Optimizing Parameters of Multiple Sequence Aligners using Particle Swarm Optimization

Alex Ellis*, Dylan Sun*, Stephen Wu*

Abstract

Multiple Sequence Alignment (MSA) is a fundamental tool in computational biology, employed to align DNA, RNA, or amino acid sequences for evolutionary analysis and protein structure inference. Despite the availability of several MSA aligners, like CLUSTALW and MUSCLE, the selection of optimal parameters for alignment remains a challenge, often leading to suboptimal alignments when default settings are used. In this study, we introduce a novel approach utilizing an example of algorithm in nature – Particle Swarm Optimization (PSO) to optimize the parameter settings of MSA aligners based on biological characteristics of the sequence sets. In this setup, a characteristics library is constructed, which is then used to match input sequences to existing sequences based on similarities in biological characteristics. This process yields the most optimal alignment parameters for ClustalW alignment in any inputted sequences. This approach enables sequence alignment problems to become more dynamic by addressing the NP-hard problem of selecting optimal parameters.

1. Introduction

Multiple Sequence Alignment

Alignment of biological sequences like DNA and amino acid sequences is a crucial computational biology tool for gaining insights into characteristics of the species. It has many implications in biological research including evolutionary analysis and protein structure inferences. Multiple Sequence Alignment (MSA) is a computational problem that is used to align more than two biological sequences (DNA, RNA, or Amino Acid Sequences), and it is considered to be an NP-complete optimization problem [7]. A mathematical definition of the MSA problem is as follows: Given a set of k sequences S = {s1, s2, …, sk}, where each si in S is a string made from an alphabet Σ, a multiple sequence alignment on the set of sequences S is an array A = [s1', s2', …, sk'], where each si' in A is a string made from an alphabet Σ ∪ {"-"} and all si' in A have the same length. Removing the gap symbols "-" from any si' should equal to si. The goal of MSA algorithms is to minimize the the total cost of the alignment A, which is calculated by the following expression:

$$\min_{A} \sum_{j} c(a_j),$$

where aj is the jth column of the array A [4]. MSA has significant applications in biological research as it reflects the biological relationships between different species, which could be further used to infer other important information like phylogenetic relationships. An accurate MSA aligner would reveal the biological relationships more clearly and accurately [1].

Currently, there are many softwares that can perform multiple sequence alignment (like CLUSTALW and MUSCLE) given a set of input sequences. These aligners allow users to customize parameters like gap-opening-penalty to be used in MSA, and if the user does not wish to customize the parameters, the aligner would use its built-in default parameter configuration to perform the MSA. Such a default parameter is determined during the establishment of online softwares after methods of optimization and is common for researchers when dealing with MSA in current years to establish their results based on such default parameter alignments. Take ClustalW as an example, ClustalW (1.82) contains five different parameters for protein sequence alignment and has the default as such:

Gap opening penalty = 15.0
Gap extension penalty = 6.66
Protein Matrix = Gonnet
ENDGAP = -1
GAPDIST = 4

Each default parameter here was optimized using different standards to obtain the most reasonable value considering multiple aspects of the nature of the unaligned sequences. Take gap opening and gap extension penalty as an example. Gap opening penalty stands for the cost of opening a new gap of any length in the alignment, and gap extension penalty is the cost of extending every item in such a gap. The two values are reasoned to be dependent on different protein weight matrices, similarities in sequences, length of sequences and difference in length of sequences [8]. Default parameters for gap opening and gap extension penalties are thereby calculated based on formulas derived utilizing these dependencies [9]. However, the default parameter configuration does not always produce the best MSA of the sequences for every set of input sequences. The dynamic approach of changing parameters based on any given input sequences is currently considered as a NP-hard problem that researchers are solving in recent years. From the study [7], we got the idea that sequences may have different parameter configurations that produce the best resulting alignment depending on their biological characteristics, so a better way to select parameter configurations for a set of input sequences may be using the parameters configurations that produces the optimum result on a set of sequences with similar biological characteristics. This is referred to as the characteristics-based framework [7]. In this study, we aim to first build a characteristics library with various sets of sequences with different biological characteristics with their corresponding optimum parameter configurations for different MSA aligners, then use the characteristics-based framework to produce parameters for sets of test sequences to see if the aligners produce more accurate alignments than using only the default parameter configurations.

Particle Swarm Optimization Algorithm

To construct the characteristics library, we use Particle Swarm Optimization (PSO) to find the optimum parameter configurations of different sets of sequences. PSO is an algorithm mimicking swarm behaviors in nature to solve optimization problems. The algorithm involves moving a number of particles through an n-dimensional search space, where each particle represents a potential solution to the problem. The particles would move in the search space in each iteration and get assessed by an evaluation function. The direction of movement of each particle in the next iteration is determined by three factors: the global optimum (the optimum solution found among all particles so fat), local optimum (the optimum solution found by this particle so far), and momentum (the tendency to keep going to its current direction, which can be modified to suit different purposes). By adding randomness to the movement and carefully balancing the particles' tendencies of exploration (explore unexplored solutions) and exploitation (converge towards the current optimum solution), after several iterations, the particles would start converging towards the optimum solution [2]. The movement of each particle in each iteration can be represented by this equation

$$v_{id}(t+1) = Wv_{id}(t) + c_1 r_{1d}(p_d^{\text{best}}(t) - x_{id}(t)) + c_2 r_{2d}(g_d^{\text{best}}(t) - x_{id}(t)),$$

where vid(t) represents the momentum, pd^best (t) represents the local optimum, gd^best (t) represents the global optimum, and xid(t) represents the current location. Other variables are weights and constants to adjust exploration and exploitation of the particles. An illustration of particle movement is shown by *Figure 1*. The pseudocode demonstrating the complete PSO algorithm is shown by *Figure 2*.

