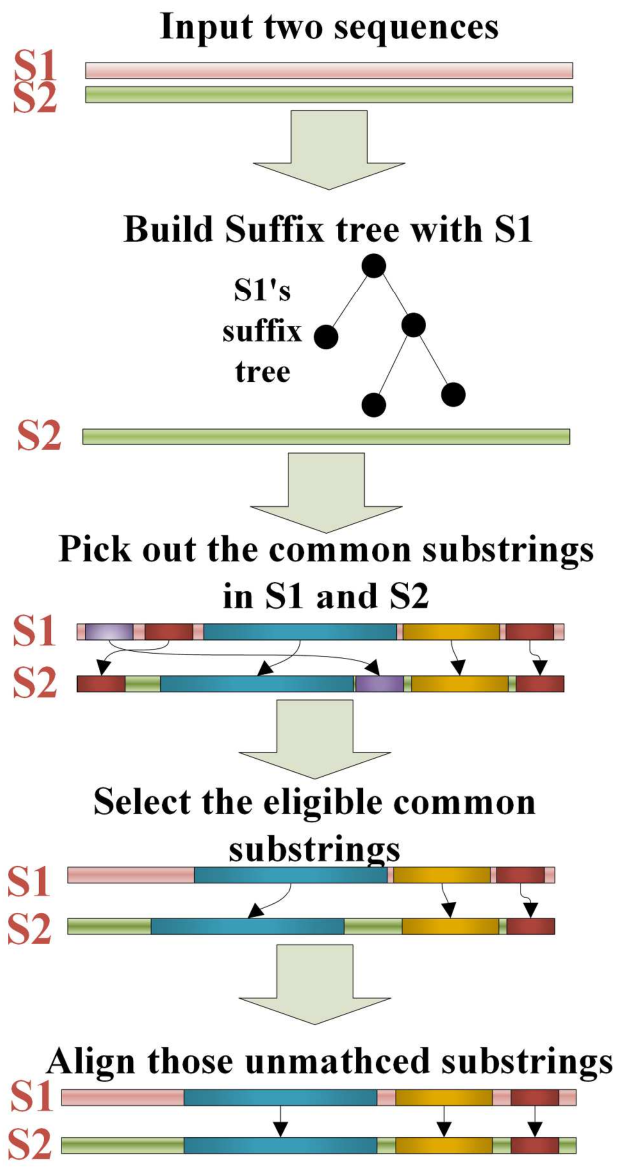
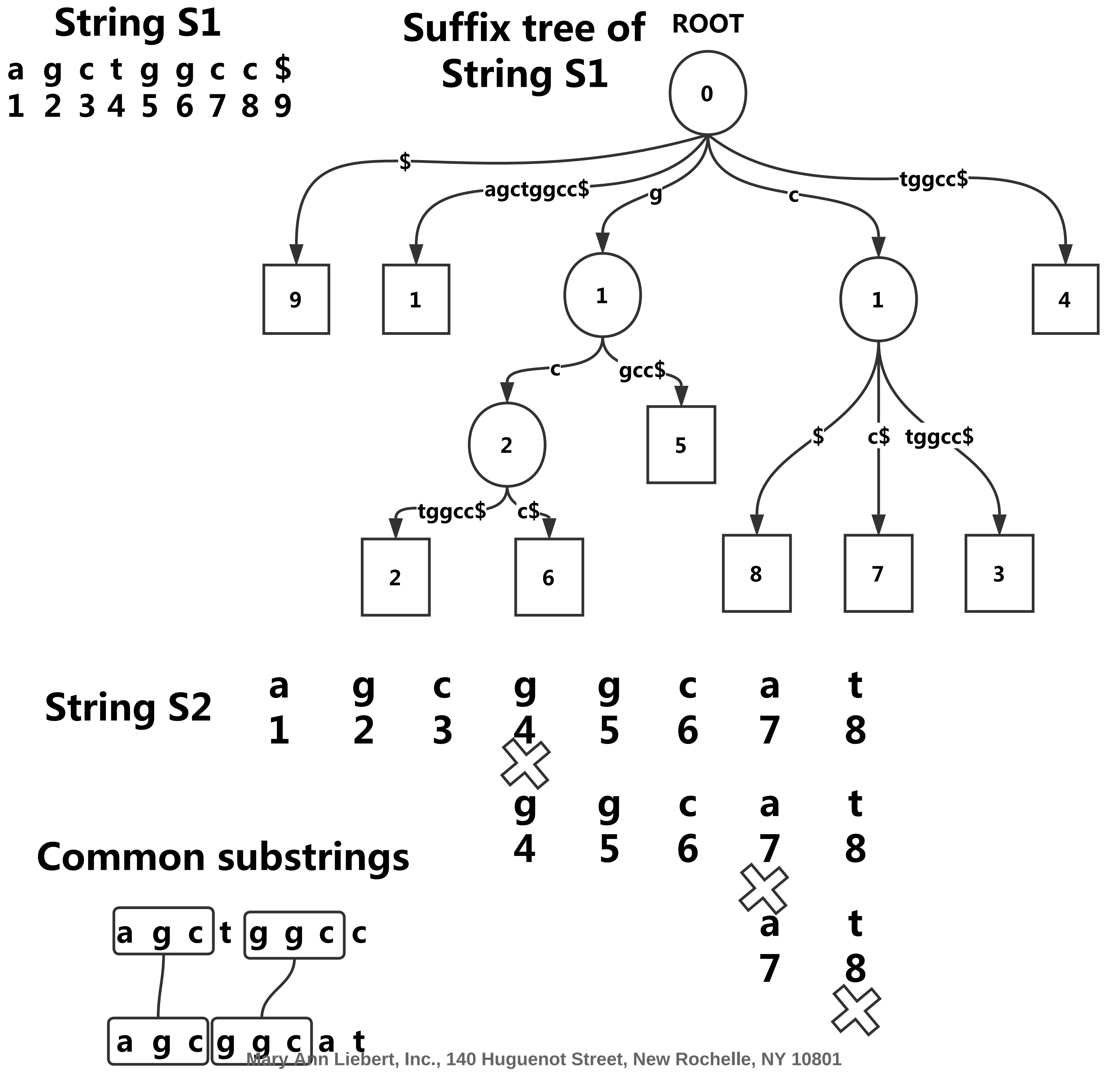

