# Multiple Sequence Alignment for DNA Storage Clusters

Duna Mazzawi
Sima Qudsi
Butross Dallah
Project in DNA Data Storage — 236503
Department of Computer Science
Technion — Israel Institute of Technology

## Abstract

*Multiple Sequence Alignment* refers to the process of aligning a cluster of strands, usually protein or DNA, to achieve a maximal regions of similarity.
More specifically, given a set *S* of *m* erroneous strands of DNA with different lengths, that assumingly originated from one common reference, the outcome of the MSA algorithm, is to be *m* strands with gap insertions into each strand, such that all conform to a length $L \geq \max\{n_i | |S_i| = n\_i\}$, and no index $0 \leq i \leq L$, yields a column consisting of only gaps, and the alignment maximizes the common substrings of the strands. MSA has been shown to be NP-complete problem.

This work shows a new *Sequence Alignment* method, that leverage existing sequencing algorithms, in order to achieve a speedup of x80 over state of the art Multiple Sequencing Alignment algorithms.
The method uses an existing pairwise alignment algorithm called FOGSAA [1], as the base of an Iterative Algorithm, along with two other phases.
Further applications and Enhancements of the suggested method can further contribute to Modify the sequencing and alignment phases, in order to achieve a robust and efficient DNA data storage system.

## Introduction

In our modern days, we are producing data at an exponential rate, those tremendous amounts of data are to be stored in digital systems that require respectively significant space and infrastructure maintenance, thus creating high demand on a new innovative storage system.

DNA is an excellent medium for data storage, due to its demonstrated information density of petabytes of data per gram.

The *DNA storage system* consist of three main phases. The first is the *synthesis* process which produces the *oligonucleotides* or *strands* that encode data, second is to store those strands in containers that are out of order, finally is the *sequencing* procedure, which is performed by reading the stored strands and constructing the decoded data. Such strands are usually of lengths in the range of 250 nucleotides, however, from each strand, millions of copies are synthesized to be stored in the containers. Those copies are read with errors (substitution, insertion, deletion), caused by all phases. Therefore, after receiving a cluster (set of reads of the same origin of copies), a *sequence alignment* process is needed in order to achieve a reconstruction of the original strand.

*Multiple sequence alignment (*MSA*)* is a classic problem in bioinformatics where the goal is to align several long DNA sequences under the assumption that all of them represents different species in an

evolutionary chain. From the resulting MSA, phylogenetic analysis is conducted. Since MSA has been developed from the bioinformatics perspective, it is suitable for long genes and protein sequences, furthermore it did not require high efficiency rates.

For the purpose of customizing the MSA algorithm in the DNA data storage field, and since the type of DNA strands used currently differ from genes, moreover since our demands of those algorithms require high efficiency and low time and space complexity, in order to compete with existing data storage systems, adjustments and optimizations to the existing algorithms has to be done.

This work, studies the different existing MSA algorithms, compares them, and suggest an optimizing run complexity algorithm, an algorithm consisting of a phase of iterative alignment with FOGSAA [1], aligning all strands to one prediction of a reference strand, within the second phase is of a dynamically chosen number of iterations, where corrections are made to the predicted reference, based on the results of the alignment and the majority vote, then once again the alignment is done to the original cluster with the corrected reference.

*Semantics:*

Given m sequences $S_i, 1 \leq i \leq m$, as in the form bellow:

$$S = \begin{cases} S_1 = (S_{11}, S_{12}, \dots, S_{1n_1}) \\ S_2 = (S_{21}, S_{22}, \dots, S_{2n_2}) \\ \vdots \\ S_m = (S_{m1}, S_{m2}, \dots, S_{mn_m}) \end{cases}$$

A multiple sequence alignment of this set, is resulted by inserting any amount of gaps needed into each strand $S_i$ until all strands confront to a common length $L \geq \max\{n_i | |S_i| = n\_i\}$, and no column is full of gaps.

We have observed that the different existing algorithms, rely on a naïve pairwise sequence alignment dynamic algorithm.

When customizing the problem to our DNA data storage systems, and since we know the error rates of the different consisting phases of the system, we can construct a more informed pairwise alignment algorithm, starting with the basic two strand alignment.