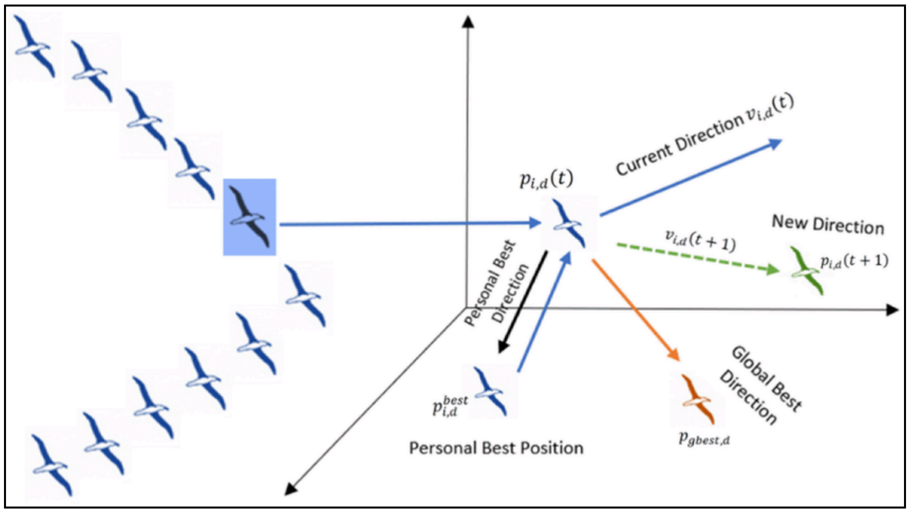
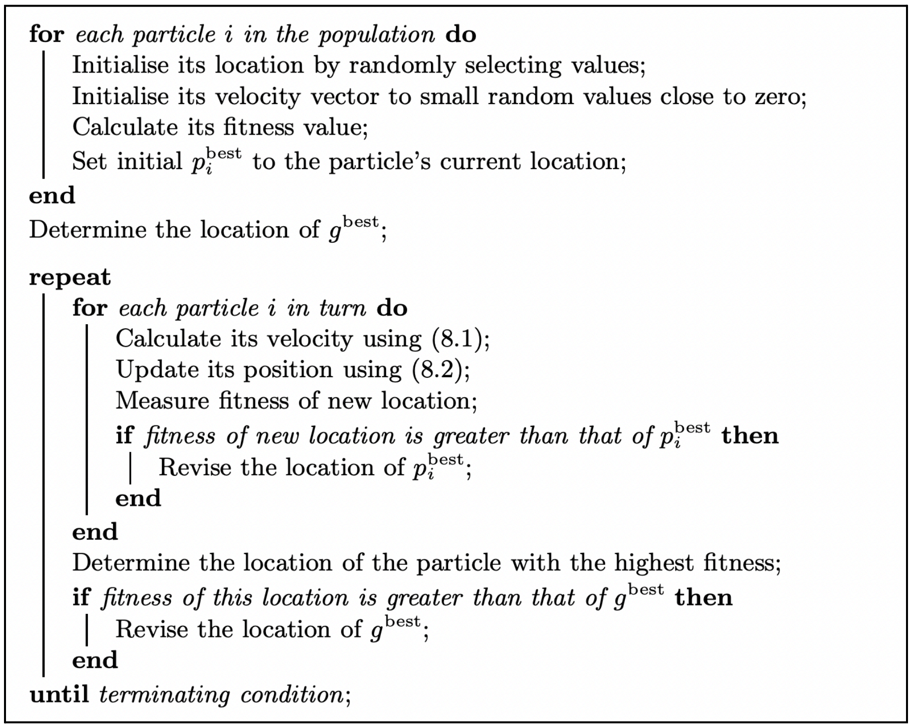


Figure 1: A graphical illustration of how each particle moves in each iteration in PSO. Image source: https://www.researchgate.net/figure/Graphical-representation-of-PSO_fig1_350600249



```
for each particle $i$ in the population do
    Initialise its location by randomly selecting values;
    Initialise its velocity vector to small random values close to zero;
    Calculate its fitness value;
    Set initial $p_i^{\text{best}}$ to the particle's current location;
end
Determine the location of $g^{\text{best}}$;
repeat
    for each particle $i$ in turn do
        Calculate its velocity using (8.1);
        Update its position using (8.2);
        Measure fitness of new location;
        if fitness of new location is greater than that of $p_i^{\text{best}}$ then
            Revise the location of $p_i^{\text{best}}$;
        end
    end
    Determine the location of the particle with the highest fitness;
    if fitness of this location is greater than that of $g^{\text{best}}$ then
        Revise the location of $g^{\text{best}}$;
    end
until terminating condition;
```

Figure 2: Pseudocode for PSO Algorithm (Image from [2]).

