

SCIENTIFIC REPORTS



OPEN

SPARK-MSNA: Efficient algorithm on Apache Spark for aligning multiple similar DNA/RNA sequences with supervised learning

V. Vineetha , C. L. Biji & Achuthsankar S. Nair

Multiple sequence alignment (MSA) is an integral part of molecular biology. But handling massive number of large sequences is still a bottleneck for most of the state-of-the-art software tools. Knowledge driven algorithms utilizing features of input sequences, such as high similarity in case of DNA sequences, can help in improving the efficiency of DNA MSA to assist in phylogenetic tree construction, comparative genomics etc. This article showcases the benefit of utilizing similarity features while performing the alignment. The algorithm uses suffix tree for identifying common substrings and uses a modified Needleman-Wunsch algorithm for pairwise alignments. In order to improve the efficiency of pairwise alignments, a knowledge base is created and a supervised learning with nearest neighbor algorithm is used to guide the alignment. The algorithm provided linear complexity $O(m)$ compared to $O(m^2)$. Comparing with state-of-the-art algorithms (e.g., HAlign II), SPARK-MSNA provided 50% improvement in memory utilization in processing human mitochondrial genome (mt. genomes, 100x, 1.1. GB) with a better alignment accuracy in terms of average SP score and comparable execution time. The algorithm is implemented on big data framework Apache Spark in order to improve the scalability. The source code & test data are available at: <https://sourceforge.net/projects/spark-msna/>.

Sequence alignment is used in bioinformatics to identify degree of similarity between biological sequences (DNA, RNA or protein), in understanding functional, structural and evolutionary relationship between them. Sequence alignment is of vital importance in molecular biology for studies involving molecular function prediction, evolutionary tree reconstruction and disease analysis. Needleman-Wunsch (NW) algorithm¹, was one of the first implementations of dynamic programming in bioinformatics. It was an optimal sequence alignment algorithm with a tradeoff in computational time and space. For two sequences of length m and n , the time and space complexity is computed as $O(m * n)$. By expanding the same algorithm for multiple sequence alignment (MSA), the complexity rises exponentially ($O(m^n)$ for n sequences of length m). Because of this high computational cost involved, NW algorithm cannot be used in multiple sequence alignment especially for large number of long sequences. Most popular implementations of MSA such as CLUSTAL², MAFFT³, MUSCLE⁴ use approximation methods such as progressive and iterative approaches for faster execution and less memory utilization. Most of the algorithms implemented so far were derivatives of NW algorithm.

The improvements in DNA sequencing technology has led to an unprecedented increase in the amount of DNA and genome data being available for studies. Therefore, it is important to improve the scalability and performance of MSA tools. Cloud computing and the recently emerged big data technologies such as the new programming paradigm called MapReduce⁵, are effective ways to process huge volume of data of the order of petabytes and more. Implementation using big data frameworks for sequence alignment/mapping were reported in the literature for instance, Sadasivam *et al.*, 2010⁶ presented a Hadoop based implementation of MSA using NW algorithm and Zhao *et al.*, 2015⁷ presented Spark based implementation of local alignment.

Most of these research focused on improving the scalability of MSA using Big data frameworks, but not much research has happened in improving the MSA technique as such. These implementations are able to support large

Department of Computational Biology and Bioinformatics, University of Kerala, Thiruvananthapuram, Kerala, India. Correspondence and requests for materials should be addressed to V.V. (email: vineevishnu@gmail.com)

set of input sequences, but when it comes to massive DNA sequences, they are either unable to support or execute slowly when the count of sequences increase beyond 100. DNA sequences are highly similar compared to protein sequences. This similarity feature can be utilized to improve the alignment, and enable algorithm to tackle the volume and achieve better performance. Q. Zou *et al.*⁸ developed an algorithm which is proven to be highly efficient in performing MSA of similar DNA/RNA sequences. The algorithm uses centre star strategy along with trie tree data structure to improve the performance. Spark version of this algorithm HAlign II⁹, to support large volume of sequences reported promising results for similar DNA/RNA sequence alignment. MASC (Multiple Sequence Alignment Based on a Suffix Tree and Center-Star Strategy) is the implementation of same algorithm on CUDA architecture to obtain much faster performance for ultra large data sets¹⁰. PASTASPARK¹¹ is another prominent implementation of MSA on Spark framework, which performs alignment based on SATé (Simultaneous Alignment and Tree estimation) and transitivity.

MSA could be further enhanced with a bounded dynamic programming algorithm¹² at the pairwise alignment level. DDGARM, an improved NW algorithm¹³ for pairwise alignment has proved that, for highly similar sequences, optimal alignment can be achieved by filling only 10% of the matrix. Our algorithm uses the concept of Q. Zou *et al.*, on progressive alignment, with modified NW algorithm for improved pairwise alignment. Key characteristics of the proposed algorithm include, (a) Suffix tree data structure for storing input sequences and identifying common substrings between sequences, (b) A knowledge base and nearest neighbor learning layer to guide the pairwise alignment, (c) Modified Needleman-Wunsch algorithm to perform pairwise alignments at each stage in order to reduce the memory and execution time of alignments and (d) Parallelization using MapReduce method for suffix tree construction and pairwise alignment to further improve the execution time.

Methods

Progressive Alignment. Progressive method is one of the basic alignment strategies used for MSA. It is known to provide reasonably good result and is the most widely used heuristic method for MSA¹⁴. Hence it is chosen as the core of our algorithm. The basic flow of progressive strategy is to prepare a guide tree and use series of pairwise alignments to align the sequences based on the branching order in the guide tree. The guide tree is formed based on the pairwise distance of sequences. Guide tree is formed in the order from shortest to longest distant pair. Initially the most closely related sequence pair is aligned and then the remaining sequences are aligned to the previous alignment until all sequences are aligned. Pairwise alignment is performed at each stage and it is refined at the final step while summing up the alignments. In the refinement step, the early gaps are revisited to adjust the penalties based on aligned sequences from other pairwise alignments. There are many MSA algorithms which uses modified forms of progressive methods². Details about progressive alignment method along with pseudo code is given in Supplementary Material (Data S2).