To refer to the new problem, we will give some denotation:

$$step(S_{ix}, S_{jy}):$$

$$match(S_{ix}, S_{jy}): two\ letters\ match\ and\ were\ matched$$
$$mismatch(S_{ix}, S_{jy}): two\ letters\ that\ don't\ match\ were\ matched$$
$$gap1(S_{ix}, S_{jy}): a\ gap\ was\ inserted\ to\ the\ first\ strand\ at\ the\ index\ i$$
$$gap2(S_{ix}, S_{jy}): a\ gap\ was\ inserted\ to\ the\ second\ strand\ at\ the\ index\ j$$

In other words, $match(S_{ix}, S_{jy})$ means that we assume that no error has appeared at this index, $mismatch(S_{ix}, S_{jy})$ denotes a substitution that happened at index j, and inserting a gap at the first strand indicates that an insertion of $S_{ix}$ occurred in the second strand, and on similar basis, inserting a gap into the second strand suggests that a deletion of the base $S_{jy}$ occurred.

Therefore, we can give preferences to these choices based on the error rates given, we give penalties to each choice of possible step, in order to prioritize the most likely step, denote: $choice_{penalty} = f(rate)$, as the penalty score given to each one of the possible choices, calculated as a monotone decreasing function of the given error rate. And the goal is to minimize this summery of penalties:

$$optimal = \min\{\sum_{i}^{L} choice_{penalty}(S_{ix}, S_{jx})\}$$

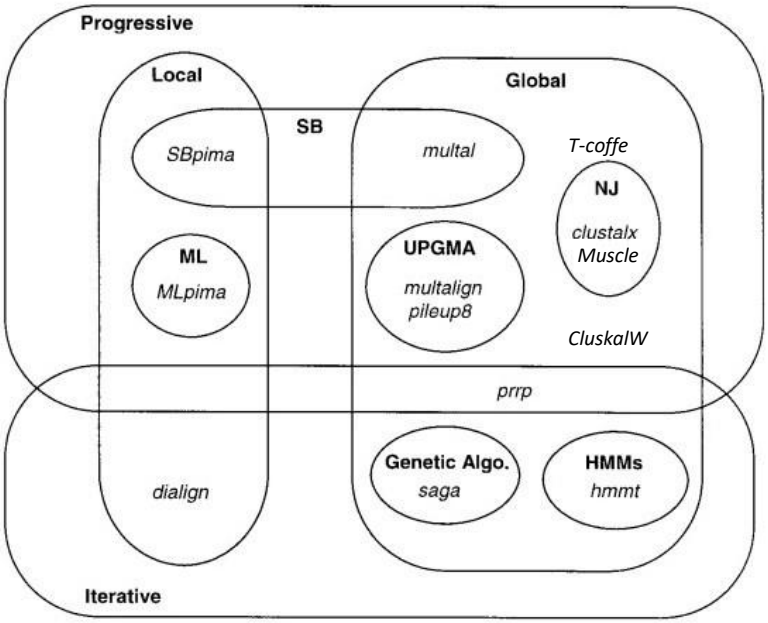
Note that, since the rate of not erroneous indexes is the highest, and since the function is decreasing, we obtain a minimal penalty, by maximizing the matching indices.

## Methods

The naïve approach to solve the problem is a dynamic program that identify the globally optimal solution. For n sequences of length L, the naïve method requires n-dimensional matrix, equivalent to the matrix in the pairwise sequence alignment, and therefore the complexity increases exponentially, $O(L^n)$.

Since, such complexity is not compatible with the large number of sequences and our current computational abilities, many different algorithms were developed, trying to obtain a sufficient alignment with efficient complexity.

To overcome this problem, different heuristic approaches have been developed, resulting in a huge quantity of programs using fundamentally different strategies [2]. The traditional and most common approach has been the progressive alignment method, where the multiple alignment is built up gradually by aligning the closest sequence first and successively adding the more distant ones. Many alignment programs are based on this method, such as MULTALIGN [3], CLUSTALW [4], KALIGN [5], MUSCLE [6], T-COFFE [7] and many others, illustrated in the Figure.