Converting MSA Parameters Optimization to a PSO Framework

To optimize MSA parameters using PSO, we need to incorporate the MSA parameter optimization problem into the PSO framework. And there are two major parts we need to define in the PSO framework–the particles, and the evaluation function for the particles.

We represent each particle as a vector of quantitative parameters to be optimized. For example, a particle can be: [Gap-opening penalty, Gap-extension penalty, Maxdiv, transweight]. And we evaluate each particle by running the MSA aligner with the parameters represented by the particle, and assessing the resulting alignment using the objective SP-score, which we will discuss in the next section.

The rest of the paper would be structured as follows: Section 2 would describe how we construct the characteristics-based framework, as well as how it functions; Section 3 would present how we test the framework, present the result, and discuss the result; and finally Section 4 would assess the limitations of our study and some future directions to improve.

2. Constructing the Characteristics-based Framework

General Idea

The characteristics-based framework is divided into two components: the characteristics library, and the search algorithm.

The characteristics library is constructed using sets of sequences that have their parameter configurations optimized by the PSO algorithm. Each set of sequences used to construct the library would have a reference alignment, which is its optimum alignment. Constructing each particle as a vector of parameter configuration, during each iteration of PSO, an alignment of the sequences produced using the parameter configurations given by the particle is compared to the reference alignment via a difference function based on SP-Scoring. The goal of PSO is to have particles converge towards a parameter configuration that minimizes the difference between the resulting alignment and the reference alignment. The resulting parameter configuration, along with the biological characteristics of the sequences, are stored in the library. Biological characteristics for each sequence set are extracted by a characteristic extractor function. For each sequence set in the library, ten characteristics are extracted and stored for use in the parameter lookup mechanism.

In the search algorithm, the same extractor function is utilized to determine the biological characteristics of unaligned sequences. We then identify the set of biological characteristics that are most similar to the extracted one, and use the corresponding optimum parameter configuration to align the input sequences.

The characteristics-based framework, when completed, would take in a set of sequences to be aligned, extract its biological characteristics and use the search algorithm to find the closest biological characteristics in the library, use the optimum parameter configuration corresponding to that biological characteristics to run the MSA aligner with the input sequences, and output the resulting alignment. An illustration of the structure of the framework is shown in *Figure 3*.

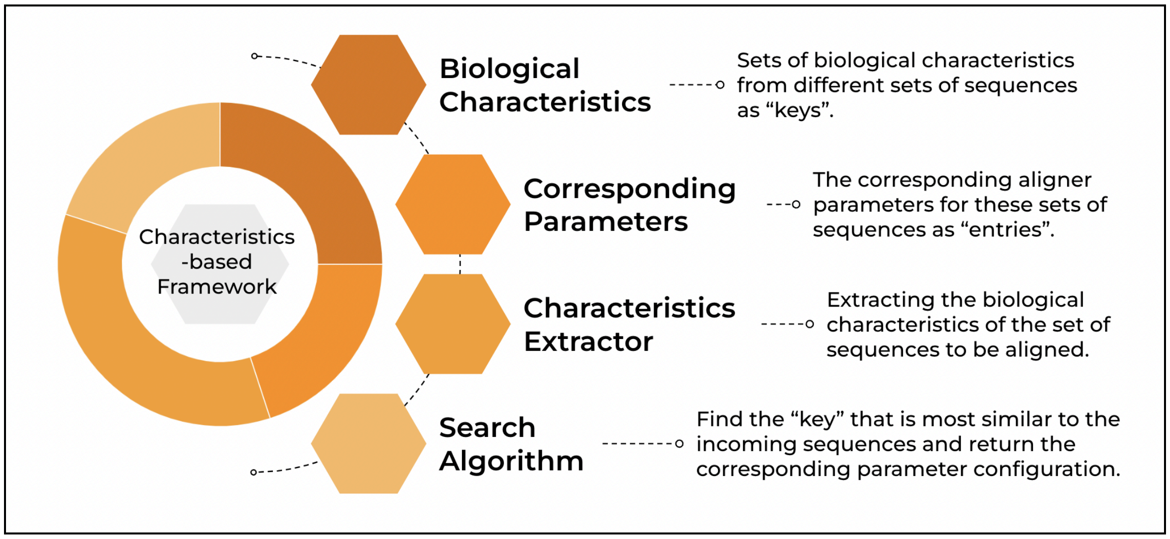

Characteristics -based Framework
Biological Characteristics
Sets of biological characteristics from different sets of sequences as “keys”.
Corresponding Parameters
The corresponding aligner parameters for these sets of sequences as “entries”.
Characteristics Extractor
Extracting the biological characteristics of the set of sequences to be aligned.
Search Algorithm
Find the “key” that is most similar to the incoming sequences and return the corresponding parameter configuration.

