EvoAlign: Multiple Sequence Alignment through Evolutionary Computing

Sreevatsa Nukala, Rachel Utama, Sanjana Bhagavatula, John Drohan
Northeastern University
nukala.sre@northeastern.edu, utama.r@northeastern.edu, bhagavatula.sa@northeastern.edu, drohan.j@northeastern.edu

CONTENTS

Multiple sequence alignment (MSA) refers to the process of aligning three or more biological sequences (protein or nucleic acid). We can utilize MSA to determine the phylogenetic relationships between species, detect regions of variability and conservation in a family of proteins, and provide evidence for comparing structure to function of a gene or protein. We tackle the MSA problem with an evolutionary computing approach. We present a set of alignment solutions and visualize tradeoffs between various scoring systems to deliver the best possible alignment, through a user-friendly python package. We believe our evolutionary framework is the ideal solution to achieve a higher solution quality than the heuristic-based approaches, such as EMBOSS and Clustal Omega, while running at reasonable benchmark speeds. The project's goals and desires call for further exploration into optimization to truly outperform established alignment softwares. ***Index Terms*— Multiple Sequence Alignment, evolutionary computing**

I. INTRODUCTION

Our project stands at the intersection of bioinformatics, machine learning, and data visualization. We seek to build a program that generates solutions for the multiple sequence alignment (MSA) [1] optimization problem and presents the tradeoffs to the users as an intelligent design support system. Multiple sequence alignment can help systematically link organisms evolutionarily, illustrate mutation events, and assess sequence conservation of proteins.

Due to the difficulty and intractability of manually processing the sequences given their biologically-relevant length, computational algorithms are used to produce and analyze the alignments from MSA. The MSA problem involves taking in a set of sequences S, and inserting any amount of gaps into each of the S_i sequences. While there exist many types of algorithms that can carry out an MSA, only one can achieve a globally optimal alignment solution, a dynamic programming algorithm. Dynamic programming refers to simplifying a complicated problem by breaking it down into simpler subproblems in a recursive manner.

Smith–Waterman is a local sequence alignment algorithm, using dynamic programming to guarantee an optimal local alignment [2]. Instead of looking at the entire sequence, the Smith–Waterman algorithm compares segments of all possible lengths and optimizes the similarity measure. We start the process by determining the substitution matrix and the gap penalty scheme. This could be as simple as +1 for matches, -1 for mismatches, and a constant -1 penalty for gaps. However, a more sophisticated method would utilize other substitution matrices more aligned toward chemical properties and function of amino acids. We start with the more simple and straightforward substitution matrix. Smith-Waterman initializes a scoring matrix with dimensions of $(length(S_1)+1)\times(length(S_2)+1)$. All the elements of the first row and the first column are set to 0. This extra row and column make it possible to align one sequence to another at any position, and setting them to 0 makes the terminal gap free from penalty. Scoring is done for each element from left to right, top to bottom in the matrix, considering the outcomes of substitutions (diagonal scores) or adding gaps (horizontal and vertical scores). Smith–Waterman then initiates a *traceback* procedure that starts at the highest scoring matrix cell and proceeds until a cell with score zero is encountered, yielding the highest scoring local alignment. Because of its quadratic complexity in time and space, it often cannot be practically applied to large-scale problems. However, we seek to remedy the issue with Smith–Waterman by utilizing it in a smaller role as a simple agent to make

smaller changes to sequences, and generate an optimal solution with more overall efficiency when run over hundreds of generations.

Most of the current solutions to the MSA problem that rely on heuristics, as testing for the optimal MSA under the sum-of-pairs (SP) measure [3] would take a computational program of $(n+1)^k(2^k-1)(k^2)$ time, which is infeasible even for small k values. We know that current market solutions such as ClustalW already exist, and those programs rely on the star alignment heuristic, which aligns all the other clusters to best match one chosen "central" cluster, and then compares pairwise alignments to finally add the individual segments based on the order of similarity. We want to apply an evolutionary computing framework to MSA, using dynamic programming modification agents, and analyze how well it can function as an intelligent design support system that can produce results comparable to, or even better than existing heuristic-based programs.