## Materials and Experiments

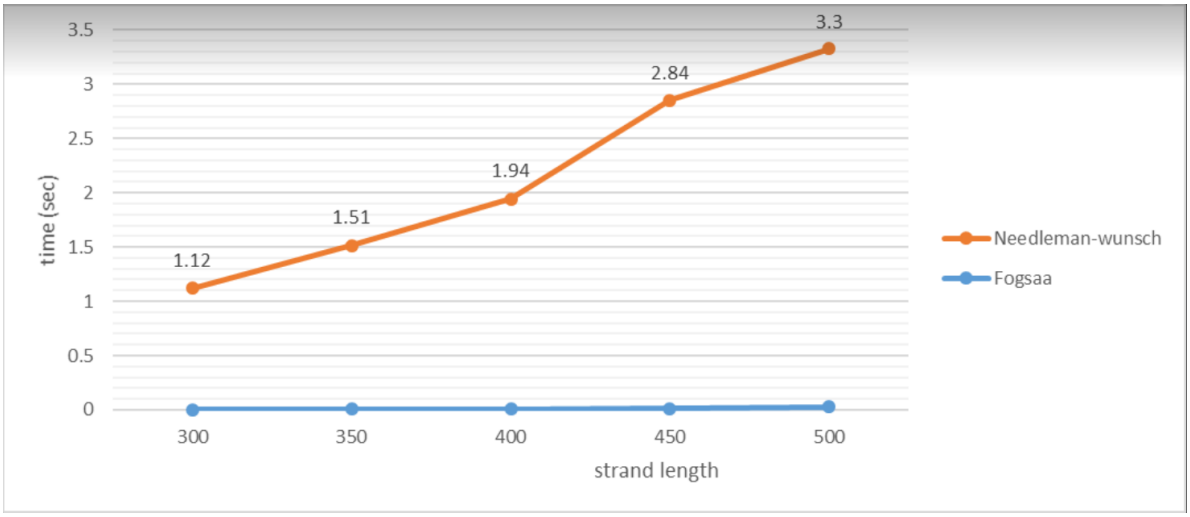
1. *Customizing the pairwise alignment*:

What all of those *MSA* algorithms have in common, is that the cornerstone in the algorithm, is pairwise alignments algorithms, that are done in most of the phases in those algorithms. Furthermore, all these algorithms use the basic dynamic pairwise alignment method (Needleman-Wunsch[8]) which has running complexity of $O(n * L)$, where n is the number of strands and $L$ is each strand's length.

Therefore, to optimize the run time to our problem, we can search for more optimal running complex algorithms, which in this case, might cost us some accuracy loss, but since in our data storage systems, much more layering of decoding and error corrections are applied, we can afford to decrease the accuracy of the alignment.

One interesting pairwise algorithm we noticed named *FOGSAA* (Fast Optimal Global Sequence Alignment Algorithm) [1], Which functions as a tree search algorithm, such that at each step the developed node is chosen base on the optimal score (minimal penalty), the tree search includes branch pruning and is more informed than the dynamic algorithm, thus it yields a great improvement considering time and space complexity.

*FOGSAA* aligns a pair of nucleotide sequences faster than any optimal global alignment method including the widely used *Needleman-Wunsch (NW)* algorithm. *FOGSAA* is applicable for all types of sequences, with any scoring scheme and with or without affine gap penalty. Compared to NW, *FOGSAA* achieves a time gain of (70-90) % for highly similar nucleotide sequences (> 80% similarity) and (54-70) % for sequences having (30-80) % similarity. For other sequences, it terminates with an approximate score.




[Comparison of the running time of *FOGSAA* and *NW* on different strand lengths, strands were simulated with *Ilumina technology*'s error rates, as a result we found that *FOGSAA* is significantly faster than *Needleman Wunsch*.]

## *2.* Evaluating The Improved FOGSAA Algorithm:

To further illustrate the complexity improvements, and compare FOGSA to the naïve dynamic algorithm, we need to define evaluation metrics that can measure the alignment's accuracy.

### *2.1.* Defining evaluation Metrics:

Given a reference strand denoted by S1, and an erroneous copy S2. Denote S3 and S4 the two strands after the alignment, and let N be their length. We introduce the Metrics:

- Distance from Lavenstein (DFL) Metric:

$$DFL(S1, S2, S3, S4) = [N - count_{match}(S3, S4)] - d_{levenstein}(S1, S2)$$

In other words, DFL computes the distance between the minimal edits needed to obtain S1 from S2, and the number of not matched indices which indicates the number of "errors" we assumed appeared (insertion, substitution, deletion).