Figure 3: An illustration of the general structure of the Characteristics-based Framework.

MSA Aligners Used

CLUSTALW

Parameters optimized for this aligner:

- Gap-opening penalty

- Gap-extension penalty
- Maxdiv (Percent identity for delay)
- Transweight (Transitions weighting)

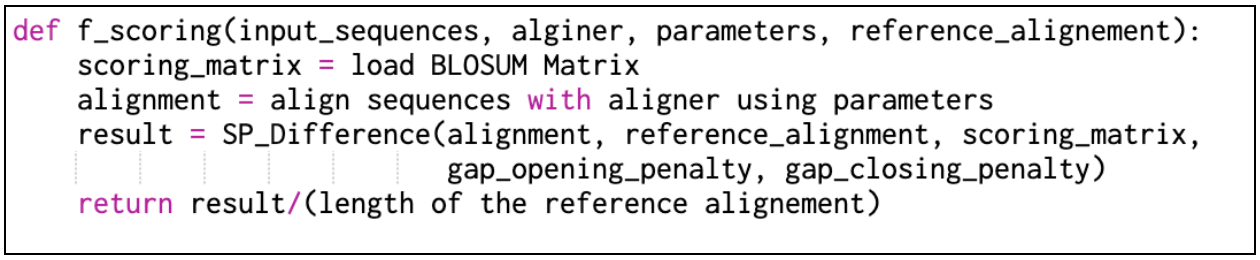
Library Setup

1. PSO Algorithm

   Our implementation of PSO was borrowed from the pyswarms library, which provides a robust implementation of particle swarm optimization. We used the “ps.single.GlobalBestPSO” function, along with 0.5, 0.5, and 0.9 for the c1, c2, and w parameters, and ran the optimizer using the .optimize function on the PSO object. The best position returned by this call was the value used in our optimized parameters library. We set the bounds as 0 to 100 for the gap opening penalty, gap extension penalty, and maxdiv, and 0 to 1 for the transweight parameter.

2. Difference Scoring Function

   The goal of the function is to run the MSA aligner on the input sequences with the parameters given, compare against the reference alignment, and return a score reflecting the difference between the two alignments (the smaller the score, the better the parameter configuration). To compute the difference between two alignments, we used the difference between the Objective SP-scores of the two alignments. *Figure 4* shows a runthrough of the scoring function.



```
def f_scoring(input_sequences, alginer, parameters, reference_alignement):
    scoring_matrix = load BLOSUM Matrix
    alignment = align sequences with aligner using parameters
    result = SP_Difference(alignment, reference_alignment, scoring_matrix,
                           gap_opening_penalty, gap_closing_penalty)
    return result/(length of the reference alignement)
```

Figure 4: Pseudocode of the Difference Scoring Function. SP_Difference is the function that computes the difference of SP-Scores between the alignment produced by the aligner with the input parameters and the reference alignment. The length of reference alignment is the length of a row of alignment (one of the unaligned sequences with gaps inserted)

As an example, the Objective SP-score difference between the alignment ['ADEH-', 'DSR--', 'AS-HL'] and the reference alignment ['ADEH', 'DSR-', 'ASHL'] would be 1.0.

The algorithm we used to compute the Objective SP-score difference between the two alignments is modified from Algorithm 1 and Algorithm 2 from this paper [6].

3. Characteristics Extractor Function

   The characteristics extractor function is used to identify a set of ten biological characteristics of a set of sequences. These ten characteristics are categorized into three groups:

Group A: General categorization of sequences

- A1. Number of unaligned sequences
- A2. Average length of the unaligned sequences
- A3. Standard deviation of sequence lengths

Group B: Comparative analysis of individual sequence among sequences

- B1. Average Kimura Distance [5] between each pair of unaligned sequences
- B2. Standard deviation of the Kimura Distance

Group C: Characteristics of individual amino acids within the sequences

- C1. Percentage of amino acids with positively charged side chains: R, H, K
- C2. Percentage of amino acids with negatively charged side chains: D, E
- C3. Percentage of amino acids with polar uncharged side chains: S, T, N, Q
- C4. Percentage of special cases amino acids: C, U, G, P
- C5. Percentage of amino acids with hydrophobic side chains: A,V, I, L, M, F, Y, W

Characteristics are categorized into three groups to analyze sequences from three distinct perspectives. Group A provides a general overview of the sequences, offering data useful for assessing the performance of aligners in terms of runtime. Group B evaluates the Kimura Distance, which is considered an evolutionary measure between sequences, thereby validating the alignment process. Group C focuses on the individual amino acid composition among sequences, a crucial factor in determining the tertiary structures of proteins based on the properties of individual amino acids.

The evaluation of all three aspects combined offers a comprehensive analysis of a sequence, enhancing the accuracy of matching newly inputted unaligned sequences with those already existing in the library. This multidimensional approach ensures a more nuanced and effective parameter selection for sequence alignment.