The guide tree construction and pairwise alignment are the major contributors for the execution time and memory utilization in progressive alignment method. Use of data structure such as suffix tree which enables efficient storage and quick search of common substrings of the sequences help in improving the complexity of guide tree step. Similarly, the pairwise alignments are performed using the dynamic programming approach which becomes the most time consuming process when the sequences involved in the MSA are quite large. Bounded dynamic programming algorithm is used to enhance the performance of pairwise alignments.

From the suffix tree, common substrings can be rapidly extracted for highly similar DNA sequences. This leaves only the unmatched regions to be aligned. The modified pairwise alignment algorithm also provides substantial improvement in execution time and memory utilization as the similarity among the sequences increases.

Suffix Trees to enhance alignment of similar sequences. Suffix trees greatly improve the performance of search on indexed string and hence are widely used in problems involving pattern matching, finding sub strings etc. Many existing alignment algorithms use suffix tree to identify matching substrings and there exists different algorithms for the construction of suffix tree^{15–21}. Ukkonen suffix tree construction¹⁵ is followed in the implementation as it is superior in terms of time and space complexity²².

Each input sequence is partitioned in to equal size segments and these segments are used to construct the suffix tree. Suffix tree is characterized by a root and each edge is labeled by the nucleotide in the sequence. For any node v , the string formed by concatenating the edge labels from root to v is the path to that node, $path(v)$. Suffix tree is known to provide optimal search time^{16,23}, which means, identifying the node v , which is closest to the root for a given pattern P , such that P is a prefix of $path(v)$ can be performed in time linear to the length of P . All leaves in the subtree of the node v then represent the occurrences of the pattern P in string S .

If there are n DNA sequences with an average length of m , the time complexity for building a suffix tree for one sequence is $O(m)$ (Lines 2–3 in Algorithm 1). After constructing the suffix tree, search the suffix tree for each segment of every sequence pair to identify the common substrings and matching segments. Searching the n sequences in the suffix tree costs $O(nm)$ (Lines 5–7 in Algorithm 1). For the unmatched segments, record the percentage identity and difference in length if any. Since the sequences are partitioned to equal size segments, only the last segment of every sequence will be having different length. Only the unmatched segments are considered for pairwise alignments and the features ie; percentage identity and difference in length are used to extract learning from knowledge base. The guide tree for performing pairwise alignments are formed based on the similarity measure extracted for each sequence pair.

Modified N-W algorithm for pairwise alignment. Our previous research had proved that, for pairwise alignments, optimal alignment can be achieved by populating only limited number of diagonals of the matrix¹³. The number of diagonals to be filled to obtain optimal alignment is not fixed in all cases. Hence, there is a need to find the minimum number of diagonals to be filled as a pre-requisite. This is done using dot plot approach. With some modifications to the dotlet²⁴ algorithm, the number of diagonals to be filled can be obtained. Test

results have proved that the similarity between sequences and the number of diagonals to be filled are inversely proportional. According to our previous research¹³, sequences with % identity (more than 50%) and difference in length (less than 25%) are reported to get a 50% improvement in memory utilization and execution time in pairwise alignments.

In our approach, pairwise alignment is performed only for the unmatched segments. As the similarity between sequences increases, the number of segments to be aligned reduces. The modified alignment algorithm further reduces the complexity as similarity increases. The most distant segment pair from the input sequences are chosen to identify the number of diagonals to be filled. This improves the execution time and at the same time ensures that all pairwise alignments provide optimal alignment as it would be the highest of diagonals count for the given input set. Even though this step involves only one pairwise alignment, this could become costly for very large sequences. Hence, a knowledge base is built with training data and a learning layer with nearest neighbor algorithm is used to extract knowledge out of knowledge base. With more learnings, the knowledge base becomes more accurate and would result in faster learning.

In the traditional dynamic programming based pairwise alignment, the complexity is $O(m^2)$ for sequence segments having length m . In our modified alignment approach, the complexity reduces to $O(m * k) + O(2m * d)$, where k is the difference in length and d is the number of diagonals filled. In case of highly similar sequences, $k \rightarrow 0$ and $d \ll m$, hence the complexity becomes $O(m)$ compared to $O(m^2)$ in the case of traditional dynamic programming approach where we fill the entire matrix. The worst case complexity would be $O(m^2)$ as $2m * d$ becomes equal to m^2 , when the similarity between sequences decreases.

Algorithm 1. (Main flow):

