Fall 12-2-2022

Improving Adjacency List Storage Methods for Polypeptide Similarity Analysis

Arianna Swensen

Improving Adjacency List Storage Methods for Polypeptide Similarity Analysis

by

Arianna Nicole Swensen

A thesis submitted to the faculty of The University of Mississippi in partial fulfillment of the requirements of the Sally McDonnell Barksdale Honors College.

Oxford

May 2023

Approved by

_____

Advisor: Dr. Yixin Chen

_____

Reader: Dr. Charlie Walter

_____

Reader: Dr. Sushil Mishra

For my parents,
Shelby,
Spencer,
Caroline,
DaJ'ai and Hailey,
and of course,
the porch gang.

I couldn't have done this without your help.

Thank you.

Acknowledgements

I thank Dr. Yixin Chen, Dr. Charlie Walter, Dr. Sushil Mishra, and Dr. Wu Xu for their advisory remarks and assistance. This work was supported by faculty and staff in the Sally McDonnell Barksdale Honors College and the University of Mississippi Computer Science Department.

Abstract

Arianna Nicole Swensen: Improving Adjacency List Storage Methods for Polypeptide Similarity Analysis

(under the direction of Dr. Yixin Chen)

Protein design is a complex biomolecular and computational problem. Working on increasingly large protein folding problems requires an improvement in current analysis methods available. This work first discusses various methods of protein design, including *de novo* protein design, which is the primary focus of this thesis. Then, a new approach utilizing a B+ tree to effectively store and query a graph of keys and vertices is proposed in order to store the number of times two polypeptides are considered to be similar. This approach is found to have a reduction in time complexity from current mapping methods and thus provides a new approach by which to compute similar metrics.

Table of Contents

Introduction……1

De Novo Protein Design Approaches……...……..1

Design Approach……4

Time Complexity Analysis……6

Insertion……...7

Search……..7

Conclusion……...8

List of References……...9

Introduction

The study of protein design has seen significant advances in the past few years. [1, 2, 3]. *De novo* protein design, which attempts to construct an amino acid sequence that adopts predescribed folds, is critical to understanding how sequencing and structure relate to each other [4]. However, most research has been focused on methods other than *de novo* protein construction, as it has limitations that have not yet been overcome.

These other methods can be then classified into two basic approaches: template methods and non-template methods. Template methods have significantly improved, but non-template methods based in machine learning have become the most popular as they generally are able to provide more efficient large-scale predictions [5]. One particular area of interest is combining well-established methodologies with deep learning networks to further increase efficiency [6].

This work seeks to focus on specifically *de novo* design and addressing the general resource problem. Right now, *de novo* design requires large computation time and resources and thus can only be feasibly applied to small proteins. This work seeks to address the computational issue by proposing a new methodology for storing the number of occurrences wherein two polypeptides are considered similar by a given metric. Defining a structure optimized for large read-write access offers several benefits, including but not limited to creating polypeptide similarity vectors.

In Section 2, I will discuss how this methodology can offer support to existing approaches of *de novo* protein design and redesign. In Section 3, the suggested computational approach will be described. In Section 4, the suggested approach will be analyzed using worst-case time complexity analysis. Section 5 will summarize this work's contributions and identify areas for further exploration.

De Novo Protein Design Approaches

A primary topic in research regarding protein design is the development of new and useful proteins that have stable

structures with a prescribed fold. A common approach involves selecting polypeptides from combinatorial libraries to create a sequence [4]. Protein redesign or design can also occur as a result of shuffling these polypeptides [7]. However, the potential sequences that can occur are in the order of x^n possibilities, where x is the number of amino acids and n is the number of amino acids in the sequence. Even with modern supercomputing resources available, generating all the resulting structures is not feasible.

Given this information, random selection of amino acids is not a computationally effective method of *de novo* protein design. Designed combinatorial libraries that use binary patterning have been shown to have promising results in reducing the amount of computational time required to find sequences that generate desired structures [8]. Newer research proposes that the reuse of fragments across various proteins may allow for more stable building of hybrid proteins [9].