Search Algorithm–Finding the closest biological characteristics

In order to determine the set of parameters to use for a given input sequence, we use our data extraction and matching pipeline. To set up our library, we used the above characteristics extractor function in order to extract the characteristics from a large set of data points from the sources we want to optimize for. This list of characteristics, along with the label for the source of each datapoint, is then pickled using the python pickle library so that each time an individual sequence is run, the code to extract all the data doesn't have to run, as this would introduce significant overhead.

When a sequence is passed into the final function, its characteristics are extracted, then its passed off to a function that normalizes the library data using standard scaler fit on that data, and transforms the input sequence with the same scaler. Then the body of data is passed off to the sklearn PCA function, and the normalized input sequence are transformed using the same PCA object. Then the mode of the 5 nearest neighbors in the dimensional reduction of the transformed input data is assigned to the label for that input sequence. This serves as the label for which the optimized PSO hyperparameters are drawn, which are in turn used to run ClustalW. An example of PCA is shown in *Figure 5*.

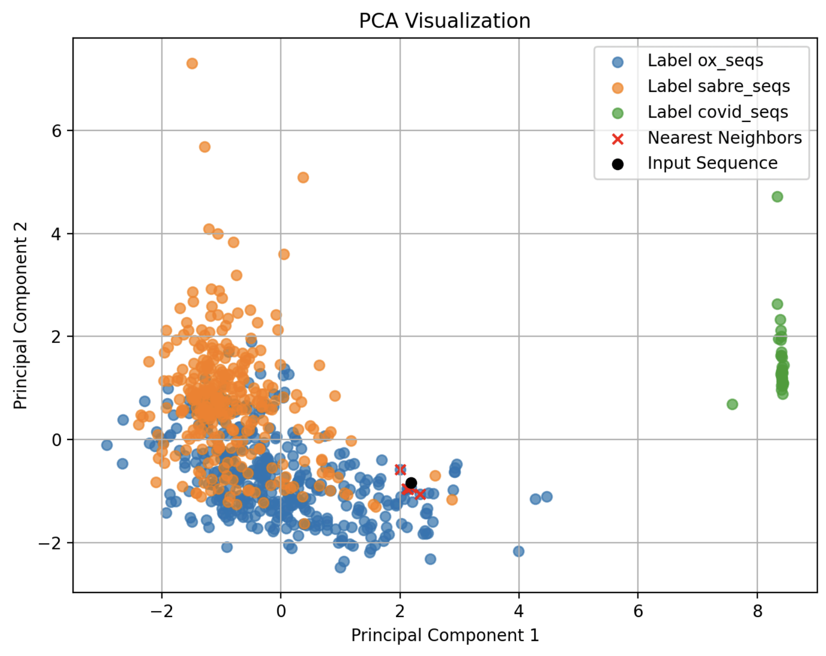


Figure 5: A visualization of PCA, mapping the biological characteristics of ox, sabre, and covid sequence sets into a two-dimensional plane.

3. Testing the Library

Dataset

To test our characteristics-based framework, we used sequences sets of 3 different species–covid, ox, and sabre. Each of these datasets contains sets of sequences along with their corresponding reference alignment. The covid dataset comes from Professor Compeau, containing UK covid sequences during 2020-2022. Due to limited computing power, we selected 35 of them to be used in PCA mapping and testing. Ox and sabre dataset come from Professor DeBlasio. We used all 395 sets of ox sequences sequences and all 423 sets of sabre sequences in PCA, and we used the sabre sequences in testing our framework.

Method of Evaluation

To evaluate the quality of the alignment produced by running CLUSTALW with PSO-optimized parameters and with the default parameters, we used Reference-based Sum-of-Pairs Score (reference-based SP score) to calculate the similarity between the alignment and the reference alignment. The reference based SP score is derived from comparing the computed alignment (the alignment produced by running CLUSTALW with PSO-optimized parameters or default parameters) and the reference alignment column by column, then compute the percentage of pairs (of bases) recovered by the computed alignment. It is computed by scanning through each column of the reference alignment and computing the pairs of bases present in that column (gaps are ignored), then looking at the corresponding column in the computed alignment to see how many of these pairs are recovered in that column. The result is (total number of pairs recovered by the computed alignment)/(total number of pairs present in the reference alignment) [10]. *Figure 6* gives an example on how to compare the commuted alignment with the reference alignment to get the number of pairs recovered and calculate the reference-based SP score.