Input: n DNA/RNA sequences $S_1, S_2, S_3, \dots, S_n$

Output: n aligned DNA/RNA sequences $S_1^*, S_2^*, S_3^*, \dots, S_n^*$

1. **For** each input sequence S_i ,
 2. Partition the sequence in to multiple equal length segments $S_{i1}, S_{i2}, S_{i3}, \dots, S_{in}$
 3. Store the sequence as segments in HDFS for worker nodes to fetch data from HDFS and construct RDDs in feature extraction and alignment processes. RDDs are iteratively used in the alignment step.
 4. Construct a suffix tree for the sequence segments using Algorithm 3 – The suffix tree is used as broadcast variable for the feature extraction in steps 5-10. (Unmatched segments are cached in memory with RDD persistence using storage level MEMORY_ONLY)
 5. **For** $j = 1$ to $n, j \neq i$,
 6. Search the suffix tree for the segments and record the matched and unmatched segments.
 7. Store the lowest percentage identity for each sequence pair.
 8. Compare the length of the unmatched segments and record the difference.
 9. Store the highest measure for each sequence pair.
 10. **end For**
 11. Check if any entry similar to the highest length difference and lowest identity exists in the knowledge base.
 12. If not, perform the learning and construct an entry for the knowledge base
 13. If yes, use that value for the pairwise alignments.
 14. Construct a guide tree based on the similarity measure of sequences in the order of most similar to distant ones. (Suffix tree broadcast variable is destroyed to free up memory for constructing guide tree and use as broadcast variable in the alignment step)
 15. Align the sequences by progressive method using the guide tree. Aligned sequence segments are saved to use in the future iterations.
 - a. Perform pairwise alignment of the most closely related sequences
 - ii. Get the aligned sequences
 - iii. Find the two next-most closely related sequences from the guide tree
 - b. Iterate until all sequences have been aligned
 16. Form the final sequence by refining and expanding the aligned sequences with the original sequences
 - a. For each sequence S_i , form suffix tree using the aligned sequences of S_i . Now best alignment for S_i is selected by realigning the subtrees which involves gaps.
-

Supervised learning layer. Bounded dynamic programming for pairwise alignment is the key in our approach to achieve better performance. From the experiment results¹³, it is known that the number of diagonals to be filled depends on the similarity level and difference in length. Prior knowledge about the number of diagonals to be filled is a pre-requisite for the pairwise alignment step. Using training dataset, a knowledge base is built with the mapping of sequence similarity to number of diagonals. Sequence similarity measure (percentage identity and difference in length) for the most distant segments are used for extracting the knowledge from knowledge base. Nearest neighbor algorithm is used to identify the best matching entry from the knowledge base^{25,26}. Less

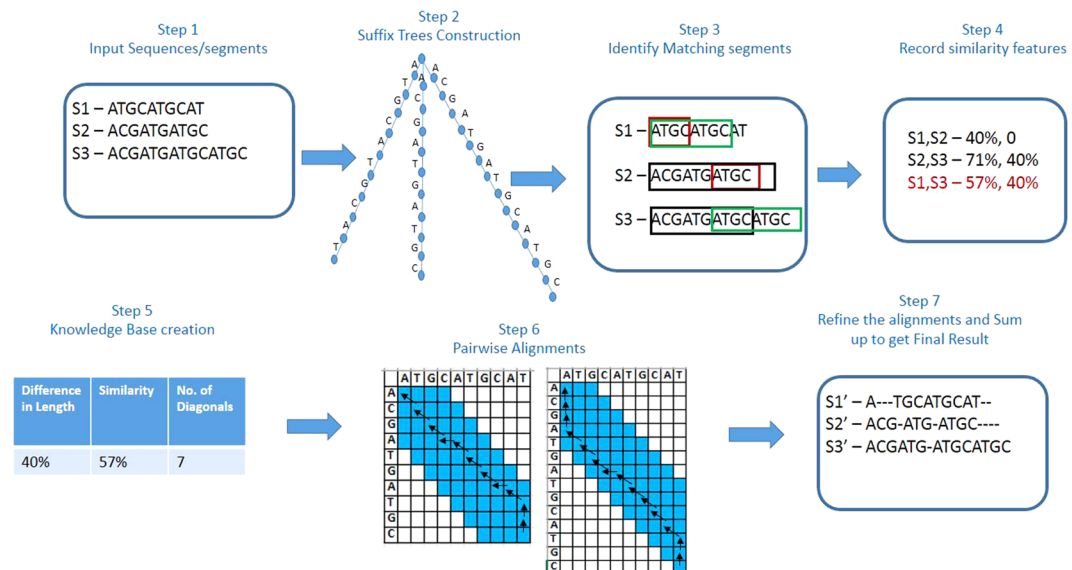


Figure 1. Sample flow of SPARK-MSNA algorithm.

number of dimensions (percentage identity and length difference) for pattern recognition was the driving behind selecting nearest neighbor as the learning algorithm. More details about nearest neighbor algorithm is given in Supplementary Material (Data S3).

Algorithm 2. (Knowledge Base creation/learning).

1. Search the suffix tree to identify the matched and unmatched segments for every sequence pair.
 2. Record the lowest percentage identity and highest difference in length for the unmatched segments of every sequence pair.
 3. Check if an exact match or closest match for the extracted features exist in the knowledge base
 4. If yes, use the learning from the knowledge base.
 5. Else, Perform the modified dotlet algorithm to get the number of diagonals to be filled for getting the optimum alignment.
 6. Make an entry in to the knowledge base with features as percentage identity and length difference
-

Algorithm 3. (Nearest Neighbor).

For $i = 1$ to n ,

Find t_m in the set, such that $d(t_m, t_i)$ is the smallest.

If $d(t_m, t_i) < \theta$

Use t_m as the best match and fetch the details

Else

Create new entry to knowledge base

The learning layer uses percentage identity and length difference as the features for classifying the input sequences. For each sequence pair, these features are extracted and the combination of highest value for difference in length and lowest value for identity are chosen for an input dataset. Then, it is matched with the knowledge base to identify the closest set. The algorithm initially checks for the exact match and in case of absence of exact match checks for the closest match (within a range of $\pm(2-3)\%$). Count of diagonals will be fetched for this closest match and that will be used for the pairwise alignment in the progressive MSA. Each time a new set of features are encountered, for which a closest match does not exist in the knowledge base, dotlet algorithm is executed to identify the number of diagonals. This learning is then entered in to the knowledge base for future alignments. More entries in the knowledge base would improve the performance and accuracy of the alignment. Figure 1 shows the flow of the algorithm with sample data.