Experimental creation of new proteins can take two routes. Proteins can be created *de novo*, as discussed previously, selecting polypeptides from collections or at random. New proteins may also be generated as a redesign of an existing protein, through evolution of an existing protein with a functional folding structure [10]. Iterative *de novo* development in particular has been used for experimental development in small proteins, in an effort to investigate the sequence determinants of folding [11]. This iterative design technique can be extended to genetic algorithm applications, such as in [12]. Other work supports the development of sequential stabilization in order to predict folding pathways and the tertiary structure based on only the primary sequence as an input [13].

Recent work in developing comparison based hierarchical structures [14] suggests that more analysis of individual pairs of polypeptides can provide insight into sequence and structure relationships. This work provides the necessary computational methodology to compute these values for future work on implications in protein structure.

The methodology used for this work to evaluate pairs of polypeptides is as follows:

1. Two polypeptides, δ and ϵ, are selected and keys k_δ and k_ϵ are generated for the polypeptides.
2. The distance Δ_1 between δ and ϵ is calculated using the following formula, where i_δ is the residue number of δ in the primary sequence, and i_ϵ is the residue number of ϵ in the primary sequence:

$$\Delta_1 = |i_\delta - i_\epsilon|$$

3. The distance Δ_3 between δ and ϵ is calculated using the following formula, where $(x_\delta, y_\delta, z_\delta)$ is the centroid of the coordinates of the amino acids making up δ, and $(x_\epsilon, y_\epsilon, z_\epsilon)$ is the centroid of the coordinates of the amino acids making up ϵ:

$$\Delta_3 = \sqrt{(x_\delta - x_\epsilon)^2 + (y_\delta - y_\epsilon)^2 + (z_\delta - z_\epsilon)^2}$$

4. If λ is a parameter determining if δ and ϵ are "far" from each other, and μ is a parameter determining if δ and ϵ are "close" to each other, $\forall\delta, \epsilon$, where $\Delta_1 \geq \lambda$ and $\Delta_3 \leq \mu$, The vector $(k_\delta, k_\epsilon, c)$ should be created or updated with a number of times indicating how often this occurs across the given set of protein data.

A simple criterion for similarity is described above, but more complex methods of calculating similarity are available, with the methodology described above being used for the sake of focusing on the computation problem at hand. The most effective way to compute similarity between different polypeptides to support various protein design and redesign algorithms is outside of the scope of this work and a topic for further research.

Creating a means in which to effectively and efficiently compute similarity between polypeptides based on various effective criteria offers the means to build combinatorial databases specific to the problem; alternatively, it provides means to direct iterative evolution of a protein designed *de novo* or an existing natural protein. The primary contribution of this work is an optimization of this

computation in a generalized manner, in an attempt to reduce CPU time necessary to compute the overall similarity vectors for a given dataset.

Design Approach

In order to meet the requirements set forth in the previous section, a data structure D must meet the following criteria:

1. That it stores a set of vectors v composed of vectors in the form $(k_\delta, k_\epsilon, c)$;
2. That v can be queried by a key, k_δ and return the subset of vectors v_{k_δ} where v_{k_δ} is all vectors that contain k_δ;
3. That v can be queried by a pair of keys, k_δ and k_ϵ and return the vector v_{k_δ, k_ϵ} where v_{k_δ, k_ϵ} is the vector $(k_\delta, k_\epsilon, c)$;
4. and that both query and insertion can be performed in $O(n)$ time or less to support large sets of data in a way that reduces necessary computational time.

A mathematical representation of these vectors can be formalized as an undirected graph G with the set of vertices V and edges E, with each edge having a weight w. [Formally, $G = (V, E)$.]

Assuming proteins contain a number of polypeptides p, and number of proteins n exist in the dataset, the amount of potential pairs c can be defined as:

$$c = C(np, 2) = \frac{(np)!}{2!(np-2)!}$$

c becomes increasingly large as either n or p are increased. However, the significant values are limited by the bounds established in the polypeptide similarity metric described above, which means that there will be generally a sparse set of pairs c_ϕ for all c that are useful and should be included in the computations.

Thus, while an adjacency matrix generally provides $O(1)$ search time, it is infeasible for G to be stored in this manner as it will require $O(|V|^2)$ storage space, where V is equal to the number of unique polypeptides in all proteins n.