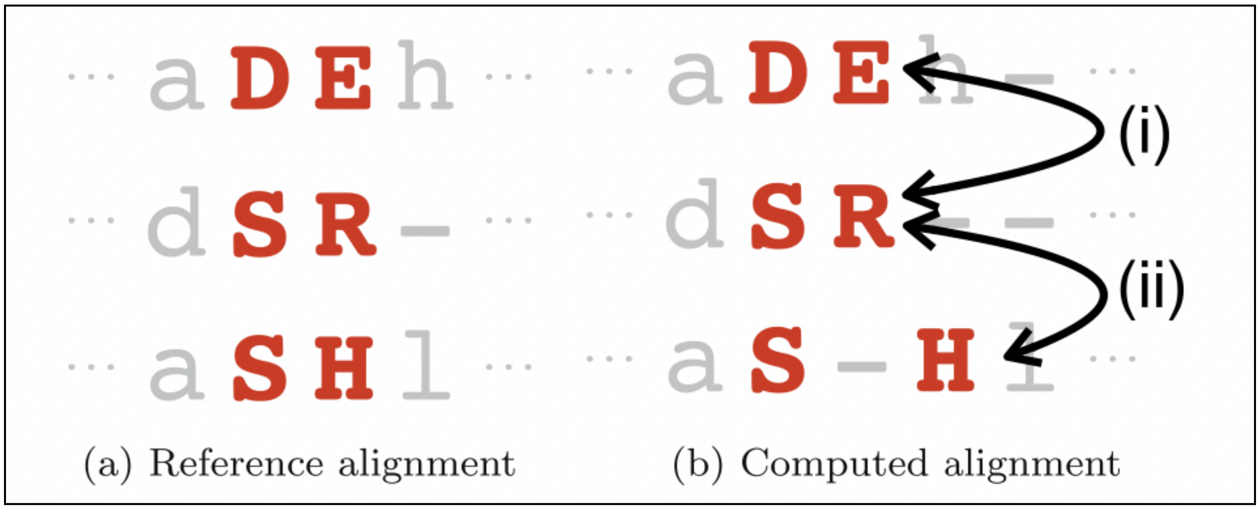
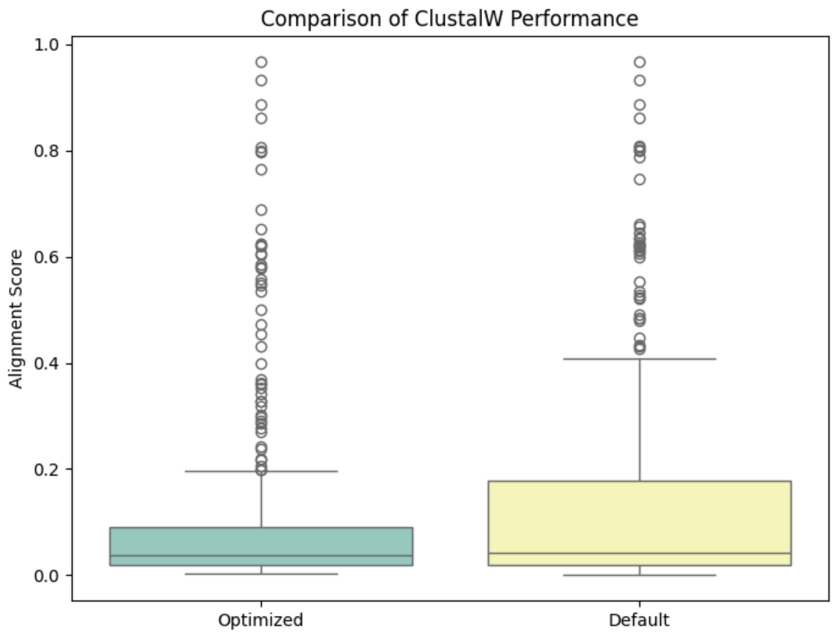


Figure 6: An example of computing the reference-based SP score (Image from [3]). We compare the computed alignment with the reference alignment column by column (only look at the columns in red). Note that the column on the left in the reference alignment has pairs "DS", "DS", and "SS"--all three of which is recovered by the left column in the computed alignment; the column on the right in the reference alignment has pairs "ER", "EH", and "RH", and the right column of the computed alignment only has pair "ER" (note that we are ignoring gaps), so only "ER" is recovered. Thus, the reference SP score for the columns in red is: (total number of pairs recovered by the computed alignment)/(total number of pairs present in the reference alignment) = *(3+1)/(3+3) = 66.67%.*

Result

The result of this optimization algorithm is a set of parameters that in our testing significantly underperformed against the default parameters. In a large battery of tests on the saber dataset, which comes from sequences from sabertooth tigers, we found with a p value of

0.0000009 that the default parameters performed better than the assigned dataset. The effect size for this difference was also not insignificant, as can be seen from the graph below.



Discussion

There are a lot of possible explanations for why this optimization failed. The most significant one is that the parameters were optimized with the objective SP score function, while the final scoring was done with the reference SP score. We aren't sure how different the output space for these two functions are, but a significant deviation could explain the underperformance.

It may also be that optimization on MSA is just hard, and the lack of useful gradients in many places could make the optimization problem too difficult for PSO to tackle with the parameters we chose. It may be the case that with more particles and more iterations that the algorithm would've randomly found better minima than we were able to on our limited hardware. This also reflects another possible issue, which is that the covid dataset could only be run on a low number of particles and iterations due to the aforementioned hardware limitations, making the optimization suspect at best. For the smaller dataset, there may not be enough information for the characteristic extractor function to return a meaningful description of the data, making the PCA dimensionality reduction less effective. The ideal case for this kind of optimization, which would be longer datasets with a more robust optimization, was ultimately not possible in this project, though it's suspect how much this would actually help address the gap between our performance and the default performance.

4. Limitations and Future Directions

MSA Parameter Advising Sample Space

MSA parameter advising could be a difficult problem to optimize on because it has a search space that is not continuous.