- LCS Distance (LCSD) Metric:

$$LCSD(S1, S2, S3, S4) = LCS(S1, S2) - count_{match}(S3, S4)$$

The difference between LCS length, which is the length of the longest common subsequent, and the number of matches obtained in the alignment. This metric indicates how many possible matches we have missed.

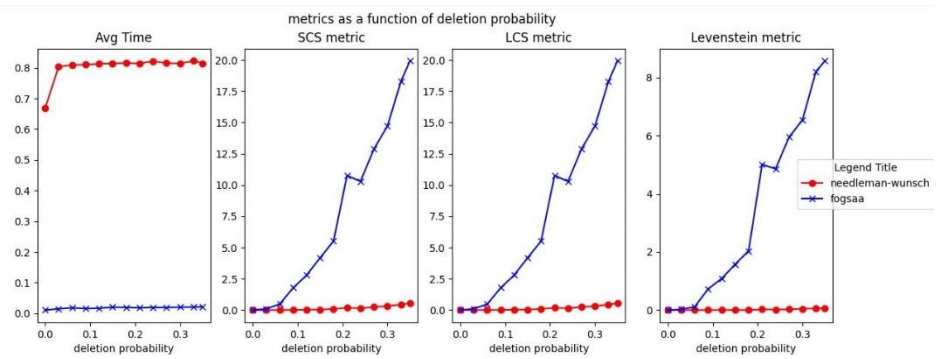
- SCS Distance (SCSD) Metric:

$$SCSD(S1, S2, S3, S4) = N - SCS(S1, S2) + count_{mismatch}(S3, S4)$$

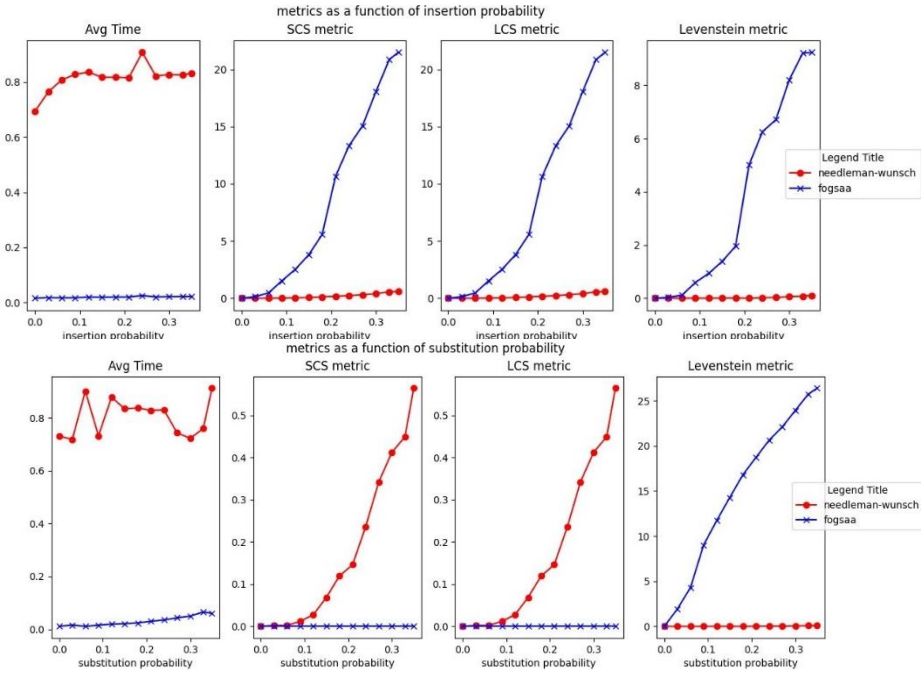
The difference between the length of the aligned sequence, and the shortest common super sequence length, which indicates the unnecessarily added gaps, we also subtracted the number of mismatches, because when a substitution of one base occurs, it adds 2 bases to the SCS, but in the alignment they should be mismatched and not considered as two gaps.

### *2.2.* Comparisons:

- To reveal the correlation between the different error type rates and the Metrics defined, in addition to evaluating the algorithm's accuracies, an experiment was conducted, where 10 files that contain an erroneous cluster of size 1000 with strands of length 250 base, each with different error rates, were simulated by the simulator. We ran the two algorithms, and computed the average metric value for each file, and so each graph describes the average metric value affected by the error rates.