Input two sequences
S1
S2
Build Suffix tree with S1
S1's suffix tree
S2
Pick out the common substrings in S1 and S2
S1
S2
Select the eligible common substrings
S1
S2
Align those unmathced substrings
S1
S2

Figure 1 Pairwise alignment process using a suffix tree

150x277mm (300 x 300 DPI)


String S1
a g c t g g c c $
1 2 3 4 5 6 7 8 9
Suffix tree of String S1
ROOT
0
$
agctggcc$
g
c
tggcc$
9
1
1
1
4
c
gcc$
2
5
$
c$
tggcc$
tggcc$
c$
2
6
8
7
3
String S2
a g c g g c a t
1 2 3 4 5 6 7 8
g g c a t
4 5 6 7 8
a t
7 8
Common substrings
a g c t g g c c
a g c g g c a t

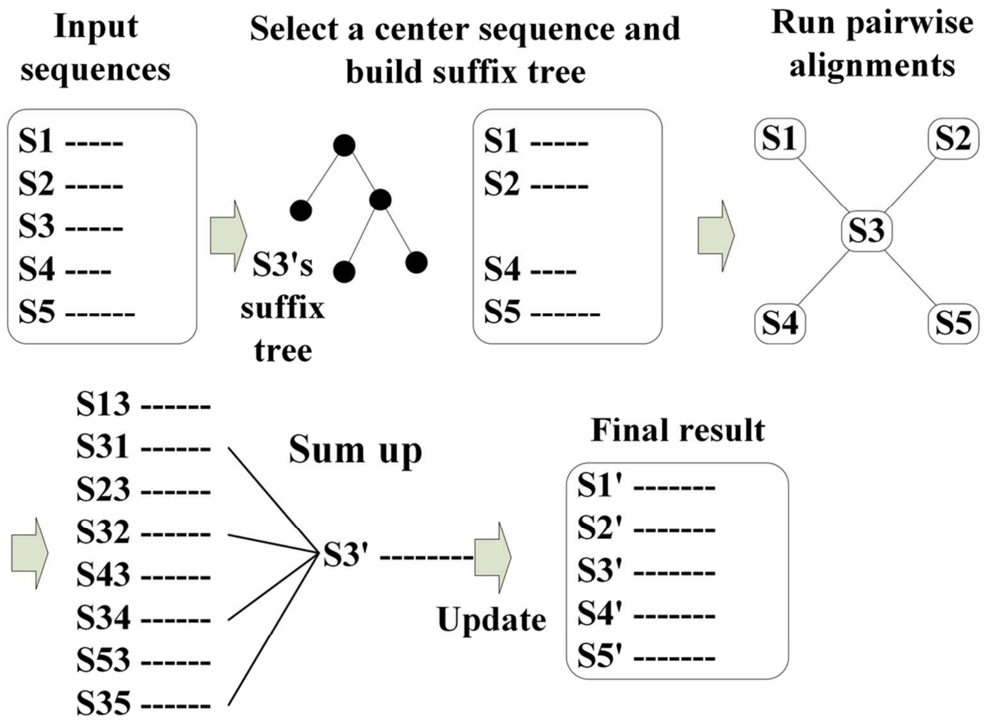


Figure 3 MASC flowchart

103x77mm (300 x 300 DPI)

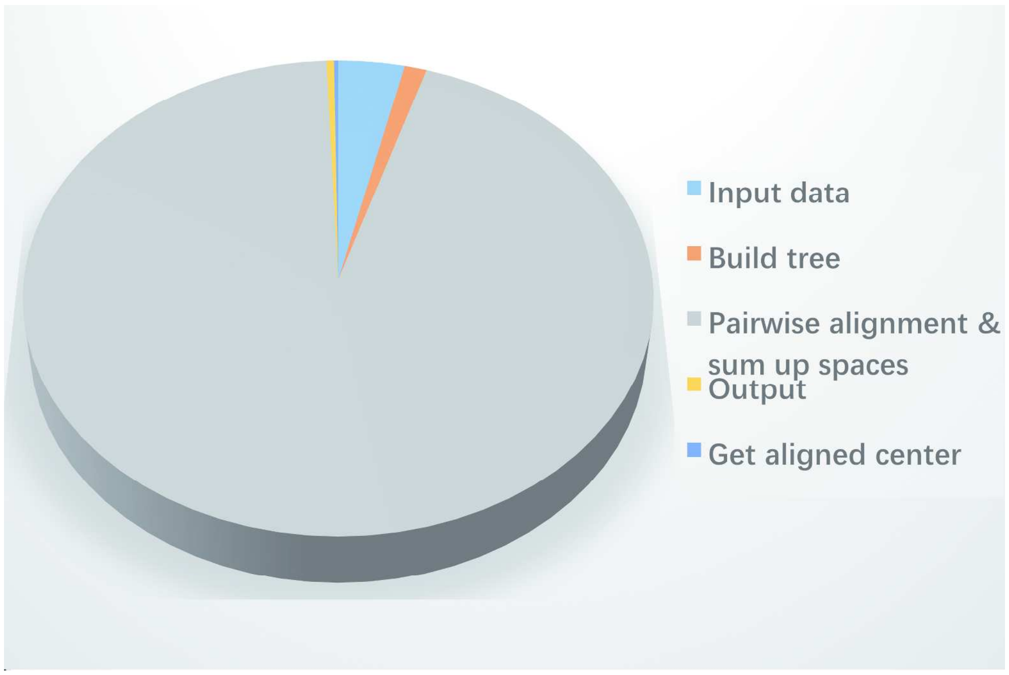
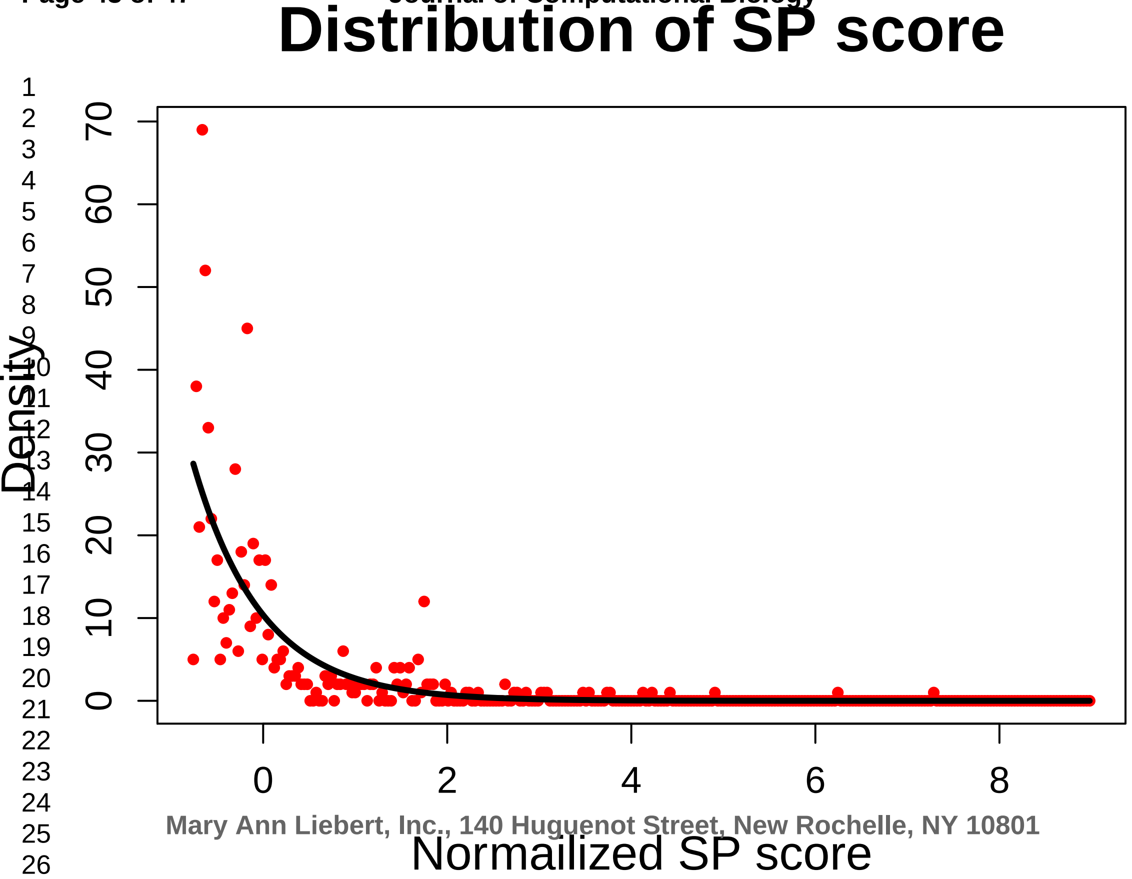