For pairwise sequence alignments, a utility function that outputs the quality of an alignment given a reference alignment is proven to be piecewise structured [1]. *Figure 7* shows an example of how the alignment quality can change as a parameter changes.

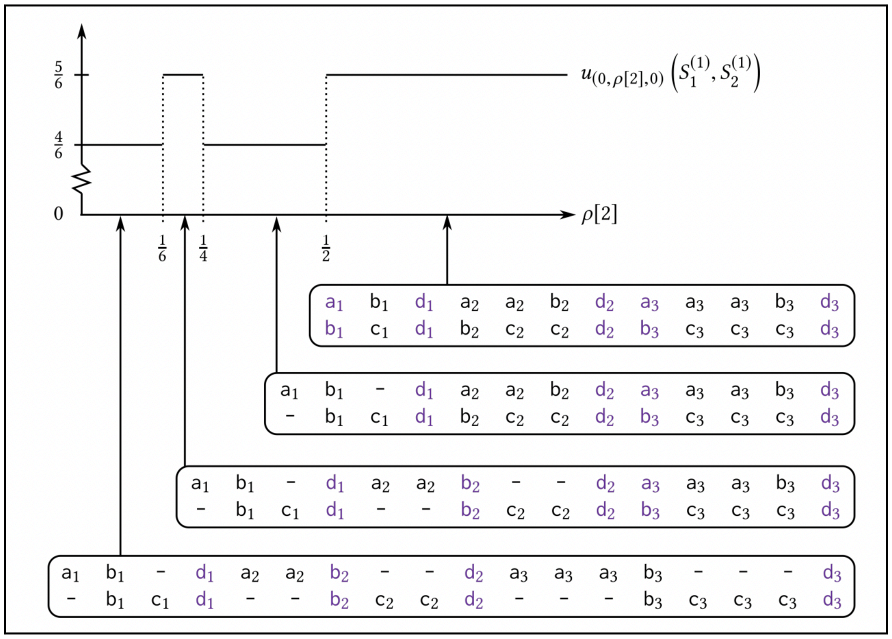


Figure 7: An example of alignment quality change when varying the indel parameter (Image from [1]). The y-axis represents the alignment quality and the x-axis represents the value of the indel parameter. Note that as the indel parameter changes, the alignment quality does not change continuously, but in a piecewise manner.

Because when aligning a pair of sequences, the alignment quality change as one of the parameters varies can be represented by a piecewise function, which can be viewed as a one-dimensional search space, the alignment quality change when varying multiple parameters would also be discontinuous, thus giving an n-dimensional search space that is not continuous as the alignment quality is a piecewise function of each dimension (each parameter). Aligning multiple sequences simply adds more dimensions to the search space, but the influence of changing each parameter on aligning each pair of sequences is still discontinuous. Therefore, the search space for MSA parameter advising is a multi-dimensional search space that is not continuous.

And finding the global optimum in a discontinuous search space is not easy, as we cannot simply rely on methods similar to gradient descent with some randomness. This could be one of the reasons that our PSO algorithm failed to find the optimum parameter–due to the discontinuous nature of the search space, the particles could be converging towards a local maximum or could have trouble converging to a spot as slight changes in location inside the search space could cause dramatic changes in the alignment quality.

Predator-prey PSO Algorithm

One of the problems that could occur when running PSO is that the particles could converge onto a local optimum instead of the actual global optimum of the whole search space. This problem is especially difficult to prevent when working with a discontinuous search space. To alleviate the issue, we could use a Predator-Prey PSO algorithm instead.

The Predator-Prey PSO algorithm added a set of "predator" particles on top of the original PSO algorithm. The idea is that "predators" would move towards the current global optimum and "chase away" all the "preys" that are converging there, forcing them to find a new global optimum. The "prey" particles have the same mechanism for searching in the search space but are repelled from the "predator" particles (we can simply add a repulsion term to the movement expression of the original particles). The "predator" particles would move towards the current global optimum, and its movement can be represented by the following expressions:

$$v_{\text{predator}}(t+1) = \alpha(g^{\text{best}}(t) - x_{\text{predator}}(t))$$
$$x_{\text{predator}}(t+1) = x_{\text{predator}}(t) + v_{\text{predator}}(t+1)$$
,

where v is the velocity and x is the position of the predator [2]. *Figure 8* presents an example of a Predator-Prey PSO algorithm with 2 "predator" particles.