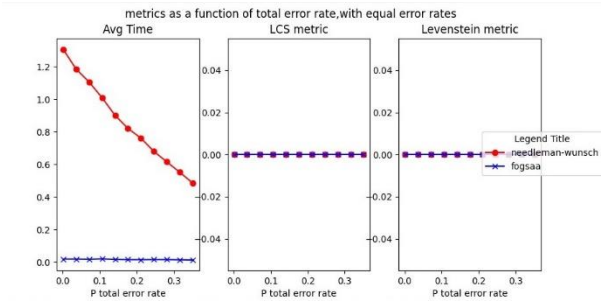


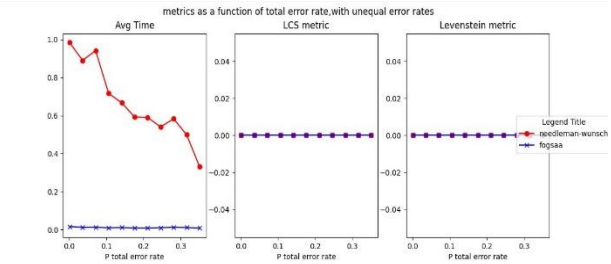
[The graph describes the average time and the metrics for 10 different clusters, comparing between the alignment with FOGSAA and Needleman Wunsch Algorithms, the *x* axes describes the deletion error rate, when the other error rates are 0. We can observe that the average metrics values are increasing with the increase of the deletion rate, and we can observe that the deletion rate affects mainly the SCS distance and LCS distance. However, what can be noticed is that throughout all of the alignments, *FOGSAA* was significantly faster of approximately *x90%*]




[Similar to previous graphs, with the substitution error rate as the x axis, here we notice that the substitution rate affected NW algorithm significantly with the LCS and SCS metric]

- Second, to measure the accuracy of *FOGSAA* alignment, to a more realistic error values that are common to the current sequencing technologies, we conducted a similar experiment, where we changed the total sum of error rates as axis *x*. In the first figure, the different error types rates were divided equally, whereas in the second, they were divided randomly.




## *2.3.* Conclusions:

- After observing the resulting graphs, we can conclude that *FOGSAA* alignment is significantly faster the NW, however, it comes in costs of accuracy, concerning the optimality of the alignment.
- In the second experiences, we can see that within the range of lower error rates that are correspondent to the current technologies, the accuracy of both algorithms is perfect relative to our pre-defined metrics.

- In addition, we have noticed that both the SCSD and LCSD graphs coincide, and decided to drop the SCSD metric.

**Claim 1:** *LCSD is equal to SCSD.*

**Proof:** Let S1, S2 be two strands of lengths n, m respectively, and let S1', S2' be the two strands after the alignment of length N.

Denote $match(S1', S2'), mismatch(S1', S2')$ as the number of matched/mismatched letters in the aligned strands.

Similarly let $gaps(S1'), gaps(S2')$ as the number of gaps inserted into S1 and S2 to create S1', S2' respectively.

First of all, we deduct the relation between LCS and SCS, for two input sequences, an SCS can be formed from a LCS. By simply inserting the non-LCS symbols into the LCS sequence preserving their original order.

For example, the longest common subsequence of X[1..m]=abcbdab and Y[1..n]=bdcaba is Z[1..L]=bcba. we obtain a shortest common supersequence U[1..S]=abdcabdab.

Thus, $|LCS| + |SCS| = m + n$ $_{(1)}$ holds for any two input sequences.

Secondly, the following equations hold:

$$N = match(S1', S2') + mismatch(S1', S2') + gaps(S1') + gaps(S2')_{(2)}$$

$$N = n + gaps(S1'), N = m + gaps(S2')_{(3)}$$

$$\rightarrow m = match(S1', S2') + mismatch(S1', S2') + gaps(S1'),$$
$$n = match(S1', S2') + mismatch(S1', S2') + gaps(S2')$$

Based on the Definitions of LCSD and SCSD:

$$SCSD = N - |SCS| + mismatch(S1'S2') \rightarrow^1$$

$$SCSD = N - m - n + |LCS| + mismatch(S1', S2') \rightarrow^2$$

$$SCSD = m + gaps(S2') - m - n + |LCS| + mismatch(S1', S2') \rightarrow^3$$

$$SCSD = |LCS| + gaps(S2') - match(S1', S2') - mismatch(S1', S2') - gaps(S2') + mismatch(S1', S2')$$

$$SCSD = |LCS| - match(S1', S2') = LCSD \blacksquare$$

*3.* Customizing Multiple Sequence Alignment:

Sequence Alignment in the DNA-Storage domain, differs from the use of Sequence Alignment in the Biological domain, and so to customize the Algorithm such that it answers to our assumptions and demands, we first will list the preliminary assumptions:

For an encoded DNA strand that was stored and read, we usually assure some characteristics:

- The error rates of each type of error, (substitution, insertion, deletion) relevant to the sequencing technology, is known and given.
- The reads of a strand undergo reconstruction algorithms before the MSA phase.
- The strand is GC-balanced.
- The strand does not include long runs of the same base.

Utilizing these assumptions in our MSA algorithm, can optimize the complexity and build a more unformatted algorithm.

Integrating the above assumptions into *FOGSAA* pairwise alignment algorithm:

- Since FOGSAA is a tree search algorithm that prioritizes the branches with maximal score, we built a decreasing monotone function that provides negative penalty values to each error type, respective to the error type rate subordinate to the used sequencer technology.
- We assume that the reference strand is obtained as a results of a given reconstruction algorithm, or some other estimator, this can be done due to the knowledge that the reads were all originated from the same encoded strand, in contrary to the biological use of the alignment methods, and therefore we have the advantage of having a predicted reference strand. We utilized this advantage, by creating an iterative pairwise algorithm, that aligns every read to the reference, and therefore we have n (number of reads) alignments of strands, which is a huge improvement compared to the progressive methods previously used.
- As for the other characteristics provided, we used them in eliminating strands that did not abide, on the bases that they are very erroneous or were mistakenly added to the cluster.

## The Model

*1.* MSA Iterative algorithm based on the Improved FOGSAA:

The algorithm is divided into two phases**:**

Phase 1:

Given the erroneous cluster and an estimated reference, we align the cluster iteratively to the estimated reference strand given.

Reference evaluation and correction phase:

After we obtain the cluster aligned, we can conduct an evaluation of the provided reference based on the majority vote of the alignment.

Given the aligned cluster, we iterate on the reference indexes, and observe the majority's vote on the correctness of this base.

- The majority has a gap in this index, it indicates that this base was an insertion error in the reference, therefore we can delete it.

- The majorities vote is a mismatch with the majority of the base x, then this indicates that a substitution has occurred in the reference, and therefore we can correct the base to x.
- The majority decided to insert a gap in the reference at this index, (we observe that by updating a histogram that counts the number of different strands that indicated that a gap is to be added in the reference at this index), then it indicates that a deletion has occurred in the reference and therefore we can insert the base that appears in the majority's vote.

The evaluation of the reference is calculated as the number of indexes that the majority has voted for an error occurring in the reference, divided by the reference's length.

The reference evaluation function is called, and if the evaluation appears to be under a pre-defined value, then phase two proceeds.

Phase 2:

The reference is corrected by the above explained function, and a second round of iterative pairwise alignment to the new reference is performed.

This phase is conducted *k* times, where k is a pre-defined parameter.

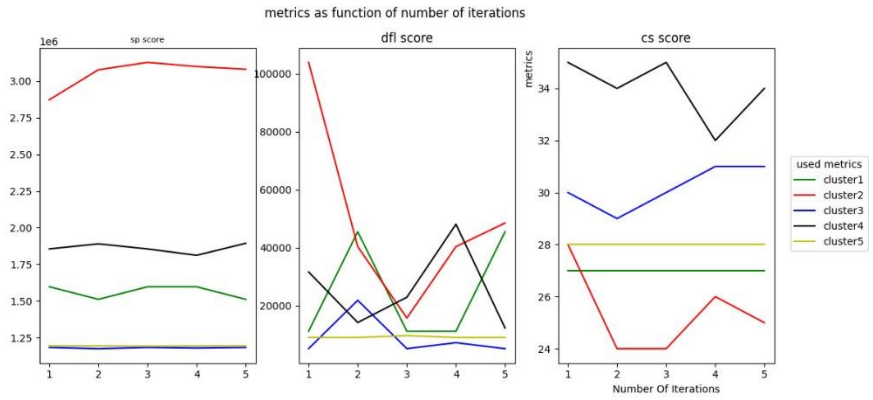
In order to find the most promising *k* , we conducted a tuning process which is described later on.