Figure 4 Time cost of MASC steps

494x328mm (144 x 144 DPI)


Distribution of SP score
Density
0
10
20
30
40
50
60
70
0
2
4
6
8
Normailized SP score

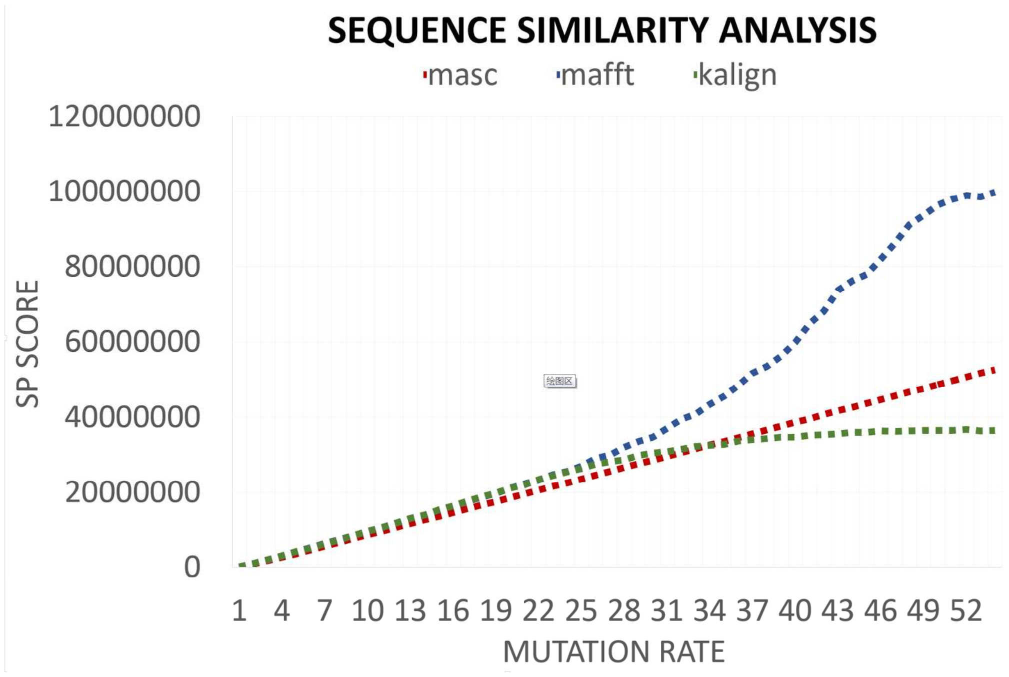