Parallel implementation with Spark. Parallel computation is implemented using MapReduce model at two stages in the algorithm. The suffix tree construction and the pairwise alignment of progressive method. MapReduce can be implemented using Hadoop or Spark. Due to the additional improvement in time provided

Difference in Length of sequences (%)	Similarity (%)	percentage of diagonals filled in the 2×2 matrix
0.30	99.20	0.15
0.20	99.30	0.14
0.40	98.00	0.16
0.30	98.10	0.15
0.10	95.90	0.18
0.23	97.30	0.17
0.34	98.20	0.16
0.35	96.40	0.18
0.70	99.00	0.4
1.20	99.00	0.3
5.80	75.00	6.2
25	50.00	20

Table 1. Sample knowledge base constructed for testing.

	Similarity (%)	Execution time (without knowledge base)	Execution time (with knowledge base)
Dataset 1	95	1 min 11 sec	50 sec
Dataset 2	70	1 min 31 sec	1 min 4 sec
Dataset 3	45	1 min 47 sec	1 min 14 sec
Dataset 4	35	2 min 5 sec	1 min 29 sec
Dataset 5	20	2 min 43 sec	1 min 55 sec

Table 2. Execution time taken by SPARK-MSNA for datasets with different similarity. Datasets were of equal size (3.75MB).

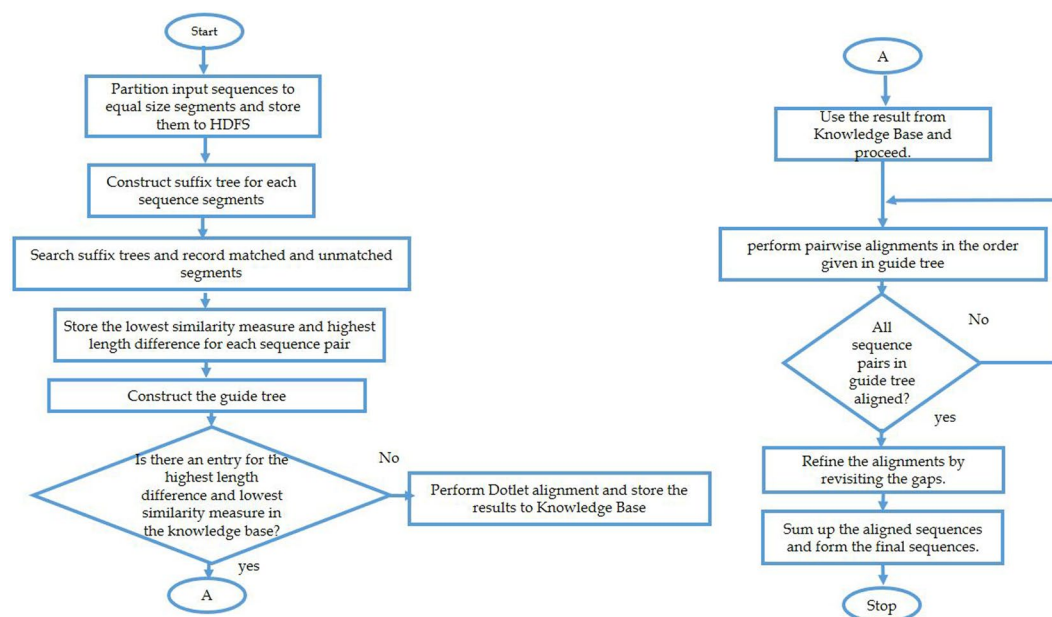


Figure 2. Flow chart of SPARK-MSNA algorithm.

by spark with its in-memory computation, spark is chosen as the MapReduce framework²⁷. More details about MapReduce programming model is given in Supplementary Material (Data S4).

Although usage of suffix tree with Ukkonen's algorithm results in linear time complexity, this could be costlier when sequences involved are quite large in size. Performance is further improved with parallel construction using MapReduce programming model²⁸. The suffix tree is partitioned vertically and each partition is constructed independently. The prefixes generated from the vertical partitioning forms the key and its starting positions form the value. This key-value pair is processed using the map task and subtrees are constructed in parallel by compute

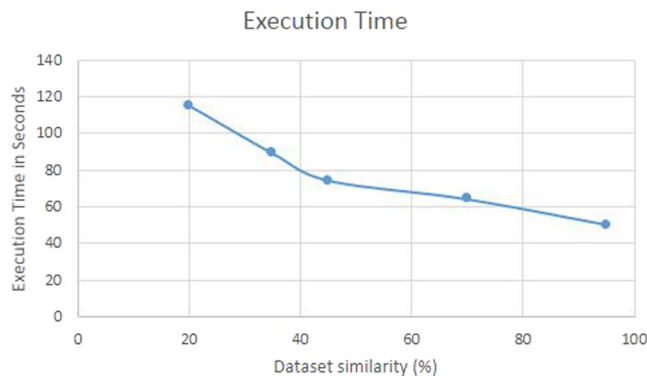


Figure 3. Execution time of SPARK-MSNA decreases as similarity of input sequences increase.

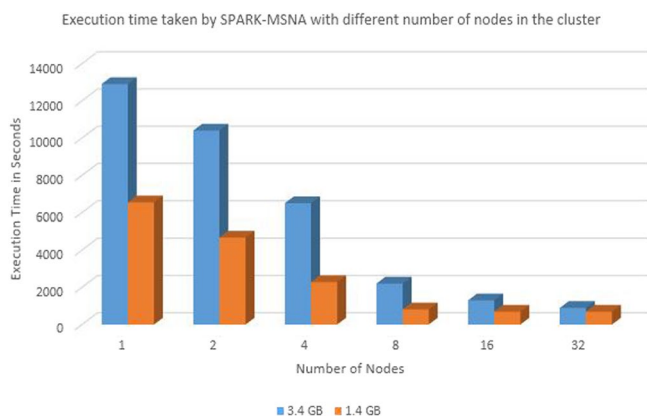


Figure 4. Improvement in execution time of SPARK-MSNA with more number of nodes.