II. Materials and Methods

When working with biological data, the fasta file format is the most prevalent method for storing sequences. These files typically contain a description followed by a string of characters to represent the individual elements of each sequence. Fasta files can be found using the NCBI website, or the National Center for Biotechnology Information, which stores an expansive network of collected sequence data [4].

Since this project focused on aligning protein sequences, only amino acid fasta files were analyzed. When performing the alignments it is important to select a protein that is well conversed across different species so that the general sequence of amino acids is the same with only minor substitutions and deletions. We selected P53, a tumor suppressor gene, to test our library since many animal species have relatively similar P53 amino acid sequences. To create a testing dataset we searched NCBI for fasta sequences of the desired protein and downloaded three of them to concatenate into a single fasta file for testing.

Once the fasta files were collected and stored the data preparation could be performed. An alignment object was first created and then used to read the fasta files using the read_fasta() method. Since NCBI fasta files are typically downloaded individually for each species, we enabled the read_fasta() method to accept either a single fasta file with multiple sequences, or a list of fasta files each containing any number of sequences. The method uses the BioPython [5] library to parse the fasta files for sequence data and stores them as strings in a list. Since the sequences will likely be different lengths, the read_fasta() method also adds trailing sequence elements in the form of a "*" so that they can all be stored within numpy character arrays increasing iteration efficiency.

Once the alignment object is instituted and the sequences are stored, an Evo object is created to perform the alignment. The Evo library first needs fitness criteria to assess the overall success of an individual alignment. Sum pair scores is an alignment metric that is commonly applied to grade sequence alignments based on matches and gaps so it was used as a fitness criteria in the evolutionary framework. However, since replacing one amino acid with another is not necessarily equivalent due to the individual chemical properties of amino acids it is critical to weight the substitutions accordingly. Blosum matrices [6] can be applied to the sum pair scores metric to automatically apply the appropriate weights. For this project, we decided that selecting one BLOSUM matrix was sufficient, as they all tended to converge to one solution, as opposed to providing tradeoffs.

For our second and third fitness criteria, we web scraped IMGT Education's amino acid properties table [7] for the volume and hydropathy values of each amino acid. In order to convert these amino acid properties into fitness criteria, we performed pairwise combinations of all the amino acids and calculated the differences in their volumes and in their hydropathy properties. A greater difference implies a worse match, and thus, the absolute value of these scores was assigned as negative values to match the Evo framework, where the higher BLOSUM matrix score was indicative of a better match.

Now that the strength of the population could be readily calculated the evolutionary framework required an agent to perform small changes to create new solutions for the population. The Smith-Waterman algorithm [2] performs local pairwise alignments by adding gaps to different sequences. The gaps are strategically placed to reduce the number of mismatches between the two sequences. The Smith-Waterman alignment agent created randomly selects two different sequences from a population and performs an alignment between them. The solution is then evaluated using the fitness criteria and saved to the population.

Once the evolutionary framework was complete with agents and fitness criteria the unaligned sequences stored in the alignment object were added as the first solution in the population. The evolutionary computing framework can then be instructed to run for any number of generations where it will apply agents to the population and evaluate new solutions using the fitness criteria. After a set number of generations, solutions that are completely dominated by another, meaning they are lower by every possible metric, are removed from the population. When the generations conclude the resulting alignment solutions can be visualized accordingly and saved.

The library also automatically pickles the generated solutions so that the next time it is run, it can begin with the solutions it previously created. These solutions are stored in a file that can be deleted to start a completely new alignment.