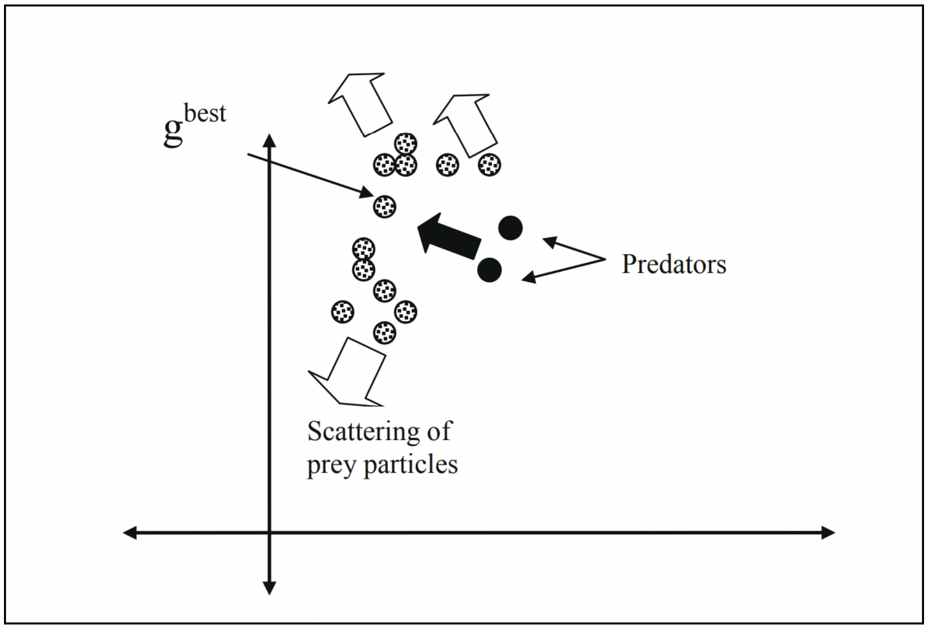


Figure 8: An example of Predator-Prey PSO algorithm that has 2 "predators" chasing the "prey" on the global optimum (Image from [2]).

Other Improvements

The first improvement we could make to our pipeline is to run our PSO optimization using the reference SP rather than the objective SP. Our results point to the possibility of improving the smoothness of the output space is likely significantly outweighed by the deviation in the output space between the objective and reference function. In terms of the existing pipeline, one of the significant advantages of our architecture is that it's very extensible with new sequences with a run of PSO and extracting all the data characteristics for the dimensional reduction. Future improvements could be streamlining this process with dedication functions for adding new data to the dataset. We could also extend this pipeline to include other multiple sequence aligners, which would involve writing dedicated functions to align with those new aligners, along with building parameter libraries for the parameters unique to those aligners.

Lastly, we could consider redesigning large portions of our pipeline, breaking down and testing what works and what didn't and altering the portions that don't contribute to an improvement in the test alignment scores. This could involve using different PSO algorithms, such as the predator prey PSO algorithm described above, changing the closest biological characteristics function from PCA to another dimensionality reduction algorithm, or another mechanism entirely. We could also change the characteristics we are extracting, since our pipeline implicitly assumes that these characteristics map somewhat well to the appropriate parameters for that sequence, but this is an assumption that future research could challenge.

Reference

[1] Balcan, M. F., Deblasio, D., Dick, T., Kingsford, C., Sandholm, T., & Vitercik, E. (2021). How much data is sufficient to learn high-performing algorithms? generalization guarantees for data-driven algorithm design. *Proceedings of the Annual ACM Symposium on Theory of Computing*, 919–932. https://doi.org/10.1145/3406325.3451036

[2] Brabazon, Anthony., O\'Neill, M., & McGarraghy, Seán. (2015). Natural Computing Algorithms (1st ed. 2015.). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-43631-8

[3] DeBlasio, Dan., & Kececioglu, John. (2017). *Parameter Advising for Multiple Sequence Alignment* (1st ed. 2017.). Springer International Publishing. https://doi.org/10.1007/978-3-319-64918-4

[4] Hunt, F. Y., Kearsley, A. J., & Wan, H. (2003). An optimization approach to multiple sequence alignment. Applied Mathematics Letters, 16(5), 785–790. https://doi.org/10.1016/S0893-9659(03)00083-1

[5] Nishimaki, T., & Sato, K. (2019). An Extension of the Kimura Two-Parameter Model to the Natural Evolutionary Process. Journal of molecular evolution, 87(1), 60–67. https://doi.org/10.1007/s00239-018-9885-1

[6] Ranwez, V. (2016). Two simple and efficient algorithms to compute the SP-score objective function of a multiple sequence alignment. *PloS One*, *11*(8), e0160043–e0160043. https://doi.org/10.1371/journal.pone.0160043

[7] Rubio-Largo, Á., Vanneschi, L., Castelli, M., & Vega-Rodríguez, M. A. (2018). Swarm intelligence for optimizing the parameters of multiple sequence aligners. Swarm and Evolutionary Computation, 42, 16–28. https://doi.org/10.1016/j.swevo.2018.04.003

[8] Smith, T. F., Waterman, M. S., & Fitch, W. M. (1981). Comparative biosequence metrics. Journal of molecular evolution, 18(1), 38–46. https://doi.org/10.1007/BF01733210

[9] Thompson, J. D., Higgins, D. G., & Gibson, T. J. (1994). 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(22), 4673–4680. https://doi.org/10.1093/nar/22.22.4673

[10] Zhou, M., Yang, S., Li, X., Lv, S., Chen, S., Marsic, I., Farneth, R. A., & Burd, R. S. (2017). Evaluation of trace alignment quality and its application in medical process mining. *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, 258–267. https://doi.org/10.1109/ICHI.2017.57