Designing a threading program using double dynamic programming

Eliott TEMPEZ - M2 Bioinformatique
UE Programmation et gestion de projets

Université Paris Cité

1 Introduction

Understanding and elucidating the structure of proteins has been a major scientific issue since the first half of the 20th century [1]; beyond their nucleic sequence, the spatial structure of proteins is often linked to their biological function [2]. As proteins are present in all living organisms, understanding their 3D conformation is crucial in many fields, such as medical research. Although experimental approaches have made it possible to identify many protein structures, they do have technical and material limitations. Algorithmic approaches have made considerable progress, particularly over the last five years [3] ; however, these computational approaches are not new and have been contributing to the progress of knowledge on protein structure since the 1980s [4].

Threading is a protein modelling technique developed in the 1990s [5]. Its aim is to identify the tertiary structure (i.e. the folding) of proteins with no homologs, based on existing proteins for which the structure has already been elucidated experimentally. The method involves 'threading' the sequence of the protein of interest onto the 3D structure of the model protein. An evaluation function, based on the physico-chemical properties of the residues making up the protein of interest and their theoritical positions, is then used to identify the most appropriate structure for this protein sequence [6].

Dynamic programming is an algorithmic method developed in the 1950s which involves dividing a problem into several sub-problems, and solving them by storing the intermediate result so as to retain only the most optimal result [7]. It is widely used in biology, particularly for aligning nucleic or protein sequences [8]. The aim of this project is to re-implement and evaluate the threading approach based on double dynamic programming developed by Jones et al. between 1992 and 1998 [5, 9, 10].

To implement this algorithm, we consider two different proteins: the protein of interest -for which the sequence is known but the conformation is unknown- and a reference protein for which the conformation is known and the sequence is irrelevant. The primary aim of this project is to calculate the overall energy of the theoretical structure resulting from the threading of the protein of interest onto the model, so that we can compare several threadings and deduce the optimal conformation for the protein of interest.

2 Materials and methods

2.1 Program implementation

The code was created using python 3.12.5. Version control was performed with github, for which the repository is available at https://github.com/eliott-tempez/M2_projet_court_threading. All matrix management was done using the python library Numpy.

Information on the protein of interest was retrieved in the form of a fasta file, and information on the model protein in the form of a pdb file. To calculate the theoretical energy of the conformation, we used the Dope score, which is the statistical potential calculated for a certain distance between

two atoms of a protein [11], and which was retrieved in a text file via the Modeller program [12]. Here, we only used the alpha carbons of each residue in the model protein, but it is possible to use other atoms if relevant. To read those different files, the python libraries Biopython (for the fasta file) and Pandas (for the Dope scores file) were used.

We have a protein of interest of length n residues, and a model structure of length m residues. The first step in this double dynamic programming implementation is to create a set of low-level matrices that will calculate the optimal alignment of the protein of interest to the model structure for each of the fixed pairs $[i, j]$, $\forall i \in \{1, \ldots, n\}$, $\forall j \in \{1, \ldots, m\}$, thus creating n*m low-level matrices (see example: fig. 1). For each of these matrices, each cell with coordinates $[k, l]$, $\forall k \in \{1, \ldots, n\}$, $\forall l \in \{1, \ldots, m\}$ will be iteratively considered, after initialisation of the line $l = 0$ and the column $k = 0$ with the gap penalty, whis is a score deducted to penalise the introduction of gaps in the alignment.

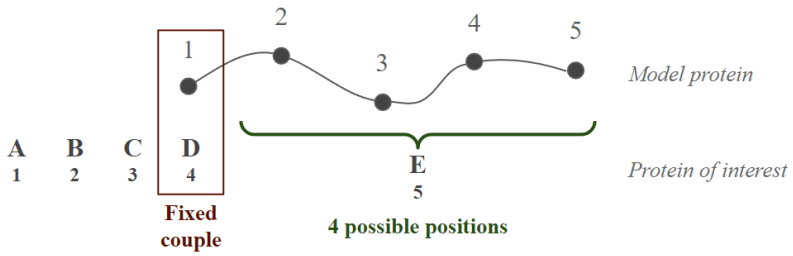


Figure 1: Schematic representation of the alignment for the example of the fixed pair [4, 1]. There are four different possible alignments here, where residue E aligns with residue 2, 3, 4 or 5 of the model protein.