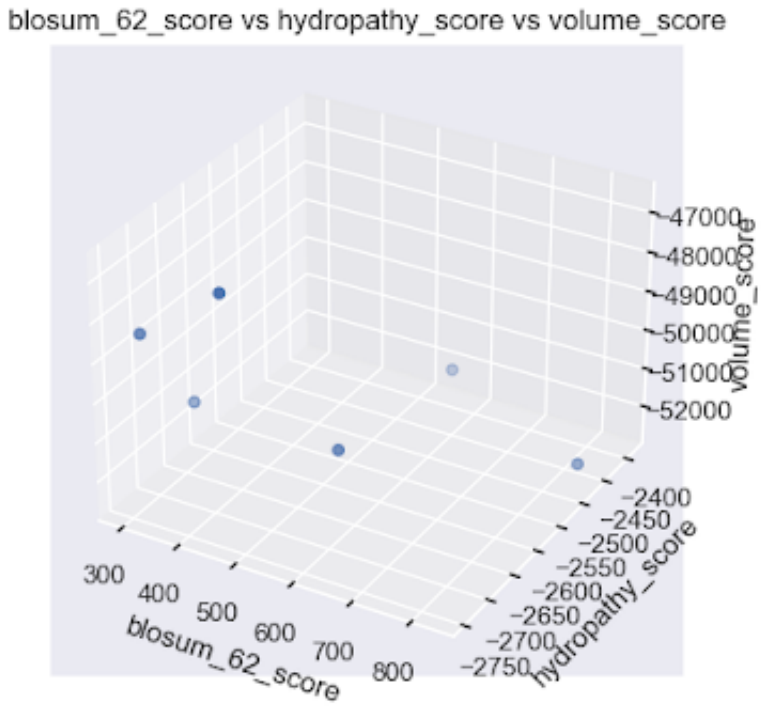
III. Analysis

Our program generates three main outputs: the alignment saved as alignment.fasta, a .png image of the 3D Pareto optimization plot, and a .png image of the pair plot of tradeoffs. The fasta file created is saved in the original format, but now has gaps inserted for the alignment. Before the alignment is saved as a fasta file, the user has the option to

Fig. 1: Fasta file of the best solution

```
>0 <unknown description>
MQEPQSELNIDPPLSQETFSELWNLL*PENNVLSSELCPAVDELLLPESVVNWLDE*DSD
DAPR*MPATSAPTAPGPAPSWPLSSSVPSPKTYPGTYGFRLGFLHSGTAKSVTWT*SP**
*P***PPN*T*****************VRAMAIYKKSEFVTEVVRRCPHHERC***SDGLA
PPQHLIRVEGNLRAKYLDD*NYM************************CNSSCMGGMNRR
PILTIITLEDSSGNVLGRNSFEVRVCACPGRDRRTEEENFHKKGEPCPEPPPGSTKRALP
PSTSSS**********PQ*KKKPLDGEYFTLQIRGRERYEMFRNLNEALELKD*****EP
GGSRAHSSHLKAKKGQSTSR*HKKLMFKREGPDSD************************
***
>1 <unknown description>
MTAMEESQSDISLELPL*FSGLWKLLPPE*DILPSPHC**MDDLLLP*QDVEEFFEGPS*
EALRVSGA*PAAQDPVTETPGPVAPAPATQKTYQGNYGFHLGFLQSGTAKSVMCTYSP**
*****PLNK****QLW**********VRAMAIYKKSQHMTEVVRRCPHHER****GDGLA
PP****RVEGNLY*******************************AYMCNSSCMGGM***
*******LEDSSGNLLGRDSFEVRVCACPGRDRRTEEEN*******CPELPPGSAKRALP
TCTSAS*********PPQ*KKKPLDGEYFTLKIRGRKRFEMFRELNEALELK******ES
GDSRAHSS*L*****Q**PRAFQAL*IKEESP*NC************************
***
>2 <unknown description>
MEEPQSDPSVEPPLSQETFSDLWKLLPENNVLSPLPSQAMDDLMLSPDDIEQWFTEDPGP
DEAPRMPEAAPRVAPAPAAPTPLSSSVPSQKTYQGSYGFRLGFLHSGTAKSVTCTYSP**
*****PPPGT***R************VRAMAIYKQSQHMTEVV*R*****R****SDGLA
PPQHLIRVEGNLRVEYLDDRN************************CNSSCMGGMNRR
PILTIITLEDSSGNLLGRNSFEVH****PGRDRRTEEENLRKKGEPHHE*********L*
PST**K**********PQPKKKPLDGEYFTLQIRGRERFEMFRELNEALELKD*****EP
GGSRAHSSHLKSKKGQSTSR*HKKLMFKTEGP**************DSD**********
***
```