Note that: In order to fairly compare the algorithm to other MSA technologies, and since they don't utilize the estimated reference, we decided to add a feature, that can conduct the same algorithm, but instead of starting with the given estimated reference, only the original strand length is needed, we randomly choose from the strands in the cluster that have the same length.

The algorithm is linked in the additional information.

2. Tuning process

As described above, our algorithm is constructed of two phases, where phase 2 aims to improve the alignment by improving the reference an repeating the alignment once again, this phase is repeated k times.
In order to decide the best value to assign to k, we conduct an experiment where we ran our model several times on 5 clusters, where each time we assign a different value for k, and then we evaluate the resulted alignment.



[In the first diagram, a higher score indicates a better alignment, and we can see that our model received the highest scores in for k = 3, the same applies to the third diagram, with k = 3 achieving relatively the best results. Whereas, the second diagram, lower values indicate better results, which were received in k = 3]

As a result, we decided to proceed out work with k = 3.

Note: we limited the values of k to {1,2,3,4,5} and didn't experiment bigger values because we want the algorithm to be fast, while using higher values can slow down our model.

*3.* Defining Metrics for evaluating the MSA algorithm:

To assess the performance and to evaluate the MSA quality, there was a need to find or create appropriate metrics that could reflect the alignment efficiency.

Famous metrics that are widely used to estimate the quality are the sum-of-pairs score (SPS) which is calculated such that the score increases with the number of sequences correctly aligned. The column score (CS) is a binary score given to each column in case it was fully identical to all the strands. The last metric we developed using the distance-from-Levenstein metric (DFLS) we showed earlier but calculated for each two strands in the given cluster, we expect that lower scores in this metric to be better because it indicates that the alignment results are not far from the edit distance between the strands.

- sum-of-pairs score (adjusted):
  Suppose we have a cluster of N aligned sequences, each sequence consists M bases.
  For each pair of sequences $s_i, s_j$, and each index $k$, $k \epsilon [M]$, we compare $s_{ik}$ $with$ $s_{jk}$ based on the comparison we assign a value as follows :

$$val(s_{ik}, s_{jk}) = \begin{cases} 1 & if\ s_{ik} = s_{jk} \\ -1 & s_{ik} \neq s_{jk}, and\ none\ of\ them\ is\ a\ gap \\ -1 & only\ one\ of\ them\ is\ a\ gap \\ -2 & if\ both\ are\ gaps \end{cases}$$

  Thus the SPS is:

$$SPS = \sum_{i \neq j}^{N} \sum_{k=1}^{M} value(s_{ik}, s_{jk})$$

- column score (adjusted):
  Given a cluster of N aligned sequences, each sequence consists M bases.
  For each index $k \epsilon M$, we assign value $C_i = 1$ if all sequences agree in that index, otherwise $C_i = 0$.

$$CS = \sum_{k=1}^{M} C_i$$

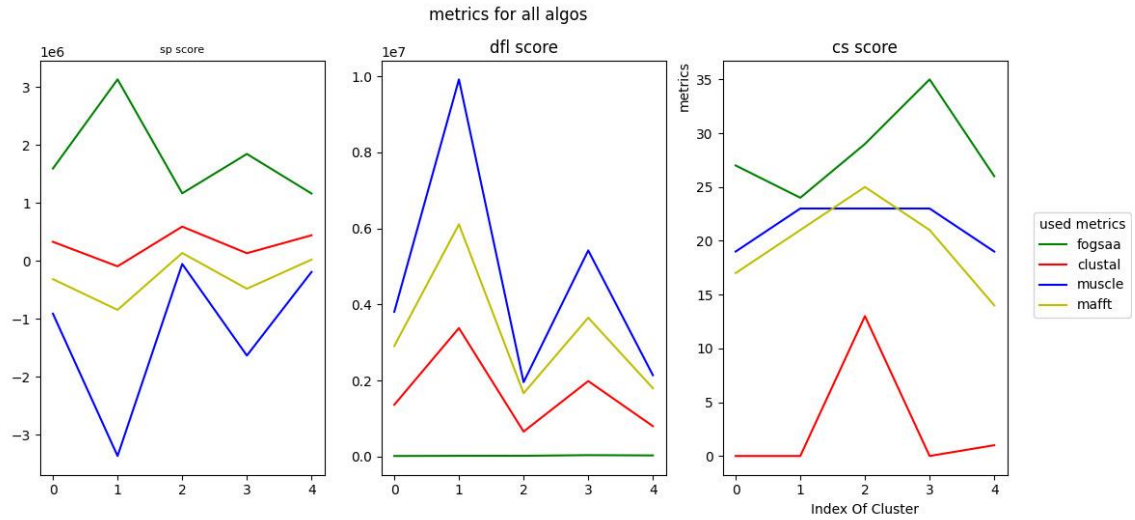
- distance-from-Levenstein Sum:
  Given a cluster of N sequences before and after the alignment, for each pair of sequences $s_i, s_j$we calculate the $DFL(s_i, s_j)$ described in the pairwise alignment comparison section, and thus:

$$DFLS = \sum_{i \neq j}^{M} DFL(s_i, s_j)$$

4. Comparing MSA algorithms

We have been provided by the project mentors a several JSON files that contain clusters, for each JSON file we aligned the strands cluster using three different MSA algorithms, Clustal Omega [4], MUSCLE [6], MAFTT [9] and the model described in this article.

The comparison was made using the metrics we described earlier. We found that our Model scored significantly better in all of the clusters in all the metrics.




[In the first diagram, a higher score indicates a better alignment, and we can see that our model received the highest scores in for all the clusters, the same applies to the third diagram. Whereas, the second diagram, lower values indicate better results.]

5. Conclusion

In this article we present a new Multiple Sequence Alignment algorithm that is based on a pairwise improved algorithm, FOGSAA [1]. We identified the differences between the general MSA problem, and the specific MSA problem in the DNA Storage systems, which allowed us to establish assumptions that can be utilized to customize the algorithm to fit our demands. Based on these assumptions we were able to construct a new Model that is iterative, which replaces the current progressive algorithms, and outperforms them.

## Future Improvement Suggestion

For the future, we are planning to use the Artificial Intelligence Algorithms in order to improve our alignment. Since we are looking at our problem as a search problem, Our current model is a blind search algorithm which is similar to Uniform Cost Search Algorithm, it uses the backward cost $g()$ as a function of error rates. Our ambitions are towards upgrading the algorithm into a combination of the current UCS and an informed greedy algorithm, which uses a forward cost function, aka A*.

A possible heuristic we thought about is the difference between the edit distance and the non-matching steps done up until the current state, or zero if the value is negative. Note, that the suggested heuristic is admissible, (1) assume that an actual alignment path received a value that is less than the heuristic, then the total number of none-matching steps is less than the edit distance, which is by definition the minimum number of operations required to transform one string into the other, in contradiction. (2) By definition of the heuristic it cannot receive a negative value.

Therefore, since it is proven that A* finds the optimal solution path, when given an admissible heuristic, it is promising to explore this domain.

## References


[1] Angana Chakraborty & Sanghamitra Bandyopadhyay, FOGSAA: Fast Optimal Global Sequence Alignment Algorithm, 29 April 2013.

[2] Julie D. Rhompson, Frederic Plewaniak and Oliver Poch, A comprehensive comparison of multiple sequence alignment programs.

[3] Barton,G.F. and Sternberg,G.E. MULTAKALIGN, 1987, J. Mol. Evol.

[4] Paula Hogeweg; Des Higgins; European Molecular BiologyLaboratory, ClustalW: Widespread Multiple sequences alignments program, 2008.

[5] Timo Lassmann and Erik LL Sohnhammer, Kalign – an accurate and fast multiple sequence alignment algorithm, 2005.

[6] Robert C. Edgar, MUSCLE: multiple sewuence alignment with high accuracy and high thoughput, 2004.

[7] Cedric Notredame, Desmond G. Higgins and Jaap Heringa, T-COFFEE: A Novel Method for Fast and Accurate Multiple Sequence Alignment.

[8] Needleman, S.B. and Wunsch, C.D. (1970) A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. Journal of Molecular Biology, 48, 443-453.

[9] Katoh, Kazutaka & Rozewicki, John & Yamada, Kazunori. (2017). MAFFT online service: Multiple sequence alignment, interactive sequence choice and visualization.


## Additional Information

Link to our MSA Algorithm Model:

https://github.com/duna-m/FogsaaBasedMSA