The first step in this matrix traversal is to block all the cells $[k, l]$, such that $k < i$ and $l \geq j$, and such that $k \geq i$ and $l < j$ by imposing infinite numbers in them, to force the alignment of the residue i at the position of j. The Dope score will then be calculated between the pair $[i, j]$ and each of the other pairs $[k, l]$ with $i \neq k$ and $j \neq l$ (see example: fig. 2), then a global alignment will be done according to the Needleman-Wunsch algorithm [8]. Since the aim is to minimise the overall energy, the minimum value will be chosen according to :

$$L_{i,j}(k,l) = \min \begin{cases} L_{i,j}(k-1, l-1) + \text{dope}(i,j,k,l) \\ L_{i,j}(k, l-1) + \text{G}, \\ L_{i,j}(k-1, l) + \text{G}. \end{cases}$$

with $L_{i,j}(k,l)$ the cell of coordinates $[k,l]$ for the low level matrix $[i,j]$ and G the gap penalty (set at 0 for this project).

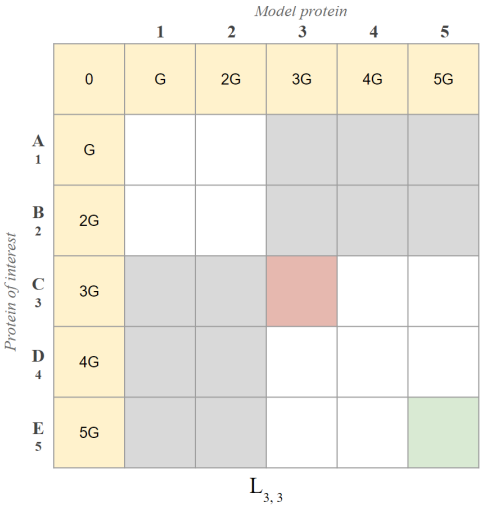


Figure 2: Schematic representation of the $L_{3,3}$ matrix before filling. The yellow cells do not correspond to residues, and have been initialised with G, the gap penalty. The grey cells represent the inaccessible cells, automatically filled with an infinite value to force the alignment of i and j, for which the cell is in red. Finally, the green cell represents the energy score of the global alignment of the $L_{3,3}$ matrix, which can then be used in the high-level matrix.

The second part of the dynamic programming is about a single, high-level matrix. To fill this matrix, we followed the same Needleman-Wunsch algorithm as described above, but instead of using the Dope score, we used the score from each of the low-level matrices for each pair $[i, j]$, which is the global score (green cell in the example: fig. 2) of each of the low-level matrices.

The final step of the double dynamic alignment is backtracking, i.e. reading the global alignment matrix starting from the end, to obtain the optimal alignment between the sequence of interest and the model protein. To do this, when creating each of the matrices, we also created a second matrix for which each $[k, l]$ cell contained the number 1, 2 or 3 depending on whether this cell came from the score of the cell on the diagonal, above or left respectively (see example: fig. 3). By moving up this matrix, we then obtain the position of the residues aligned with each other, and the gaps inserted. Thus, in order to run the whole program, we create $n * m + 2$ matrices of $(n + 1) * (m + 1)$ dimensions ; the algorithmic complexity of this program in both time and space is $O(n^2m^2)$.

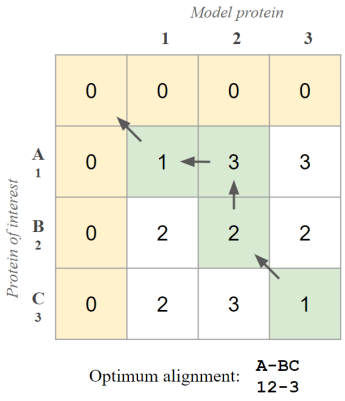


Figure 3: Schematic representation of an alignment matrix, for $n = 3$ and $m = 3$. The yellow cells do not correspond to residues, and have been initialised with the value 0. The green cells represent the optimum path: starting from cell $[k, l]$, if the previous cell (in green) is diagonal, the residues k and l are aligned. If it is at the top, a gap is inserted in the model structure. If it is on the left, a gap is inserted in the sequence.

2.2 Time optimisation

All computational analyses were performed on a personal computer equipped with an AMD Ryzen 5 5600H processor. This CPU features 6 physical cores and 12 threads (with 2 threads per core).