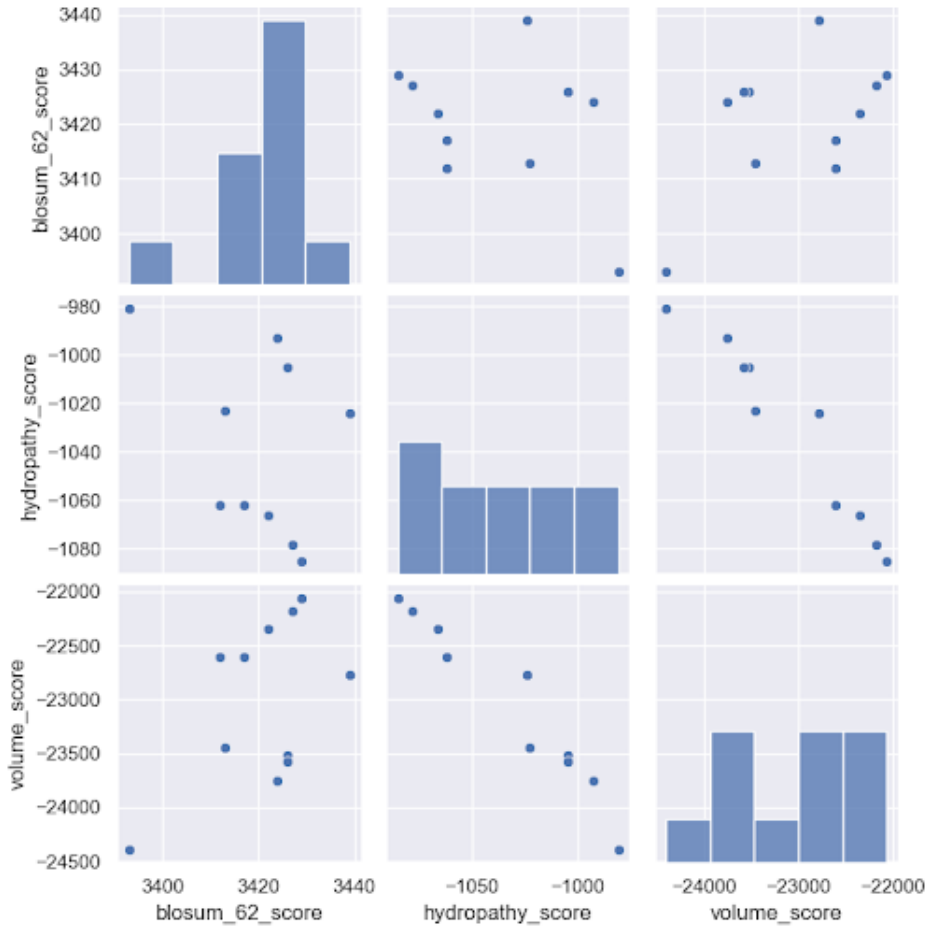
Fig. 2: 3D Pareto optimization plot



rank their fitness criteria based on what is most important to their application. If no ranking is specified, a default order (BLOSUM62, hydropathy, volume) is used and the alignment, which has the highest BLOSUM62 score, is saved in the alignment.fasta file. If there is a tie in BLOSUM62 scores, then we look to the hydropathy score (second rank), etc. Figure 1 shows the returned fasta file with the final alignment. A current limitation is the missing IDs which would be useful to include in future iterations of the project.

The first visualization returned is the 3D Pareto plot which graphs the non-dominated solutions against the three fitness criteria, demonstrating the alignment tradeoffs. Figure 2 dis-

Fig. 3: Pair plot for tradeoffs between fitness criteria



plays a final population of six alignment solutions generated by EvoAlign. For example, solutions presented in Figure 2 show alignments that have high BLOSUM62 and hydropathy scores, but low volume scores. Likewise, there are alignments with higher volume scores, but lower hydropathy and BLOSUM62 scores. The Pareto optimization plot does not include the "perfect" solution for all three fitness criteria. Instead, based on the user's application of our library they must select the solution that optimizes the fitness criteria they deem to be most important to them.

Our last visualization is a 2D pair plot of scatter plots that graphs scores of non-dominated solutions according to different pairs of fitness criteria. This allows the user to examine the relationship between the different fitness criteria in the generated solution population. In Figure 2, the volume score and hydropathy score tended to share a negative linear correlation, and the BLOSUM62 and volume score tended to share a more positive correlation. This visualization demonstrates how the fitness criteria are not independent of each other, which is to be expected, as the BLOSUM62 matrix takes into account many amino acid properties which also include volume and hydropathy. However, since the BLOSUM62 matrix considers several other properties of amino acids as well, the importance of volume and hydropathy may be buried under the contributions of the other properties.