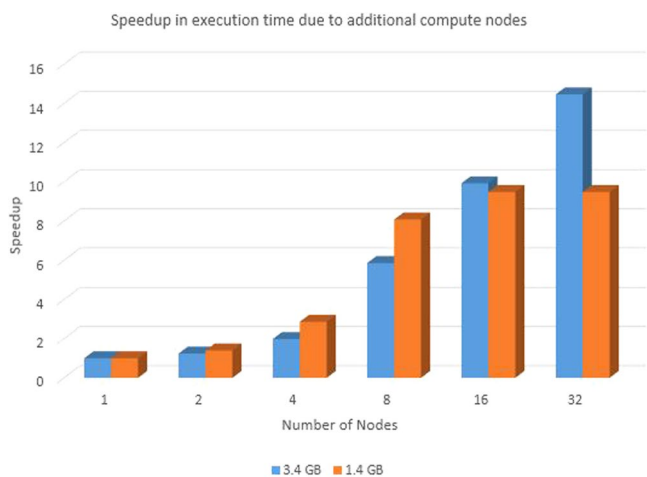


Figure 5. Speedup in execution time due to additional compute nodes.

nodes. Suffix tree construction from the subtree is combined with the map tasks in order to reduce the overhead of shuffle and reduce. Algorithm 4A shows the flow for the map function of suffix tree construction.

The pairwise alignment stage checks for matched segments and the unmatched segments alone are then taken for pairwise alignment. Pairwise alignment of segments is then executed in parallel using MapReduce model. Name of the sequence with segment index is the key and the sequence segment is the value for this map phase. Each compute node then performs the pairwise alignment using the modified pairwise algorithm. Result is then passed in the form of key-value pair where key is the sequence name with segment index and value is the aligned sequence. Aligned sequence segments for one pair of sequences are combined with the map task to avoid the overhead of reduce task. Algorithm 4B shows the flow of the map function for pairwise alignment.

Algorithm 4A. (Map function for Suffix tree construction).

ForEach Map_1 (Key = S-prefix, value = frequency)

 Implement Ukkonen's algorithm to construct sub tree

 Key = S-prefix

 Value = constructed sub tree

 Map.collect(toMap(Map.Entry::getKey, Map.Entry::getValue));

End For

Algorithm 4B. (Map function for pairwise alignment).

ForEach Map_1 (Key = sequence_pair_index_segment[i], value = unmatched sequence segments of the identified pair)

 Key = sequence_pair_index_segment[i]

 Value = aligned sequence of the unmatched segment (based on the modified N-W algorithm with reduced matrix calculation)

 Map.collect(groupingBy(Map.Entry::getKey, Map.Entry::getValue));

End For

Parallel execution does not improve the complexity of the algorithm but it helps in improving the execution time. When we have number of compute nodes equal to or more than the number of partitions to be processed, the execution time is equivalent to that of processing single partition plus an additional overhead for the reduce phase to construct the final result. In case the number of compute nodes are less, partition groups are formed and process the partition groups in parallel, for improved performance compared to sequential run. Spark framework reduces the network overhead by utilizing the data locality concept of MapReduce, but merging the scattered intermediate results to form the final result will always be there. But in case of large datasets, this additional overhead is much lower compared to the sequential execution or traditional distributed computing (OpenMP/MPI). Figure 2 shows the flow chart of the algorithm.

Results and Discussion

Test results on simulated data. Performance of the algorithm was tested using simulated data and couple of actual data sets. MSA algorithms supporting massive genome sequences are still in the evolving phase and hence there is a lack of benchmark dataset when it comes to large scale DNA MSA algorithms. The Balibase²⁹ dataset which is considered as the golden benchmark for MSA is more suitable for protein sequences and does not provide benchmark for large DNA sequences. Details about real-world applications of MSA involving large genome sequences are given in Supplementary Material (Data S1). Simulated data with different levels of similarity was used to test the effectiveness of the algorithm. Test data was created by taking a portion of the human mitochondrial genome (NC_012920.1) as first sequence and then creating the second sequence with some random modifications in the first sequence. The similarity between sequences were first tested with traditional NW algorithm for correctness. Sequence datasets were prepared with 95%, 70%, 45%, 35% and 20% similarity and fixed size of 3.75 MB with maximum length 6580 bp and minimum length 6560 bp. Training data with 50 datasets of varying similarity range was prepared to build the knowledge base. Table 1 shows the snapshot of the knowledge base created for the test data.

Prepared datasets were used to test the algorithm and identify the relationship of performance with similarity level. Testing was performed with and without knowledge base. Whenever the learning layer failed to match input data with existing entries in knowledge base, the algorithm performs dot plot step to gain the knowledge and make an entry in knowledge base. Table 2 shows the result of testing with simulated data. The results indicate that the algorithm delivers better performance as similarity between the input sequences increases. Figure 3 shows the decrease in execution time as similarity among input sequences increases. This is because, the number of segments to be aligned and the diagonals to be filled for alignment reduces as the similarity increases. With the knowledge base, the execution time and memory utilization reduces further as we do not have to perform the modified dotlet alignment to find out the number of diagonals to be filled. With large sequence data, the reduction in time due to removal of dotlet alignment would be more significant.