The complexity of this program being $O(n^2m^2)$, reducing running time would tremendously facilitate the process. For this, we parallelised the low-level matrices management, as they are independent and the most time-consuming process. We used the native python package *multiprocessing*, which allows the parallel execution of different processes inside the script, and tested it on the protein 5K2L (table 1), which was aligned to its own structure with and without parallelisation, for a total of 10 times each. We also tried using Numba, which is a python library that aims to optimise performances by generating specialized code for different array data types, and which is effective when used with Numpy. However, Numba doesn't allow the use of Pandas or handmade classes, which we used to create protein objects.

2.3 Data analysis

Name	Type	N	Description
1HNR	α	47	DNA binding domain of a nucleoid-associated protein, H-NS, from Escherichia coli
1V92	α	46	p47, a major adaptor of the AAA ATPase, from Escherichia coli
2PDD	α	43	Peripheral subunit-binding domain of dihydrolipoamide acetyltransferase, from Bacillus stearothermophilus
1ED7	β	45	Chitin-binding domain, from Bacillus circulans
1YWJ	β	41	Structural basis for peptide recognition by the FBP11WW1 domain, from Homo sapiens
2YSF	β	40	Solution structure of the fourth WW domain from the human E3 ubiquitin-protein ligase
1E0G	$\alpha+ \beta$	48	Lysm Domain from E.Coli Membrane Bound Lytic Murein Transglycosylase D
1QHK	$\alpha+ \beta$	47	N-terminal domain of Saccharomyces cerevisiae RNase HI
5K2L	$\alpha+ \beta$	49	Chitin oligosaccharide binding site, from Volvox carteri

Table 1: List of proteins used for data analysis

In order to test the script on several types of proteins, we chose 9 unique-chained proteins with a number of residues N between 40 and 50, subdivided into three fold types : all alpha (α), all beta (β), and alpha and beta ($\alpha + \beta$)(table 1). All these proteins were taken from the PDB database, and found via the "advanced search" tab. We created a second python script, which runs the main program on all template proteins in a directory, for a given sequence of interest. It then returns a single text file containing the optimum alignment, and the global energy score for each different alignment. This text file can then be read by a third python script, which outputs a histogram of the global energy value for each template (see example: fig. 4), as well as a histogram of the proportion of gaps in each alignment. For this, we used the python library Matplotlib.

3 Results and conclusion

3.1 Time optimisation

As stated before, the protein 5K2L (table 1) was aligned to its own structure. The mean runtime without parallelisation was of 9'17 (9'18 in CPU time) against 1'51 (21'12 CPU time) with parallelisation. The time difference is very significant ; we chose to keep parallelisation in the main program, which helped tremendously to obtain the results faster. The global run time for all 81 alignments was of about 2 hours in real time, and 20 hours in CPU time.

3.2 Data analysis

We observed little to no correlation between the folding type of the protein of interest and the type of the top-ranking (or rather low-ranking, since the most negative score is considered the most effective) template proteins. In fact, for all nine sequences of interest, the 4 lowest scores were attributed to exactly the same 4 proteins, ordered in almost the same way each time : 5K2L, 1V92, 1E0G, and 1QHK (see example: fig. 4). This uniformity seems to indicate inherent biases, that could be attributed to the algorithm or data. These four proteins seem to come from different organisms and have different functions, but three of them are of the folding type $\alpha + \beta$, and they all are at the higher end of the length spectrum (table 1) ; they could also possess other structural characteristics that make them effective across different protein sequences. It is worthy of note though, that among 3 of the 5 proteins that were not consistently in the top four, the sequence of interest was aligned to its own template structure in 5th place.