Figure 6 SP value of MASC, MAFFT and KAlign with mutation rate changed

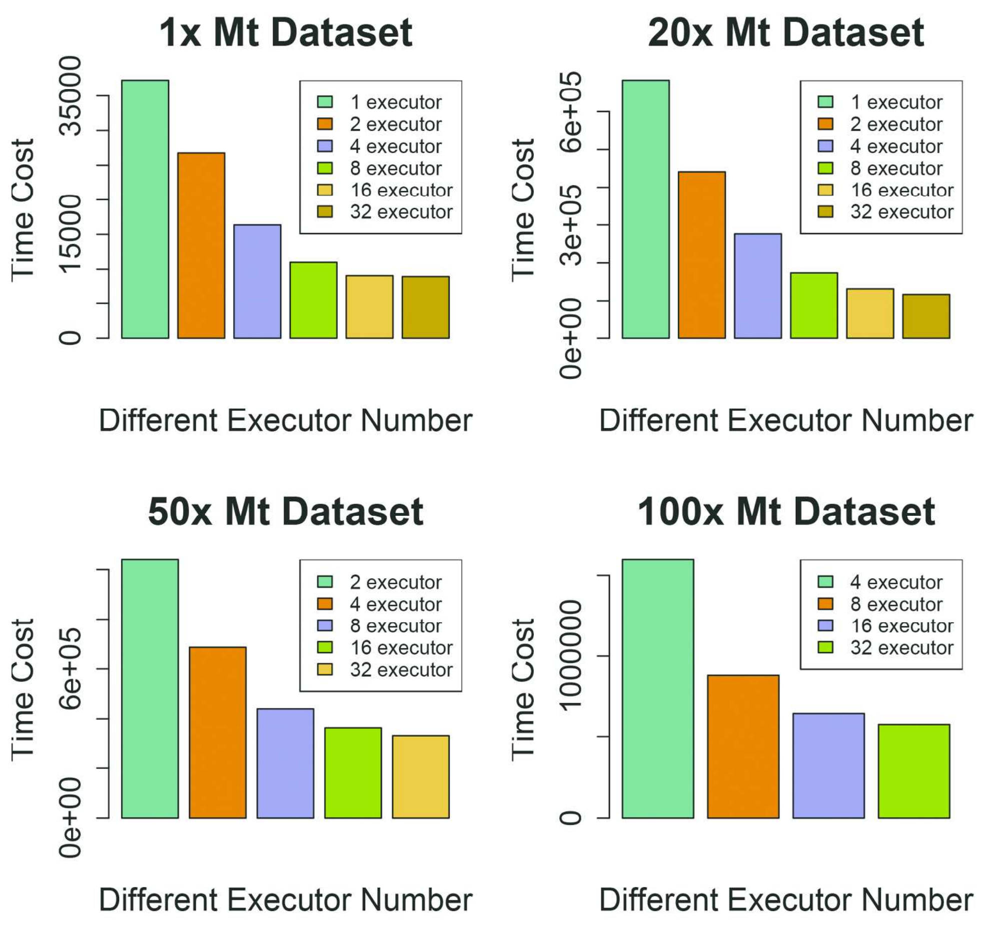
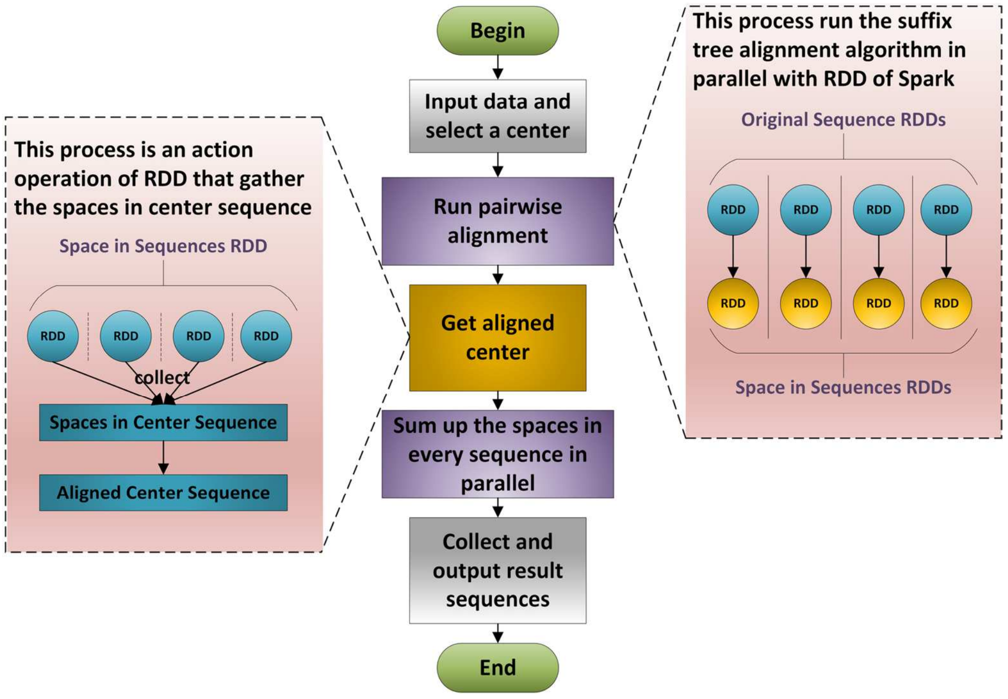