Comparison with other tools. Most of the MSA algorithms compare the test results with other commonly used tools/algorithms. Test results of SPARK-MSNA are compared with HAlign, HAlign II and PASTASPARK. MAFFT & MUSCLE are used for comparing results of small data sets. Even though MASC has reported highly competitive performance in handling large volume of data, the underlying architectures are different for both

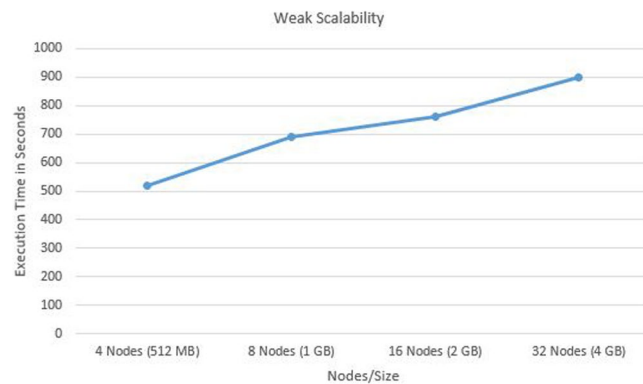


Figure 6. Weak scalability of SPARK-MSNA.

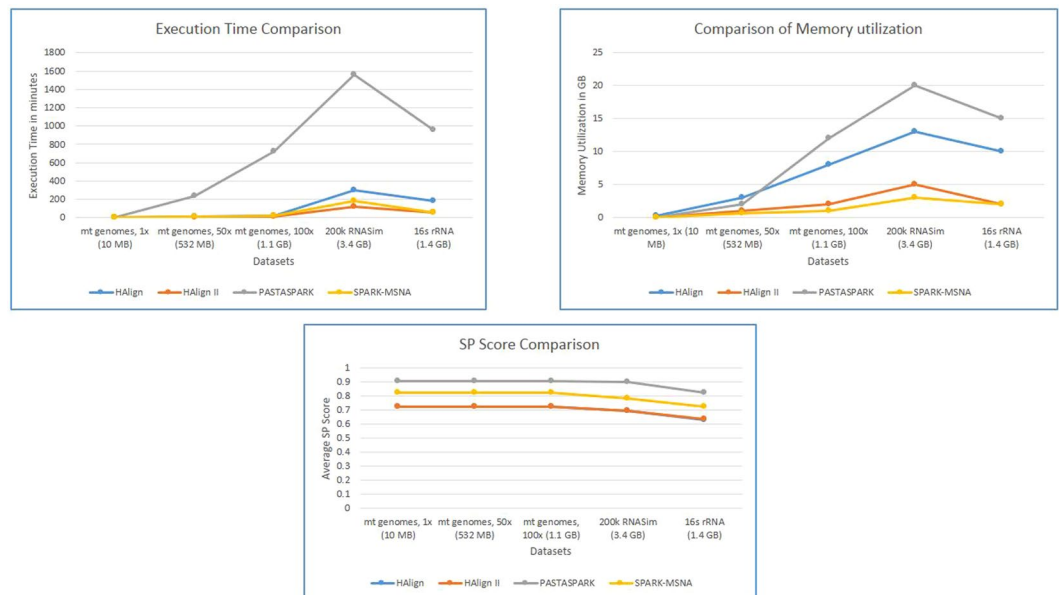


Figure 7. Performance comparison of SPARK-MSNA with other algorithms.

implementations, as MASC is implemented on CUDA processor. The algorithms HAlign⁸ and HAlign II⁹ have reported considerable improvement in performance specifically in addressing large scale DNA sequence data and our work has been inspired from HAlign; so, we have used same datasets used by HAlign to test the performance of our algorithm – human mitochondrial genomes (mt genomes) and 16 s rRNA. Dataset from PASTASPARK 200k RNASim is also included in the test data.

The human mitochondrial genome dataset is a sample for highly similar dataset. The dataset contains 672 human mitochondrial genomes with maximum length 16579 bp and minimum length 16556 bp. The percentage identity is >97% for this dataset. 200k RNASim dataset is used as dataset with moderate level of similarity with minimum sequence length as 748 and maximum sequence length as 1836. 16 s rRNA dataset is used for testing the performance on less similar sequence set. It has minimum length 807 bp and maximum length 1629 bp. Details of test datasets are provided in Table S1 of Supplementary Material. In order to compare results with other tools, tests are performed on single node cluster and multi node cluster. Spark cluster was set up on single node with 3.6 GHz 4 core CPU, 64 bit Ubuntu OS and 64 GB memory. In order to test the improvement due to parallel implementation, SPARK-MSNA was tested with more number of nodes. Figure 4 shows the execution time taken by SPARK-MSNA with 1, 2, 4, 8, 16 and 32 nodes. Large data sets of 1.4 GB and 3.4 GB are used for testing the improvement in execution time with number of nodes. Multi node cluster set up is used to compare performance with HAlign II. A cluster of 12 servers with intel Xeon E5-2620 processor with 8 cores and Spark 2.3.0 were used for the testing. Figure 5 shows the speedup of execution time due to additional nodes and Fig. 6 shows the weak scalability of the algorithm. Table S2 of Supplementary Material shows the test results of various algorithms using test datasets. Figure 7 shows the performance comparison of SPARK-MSNA with HAlign, HAlign II and PASTASPARK. MAFFT & MUSCLE had limitations in processing datasets of size more than 1 GB. Considering the high volume of genome sequence data generated by NGS techniques and the predicted transition to personalized and precision medicine, there is a pressing need on MSA tools/algorithms to support data sets of hundreds

of GBs/TBs. SPARK-MSNA provided better optimum results with better memory utilization & average SP score compared to HAlign II with slightly high execution time.