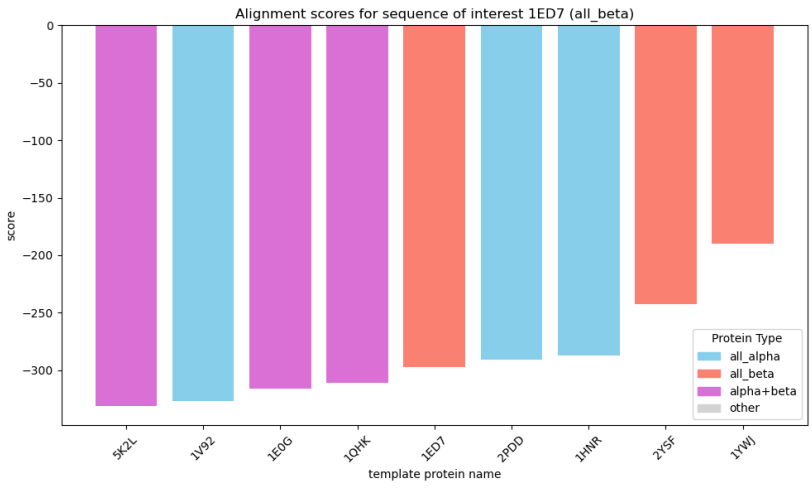
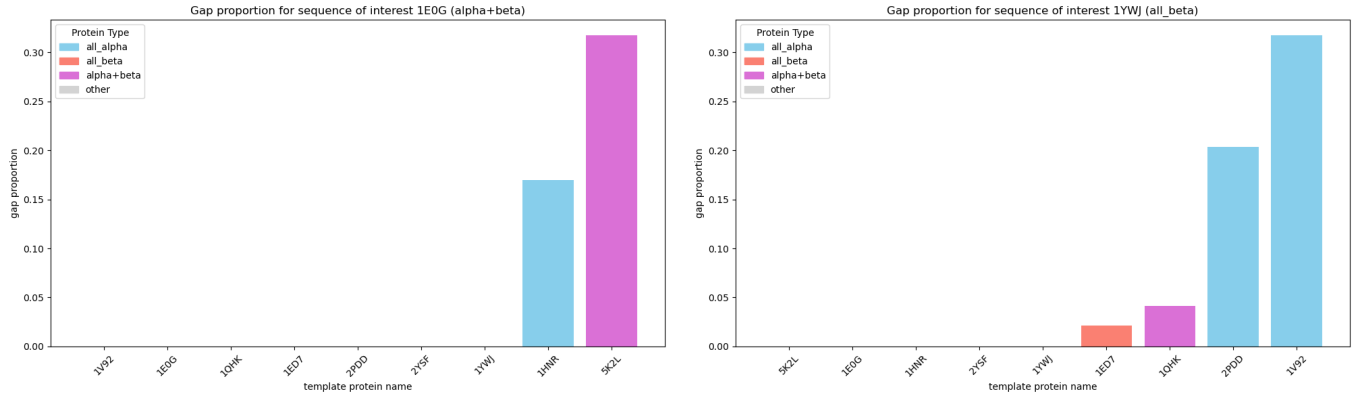


Figure 4: Alignment scores histogram for protein of interest 1ED7 (table 1) aligned with each of the 9 template proteins.

Surprisingly, in all alignments, a gap proportion of approximately 0.3 was found for either 5K2L (see example: fig. 5a) or 1V92 (see example: fig. 5b), but never both at the same time. These two proteins -which were part of the 4 top-runners when considering the alignment score- when used as sequences of interest, were also the only ones to have a gap proportion over 0.05 with their own structures. This may indeed indicate unique structural properties, such as specific motifs or regions causing misalignment or difficulty in threading, leading to gaps. Overall, to investigate if these inconclusive results can be explained by the data, it would be relevant to use a more diverse set of proteins, especially with regards to the number of residues. Globally though, the complexity of this algorithm makes it difficult to use large datasets without it being too time-intensive.



(a) 1E0G *(b) 1YWJ*

Figure 5: Gap proportion histograms for proteins 1E0G and 1YWJ (table 1) aligned with each of the 9 template proteins.

4 Discussion

Beyond running the program with new, more diverse and more qualitative data, there were a lot of optimisation ideas that we couldn't develop due to time constraints, and that would be interesting for the dual goal of reducing the running time of the program and creating a more qualitative alignment. Firstly, we could try reading all external files (the dope scores file, and the fasta and pdb files) by handmade methods. While python libraries are optimised to be as efficient as possible for these kinds of tasks, this could allow us to use Numba on all matrix-handling, thus maybe reducing the total running time of the program.

The algorithm choices themselves could also be revised at several levels : the gap penalty was implemented in the algorithm but was not used as it was set to 0 ; trying out different values, as well as implementing a different penalty for gap extensions could help ameliorate alignments. We could also try changing the score used to build the high-level matrix ; instead of using the global alignment score (bottom-right cell value of each low-level matrix), we could try out the optimum path score, meaning the sum of all cell values on the optimum path of each low-level matrix. Lastly, the backtracking algorithm could be revised with a more efficient one, so as to ameliorate the time and space consumption, or even so as to test each optimum path (in the case of several), instead of extracting only one of these paths like we did in the current algorithm.