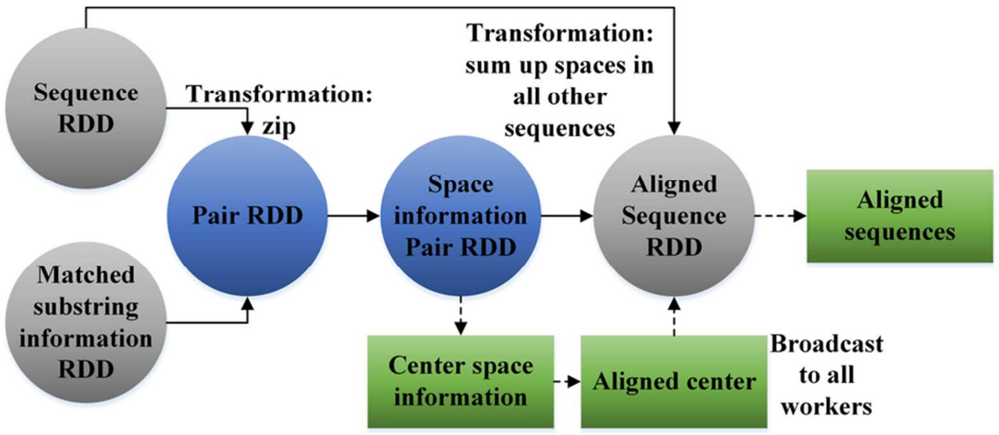
Figure 7 Running times for different mt genome datasets with different modes

177x170mm (300 x 300 DPI)


Begin
Input data and select a center
Run pairwise alignment
Get aligned center
Sum up the spaces in every sequence in parallel
Collect and output result sequences
End
This process is an action operation of RDD that gather the spaces in center sequence
Space in Sequences RDD
RDD
collect
Spaces in Center Sequence
Aligned Center Sequence
This process run the suffix tree alignment algorithm in parallel with RDD of Spark
Original Sequence RDDs
Space in Sequences RDDs

Supplementary Figure 1 Implementation of MASC on Spark

142x99mm (300 x 300 DPI)



Supplementary Figure 2 Data flow and operations

80x34mm (300 x 300 DPI)