Average SP score is used for comparing the alignment accuracy. SP score is calculated as the number of pairs of residues correctly aligned. The score is calculated as

$$\frac{\sum_{i=1}^M S_i}{\sum_{i=1}^M S_{ri}}$$

where, M is the length of the sequence, M_r is the length of the reference sequence, S_i is the score of the i th column and S_{ri} is the score of i th column in reference sequence. S_i is calculated as normalized total score of column i , with pair value calculated as 2 if residues are aligned, 1 if one of the alignments has a gap and 0 otherwise. The test results show that SPARK-MSNA performs better in terms of memory utilization and accuracy, but has increased execution time compared to HAlign II. Progressive alignment and the refinement step increases the execution time, but that helps in achieving a better alignment. The reduced matrix alignment guided by knowledge base leads to reduction in memory utilization. MAFFT and MUSCLE provide better average SP score compared to SPARK-MSNA, but they are unable to handle large volume of dataset. MAFFT and MUSCLE failed to deal with datasets of size more than 1 GB. PASTASPARK is able to handle the large volume of data, but the execution time is much higher compared to HAlign II and SPARK-MSNA.

The modified Needleman-Wunsch algorithm for pairwise alignment of unmatched segments plays a crucial role in reducing the memory utilization for SPARK-MSNA. In the pairwise alignment step, HAlign II uses complete 2×2 matrix for alignment, whereas, SPARK-MSNA uses limited diagonals (approx. 2% of diagonals) to calculate the alignment. This modification provides up to 50% reduction in the memory utilized (depending on sequence similarity)¹³. The trade-off is between execution time and alignment accuracy. SPARK-MSNA provides a better balance between the two by providing an optimum performance in terms of computational time and memory while retaining an average SP Score close to MAFFT.

The knowledge base guides the bounded dynamic programming for pairwise alignment. Hence, a rich knowledge base results in better performance and better accuracy. For highly similar sequences, the improvement is highly significant as very less number of matrix cells (diagonals) are included in the alignment. As the similarity decreases, the number of matrix cells needed in the alignment increases and for highly different sequences, complete matrix is needed in the alignment, which makes it similar to normal dynamic programming. This is evident in the test result of 16s rRNA dataset, where the memory utilization is similar to that of HAlign II.

In order to test the efficiency of knowledge base, we added the knowledge base layer (training layer) to HAlign II and tested the same using mt. genome and 16s rRNA datasets. Table S3 of Supplementary Material shows the test results. Results show that, knowledge driven bounded dynamic programming helps in achieving improved execution time and memory utilization. Average SP score remains same as HAlign II as the underlying alignment strategy remains the same in both algorithms (Centre star). This shows the importance of knowledge driven algorithms, which can learn from their experiences are key to improving the performance of MSA.

Complexity Analysis. The most important feature of SPARK-MSNA is the improved time and space complexity. The first stage of SPARK-MSNA is construction of Suffix tree. The Ukkonen's algorithm using MapReduce model is adapted here. For one DNA sequence of length m , the time complexity for building suffix tree is $O(m)$. For n sequences, the complexity is $O(nm)$. The second stage is searching the suffix tree for all possible pair combinations of n DNA sequences. The search would incur a cost of $O(m)$ for one sequence pair and we have $nC2$ pairs. So, the complexity of search becomes $O(n^2m)$. Building guide tree based on the similarity measure obtained from search is the next step. This incurs a linear cost of $O(n)$. Due to less number of dimensions and cardinality involved in pattern matching, the complexity of learning layer becomes $O(k)$.

Pairwise alignment of unmatched segments of sequence pairs is the next step. Since we are adopting the modified Needleman-Wunsch algorithm, the complexity becomes $O(kx) + O(2kd)$, where k is the segment length, x is the difference length of segments involved in pairwise alignment and d is the number of diagonals to be populated. For highly similar sequences, $x \rightarrow 0$ and $d \ll k$, hence the complexity will be $O(k)$. In traditional dynamic programming approach, the complexity is $O(k^2)$. The last step of summing up the alignment results to form the final alignment would incur a cost of $O(nm)$. The learning step involves one pairwise alignment of the most distant segment pair and it incurs a cost of $O(k^2)$, if the learning is not available in the knowledge base.

Building the knowledge base is not part of the main flow of the algorithm. It is part of the training phase and hence it does not add to the overall complexity of the algorithm. Whenever the appropriate learning is missing in the knowledge base, the learning step is implemented to enhance the knowledge base. Modified dotlet algorithm is performed to get the number of diagonals. The alignment is performed on the most distant segment of sequences and in such scenarios, there would be an additional $O(m^2)$ added to the complexity of the algorithm, where m is the sequence length.

The overall complexity of SPARK-MSNA is $O(m) + O(n^2m) + O(n) + O(k) + O(k) + O(nm)$. Considering $n \ll m$, the best case complexity is $O(m)$. As the similarity between sequences decreases, number of unmatched segments and the number of diagonals to be populated for alignment increases, this will make the worst case complexity as $O(m^2)$. Same is the case when learning step is involved.

Discussion

In this work, we have focused on improving the efficiency of MSA involving large DNA sequences by utilizing its similarity feature and improving the performance with learning layer and parallel execution. The test results and complexity shows that, SPARK-MSNA provides a better trade-off compared to other MSA tools/algorithms in handling similar large scale DNA/RNA sequences. SPARK-MSNA provides better alignment and memory

utilization with a comparable execution time with large sequences. In best case scenario, SPARK-MSNA reduces the memory utilization up to 50% along with better alignment compared to HAlign II. In worst case scenario, where we cannot reduce the number of matrix cells to be processed in the pairwise alignments, the complexity remains similar to HAlign II. Test results with learning layer added to centre star approach shows that a knowledge driven approach helps in improving the performance in terms of execution time and memory. Knowledge driven algorithms, which can learn from experience and use the learnings in future alignments are instrumental in handling large scale datasets.