While our re-implementation of the threading approach using double dynamic programming has highlighted the potential for numerous improvements, it also raises important questions about the relevance of this algorithm in light of recent advances in predictive modelling of protein structures. The emergence of machine learning-based models like AlphaFold [3] has revolutionised our understanding of protein folding by offering accurate and efficient predictions compared to traditional methods like threading. Future efforts might be better directed towards integrating these predictive models or exploring hybrid approaches.

References

[1] Linus Pauling and Robert B. Corey. "Atomic Coordinates and Structure Factors for Two Helical Configurations of Polypeptide Chains". In: *Proceedings of the National Academy of Sciences of the United States of America* 37.5 (May 1951), pp. 235–240. ISSN: 0027-8424. URL: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1063348/` (visited on 09/10/2024).

[2] Gonzalo López et al. "Assessment of predictions submitted for the CASP7 function prediction category". In: *Proteins* 69 Suppl 8 (2007), pp. 165–174. ISSN: 1097-0134. DOI: `10.1002/prot.21651`.

[3] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (Aug. 2021). Publisher: Nature Publishing Group, pp. 583–589. ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2. URL: https://www.nature.com/articles/s41586-021-03819-2 (visited on 09/10/2024).

[4] Rik K. Wierenga, Peter Terpstra, and Wim G. J. Hol. "Prediction of the occurrence of the ADP-binding -fold in proteins, using an amino acid sequence fingerprint". In: *Journal of Molecular Biology* 187.1 (Jan. 5, 1986), pp. 101–107. ISSN: 0022-2836. DOI: 10.1016/0022-2836(86)90409-2. URL: https://www.sciencedirect.com/science/article/pii/0022283686904092 (visited on 09/10/2024).

[5] D. T. Jones, W. R. Taylort, and J. M. Thornton. "A new approach to protein fold recognition". In: *Nature* 358.6381 (July 1992). Publisher: Nature Publishing Group, pp. 86–89. ISSN: 1476-4687. DOI: 10.1038/358086a0. URL: https://www.nature.com/articles/358086a0 (visited on 09/10/2024).

[6] Jian Peng and Jinbo Xu. "Low-homology protein threading". In: *Bioinformatics* 26.12 (June 15, 2010), pp. i294–i300. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btq192. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2881377/ (visited on 09/10/2024).

[7] Thomas H. Cormen et al. *Chapter 14 - Introduction to Algorithms, fourth edition*. Google-Books-ID: RSMuEAAAQBAJ. MIT Press, Apr. 5, 2022. 1313 pp. ISBN: 978-0-262-36750-9.

[8] Saul B. Needleman and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of Molecular Biology* 48.3 (Mar. 28, 1970), pp. 443–453. ISSN: 0022-2836. DOI: 10.1016/0022-2836(70)90057-4. URL: https://www.sciencedirect.com/science/article/pii/0022283670900574 (visited on 09/11/2024).

[9] D. T. Jones, R. T. Miller, and J. M. Thornton. "Successful protein fold recognition by optimal sequence threading validated by rigorous blind testing". In: *Proteins* 23.3 (Nov. 1995), pp. 387–397. ISSN: 0887-3585. DOI: 10.1002/prot.340230312.

[10] David Jones. "Chapter 13 - THREADER: protein sequence threading by double dynamic programming". In: *New Comprehensive Biochemistry*. Ed. by Steven L. Salzberg, David B. Searls, and Simon Kasif. Vol. 32. Computational Methods in Molecular Biology. Elsevier, Jan. 1, 1998, pp. 285–311. DOI: 10.1016/S0167-7306(08)60470-6. URL: https://www.sciencedirect.com/science/article/pii/S0167730608604706 (visited on 09/10/2024).

[11] Min-yi Shen and Andrej Sali. "Statistical potential for assessment and prediction of protein structures". In: *Protein Science : A Publication of the Protein Society* 15.11 (Nov. 2006), pp. 2507–2524. ISSN: 0961-8368. DOI: 10.1110/ps.062416606. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2242414/ (visited on 09/11/2024).

[12] Narayanan Eswar et al. "Comparative protein structure modeling using Modeller". In: *Current Protocols in Bioinformatics* Chapter 5 (Oct. 2006), Unit–5.6. ISSN: 1934-340X. DOI: 10.1002/0471250953.bi0506s15.