IV. Conclusion

The project successfully generates a multiple sequence alignment for amino acid sequences, builds upon the evolutionary framework covered in DS3500, saves the results as a .fasta file, and produces visualizations for all the tradeoffs based on fitness criteria. The overall library EvoAlign contains Evo, EvoAlign, Align, and Amino objects, with EvoAlign being what the user ultimately interacts with to generate the alignment. Align stores the sequences being aligned. Evo performs the evolution. Amino contains the volume and hydropathy scoring matrices, used as fitness criteria in the evolutionary framework.

The program currently experiences a few limitations, and these are the areas upon which we would like to improve. The first limitation of the program is that the species name is lost after completing the alignment. It would be useful to have the species name tied to the final alignment so that comparisons between the original and final amino acid sequences can happen. Furthermore, the program currently

keeps adding trailing sequences to maintain the length of the sequences, which leads to the buildup of gaps at the end of each sequence. Ideally, the program could identify redundant trailing sequences and remove them before saving the solution to a fasta file. Additionally, we would like to add assets and py tests to make the code more robust and test that the program is running correctly. The next improvement would be to visualize the alignment in a more visually engaging way, such as a colorful Plotly visualization or a dashboard. The speed of a program can always be optimized further; using profilers to determine the bottleneck would help to increase the overall efficiency of the program. It currently takes about 1-2 minutes to align 3 sequences, depending on the number of generations the user requests. For a greater number of sequences, the program is quite slow compared to competitors such as EMBOSS and Clustal Omega. Additionally, we would like to add capabilities for DNA input, but currently, our program's fitness criteria are specifically geared towards amino acids. Overall though, the program is at a stage where generating a multiple sequence alignment using evolutionary computing is completely functional and ready to be released for public use.

V. Author Contributions

While all our group members worked on different parts of the project, we feel that there was equal contribution overall. At our group meetings, we would discuss how to split different parts of the project and then we worked in doubles, with Sanjana and Rachel working together and John and Sree working as the other pair. This way, we were able to work around our busy schedules and still work collaboratively.

Sanjana and Rachel worked on the initial development of the Smith-Waterman modification agent, making the sum pairs score fitness criteria, setting up the sequence alignment in the Evo framework, making the presentation for the class poster presentation, and working on the analysis section of the final report.

John and Sree worked on collecting and reading the data files, implementing the Smith-Waterman modification agent, creating the visualizations, implementing the BLOSUM matrix as part of the sum pair scores fitness criteria, code optimization and cleanup, and writing the abstract, introduction, and methods part of the final report.

References

[1] R. C. Edgar and S. Batzoglou, "Multiple sequence alignment," *Current Opinion in Structural Biology*, vol. 16, no. 3, pp. 368–373, 2006, nucleic acids/Sequences and topology. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959440X06000704

[2] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of Molecular Biology*, vol. 162, no. 3, pp. 705–708, 1982. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022283682903989

[3] H. Carrillo and D. Lipman, "The multiple sequence alignment problem in biology," *SIAM Journal on Applied Mathematics*, vol. 48, no. 5, pp. 1073–1082, 1988. [Online]. Available: https://doi.org/10.1137/0148063

[4] "P53 [cricetulus griseus]." [Online]. Available: https://www.ncbi.nlm.nih.gov/protein/?term=P53

[5] P. J. A. Cock, T. Antao, J. T. Chang, B. A. Chapman, C. J. Cox, A. Dalke, I. Friedberg, T. Hamelryck, F. Kauff, B. Wilczynski, and M. J. L. de Hoon, "Biopython: freely available python tools for computational molecular biology and bioinformatics," *Bioinformatics (Oxford, England)*, vol. 25, no. 11, pp. 1422–1423, Jun 2009, 19304878[pmid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/19304878

[6] S. Henikoff and J. G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proc Natl Acad Sci U S A*, vol. 89, no. 22, pp. 10 915–10 919, Nov. 1992.

[7] "IMGT Education." [Online]. Available: https://www.imgt.org/IMGTeducation/Aide-memoire/$_UK/aminoacids/IMGTclasses.html$