The proposed knowledge base uses only similarity feature for learning. Adding more features in knowledge base and alignment approach to utilize those additional features could provide a better result in future. RDD persistence using kyro serialization instead of raw data format for improved memory utilization is also planned as a future enhancement.

References

1. Needleman, S. & Wunsch, C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48**, 443–453 (1970).
2. Thompson, J., Higgins, D. & Gibson, T. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research* **22**, 4673–4680 (1994).
3. Katoh, K. & Standley, D. MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability. *Molecular Biology and Evolution* **30**, 772–780 (2013).
4. Edgar, R. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* **32**, 1792–1797 (2004).
5. Dean, J. & Ghemawat, S. MapReduce. *Communications of the ACM* **51**, 107 (2008).
6. Sadasivam, G. & Baktavatchalam, G. A novel approach to Multiple Sequence Alignment using hadoop data grids. *International Journal of Bioinformatics Research and Applications* **6**, 472 (2010).
7. Zhao, G., Ling, C. & Sun, D. SparkSW: Scalable Distributed Computing System for Large-Scale Biological Sequence Alignment. *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, <https://doi.org/10.1109/ccgrid.2015.55> (2015).
8. Zou, Q., Hu, Q., Guo, M. & Wang, G. HAlign: Fast multiple similar DNA/RNA sequence alignment based on the centre star strategy. *Bioinformatics* **31**, 2475–2481 (2015).
9. Wan, S. & Zou, Q. HAlign-II: efficient ultra-large multiple sequence alignment and phylogenetic tree reconstruction with distributed and parallel computing. *Algorithms for Molecular Biology* **12** (2017).
10. Su, W., Liao, X., Lu, Y., Zou, Q. & Peng, S. Multiple Sequence Alignment Based on a Suffix Tree and Center-Star Strategy: A Linear Method for Multiple Nucleotide Sequence Alignment on Spark Parallel Framework. *Journal of Computational Biology* **24**, 1230–1242 (2017).
11. Abuín, J. M., Pena, T. F. & Pichel, J. C. PASTASpark: multiple sequence alignment meets. *Big Data. Bioinformatics* **33**, 2948–2950 (2017).
12. Bellman, R. On the Theory of Dynamic Programming. *Proceedings of the National Academy of Sciences* **38**, 716–719 (1952).
13. Vineetha, V. & Nair, A. S. DDGARM: Dotlet Driven Global Alignment with Reduced Matrix. *International Journal of Advanced Research in Computer Science and Software Engineering* **7**, 70–74 (2017).
14. Feng, D.-F. & Doolittle, R. F. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution* **25**, 351–360 (1987).
15. Ukkonen, E. On-line construction of suffix trees. *Algorithmica* **14**, 249–260 (1995).
16. McCreight, E. M. A Space-Economical Suffix Tree Construction Algorithm. *Journal of the ACM* **23**(2), 262–272 (1976).
17. Farach-Colton, M., Ferragina, P. & Muthukrishnan, S. On the sorting-complexity of suffix tree construction. *Journal of the ACM* **47**, 987–1011 (2000).
18. Hunt, E., Atkinson, M. P. & Irving, R. W. A database index to large biological sequences. *work* **26**, 27 (2001).
19. Bedathur, S. & Haritsa, J. Engineering a fast online persistent suffix tree construction. *Proceedings. 20th International Conference on Data Engineering*, <https://doi.org/10.1109/icde.2004.1320040>.
20. Cheung, C.-F., Yu, J. X. & Lu, H. Constructing suffix tree for gigabyte sequences with megabyte memory. *IEEE Transactions on Knowledge and Data Engineering* **17**, 90–105 (2005).
21. Phoophakdee, B. & Zaki, M. J. Genome-scale disk-based suffix tree indexing. *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD 07*, <https://doi.org/10.1145/1247480.1247572> (2007).
22. Gusfield, D. Linear-time construction of suffix trees. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology* (1997).
23. Weiner, P. Linear pattern matching algorithms. Switching and Automata Theory. SWAT'08. IEEE Conference Record of 14th Annual Symposium on, 1–11 (1973).
24. Junier, T. & Pagni, M. Dotlet: diagonal plots in a Web browser. *Bioinformatics* **16**, 178–179 (2000).
25. Muja, M. & Lowe, D. G. Fast Approximate Nearest Neighbors With Automatic Algorithm Configuration. *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications*, <https://doi.org/10.5220/0001787803310340> (2009).
26. Kanungo, T. *et al.* An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**, 881–892 (2002).
27. Zaharia, M. *et al.* Spark: Cluster computing with working sets. *HotCloud*, 10.10-10, 95 (2010).
28. Satish, U. C., Kondikoppa, P., Park, S.-J., Patil, M. & Shah, R. MapReduce based parallel suffix tree construction for human genome. *2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, <https://doi.org/10.1109/padsw.2014.7097867> (2014).
29. Thompson, J. D., Koehl, P., Ripp, R. & Poch, O. BALiBASE 3.0: Latest developments of the multiple sequence alignment benchmark. *Proteins: Structure, Function, and Bioinformatics* **61**, 127–136 (2005).

Acknowledgements

This work has been supported by State Inter University Centre of Excellence in Bioinformatics (SIUCEB), AiCADD and DBT-BIF.

Author Contributions

V.V. designed and prepared the source code of the algorithm and drafted the manuscript. B.C.L. and A.S.N. guided the research with contributions to the design of algorithm, helped designing the manuscript and revised it critically.

Additional Information

Supplementary information accompanies this paper at <https://doi.org/10.1038/s41598-019-42966-5>.

Competing Interests: The authors declare no competing interests.

Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

© The Author(s) 2019