The set of adjacency lists a that must be stored are in the form (k, v) where $k \in V$ and v, which may also be referred to as a_k, contains all of the edges E as a list of the vertices V that are connected to k and their respective weights w. a requires significantly less storage space, using only $O(|V| + |E|)$, where $E = c_{\phi}$. Querying for an edge will then provide $O(|V|)$ search time in the case that a vertex V has an edge with every other vertex in G, with a more precise time complexity $\theta(a_k)$ when searching for an edge that connects vertex k to some other vertex $v \in V$.

Thus, the challenge becomes aggregating the set a in a manner that makes it computationally efficient to find the adjacency list a_k for updates and queries. Implementation will rely heavily on the mapping of $k \rightarrow a_k$ as a (k, v) pair.

Several methods exist for mapping key-value pairs. One common approach to mapping adjacency lists as (k, v) pairs is utilization of the HashMap data structure. A hash algorithm is determined, varying based on the data and the size of the HashMap, to insert (k, v) pairs into the HashTable at a specific location, minimizing collisions. Determining a hashing algorithm to handle an unknown quantity of data is still an ongoing research problem, with the general assumption that HashTables generally do not undergo frequent insertions [15].

TreeMap implementations offer an alternative to this approach while still providing effective time and space efficiency. While many implementations of TreeMap use a red-black tree, including the Java implementation [16], I propose that a B+-tree implementation offers several benefits in this particular case. Additional implementations have been proposed, including work to optimize search time in TreeMap [16] and recent work to implement Graph structures in a universally applicable manner [18].

The B+-tree, depicted in Fig. 1, is a variant of the B-tree and a m-ary tree. All values are stored within the leaf nodes.

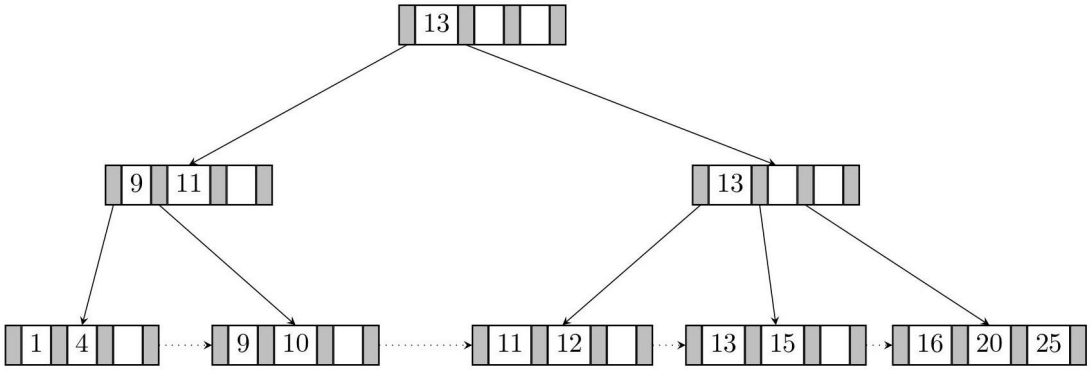

Fig. 1. A B+-tree of order 4.

The tree is sorted by key, and the leaf nodes are connected by pointers. This allows for rapid random and sequential access.

B+-trees are generally used in database applications and file organization applications [19]. As graph processing has become more common, more graph databases are using B+-trees or B+-tree variants [20] as a way to implement their file organization.

This has expanded into security applications for cloud computing [21] applications, favored over red-black trees due to the property of a lower height, originally designed to reduce query time and disk accesses. Given the size of G, I propose that the B+-tree provides an optimal tool to analyze, write, and read information from the disk to store information about polypeptide similarity analysis for comparison across metrics.

Time Complexity Analysis

The height of a red-black tree is defined by the following equality statement, where n is the number of keys in the tree:

$$h_{RB} \leq 2log_2(n+2) - 2$$

The height of a B+-tree is defined by the following equality statement, where n is the number of keys in the tree and m is the order of the tree:

$$h_{B+} \leq 1 + log_{\lceil \frac{m}{2} \rceil}(\frac{n+1}{2})$$

Then the value of m when $h_{B+} = h_{RB}$ can be approximated by the following equation, assuming the value of m is divisible by 2:

$$1 + log_{(\frac{m}{2})}(\frac{n+1}{2}) = 2log_2(n+2) - 2$$

$$log_{(\frac{m}{2})}(\frac{n+1}{2}) = 2log_2(n+2) - 3$$

$$\frac{ln(\frac{n+1}{2})}{ln(\frac{m}{2})} = 2log_2(n+2) - 3$$

$$ln(\frac{n+1}{2}) = 2log_2(n+2)ln(\frac{m}{2}) - 3ln(\frac{m}{2})$$

$$ln(\frac{m}{2})(2log_2(n+2) - 3) = ln(\frac{n+1}{2})$$

$$ln(\frac{m}{2}) = \frac{ln(\frac{n+1}{2})}{2log_2(n+2) - 3}$$

$$\frac{m}{2} = \frac{ln(\frac{n+1}{2})}{e^{2log_2(n+2) - 3}}$$

$$\frac{m}{2} = e^{\frac{ln(\frac{n+1}{2})}{2log_2(n+2) - 3}}$$

$$m = e^{\frac{ln(\frac{n+1}{2})}{2log_2(n+2) - 3}} \cdot 2$$

This value of m can then be approximated to 3 as n grows sufficiently large.

The following analyses assume that $m > 3$ and n is sufficiently large, and thus,

$$h_{B+} < h_{RB}.$$

Insertion

A red-black tree is a balanced binary search tree. As such, the time complexity for inserting a new node is $O(h_{RB})$. Then, rebalancing must occur. Recoloring a node occurs in $O(1)$ time, but in worst case scenario re-establishing the red-black properties requires recoloring up to the root, which requires h_{RB} recolors, occurring in $O(h_{RB})$ time. Thus, the worst-case cost of insertion can be generalized as the following:

$$O(h_{RB}) + O(h_{RB}) \in O(h_{RB})$$

In a B+-tree, all of the data is stored at the leaf nodes, thus there are n leaf nodes. Each internal node N has $m + 1$ children, where m is the order of the B+-tree. To insert, the cost is $O(m)$ within a node to maintain the sorted property of the internal nodes. At worse, this must occur h_{B+} times, as nodes are split.

Thus, the worst-case cost of insertion can be generalized as the following:

$$O(m \cdot h_{B+}) \in O(h_{B+})$$

Thus we can say that insertion to a B+-tree, in the worst case, will be faster than a red-black tree.

Search

Similarly to insertion, searching a red-black tree for a specific key is of time complexity $O(h_{RB})$ as the worst case is searching for a key at a leaf node.

B+ trees have up to m keys in an internal node, with all of the data being stored at the leaf nodes. Searching an internal node, using binary search, is of time complexity $O(log_2 m)$, and this occurs $O(h_{B+})$ times.

Then, the worst-case time complexity will be bounded by the following equation:

$$O(log_2 m \cdot h_{B+}) \in O(h_{B+}).$$

Thus we can say that searching a B+-tree, in the worst case, will be faster than a red-black tree.

Choosing an optimal m for reducing search complexity should be considered on a case-to-case basis, given the number of keys n will affect this analysis, and thus should be considered carefully when creating a B+-tree based map.

In the case of an adjacency list of graph G being mapped, regardless of tree implementation, there is the consideration of the lists stored at the tree nodes, which are generally searched in $O(n)$ time.

Conclusion

In this work, I have demonstrated that a B+ tree may provide significant improvements to existing mapping algorithms available that may support polypeptide similarity analysis.

Reducing the time complexity is crucial to research regarding protein folding and the causes and prediction of folds among primary sequences that do not have a known tertiary structure.

A time complexity analysis has been performed on the B+-tree and the red-black tree, the latter of which is the most common way to implement TreeMap classes. The B+-tree provides significant improvements to searching and insertion when dealing with disk access by reducing the overall height of the tree.

Given a sufficiently large list of polypeptides, stored as keys, and a carefully selected m, the number of keys present in one node of the B+-tree, these queries can be optimized for performance while storing this information on the disk for future analysis and comparison to other similarity metrics than the one described in this work.

Future work to optimize the performance of the B+-tree by implementing bulk-loading, buffers, and other insertion and query optimization methodologies can be done to increase the efficacy of the solution described in this work.

Additionally, the polypeptide similarity metric described in this work is very simplistic, and further algorithms can be developed to increase overall understanding of the polypeptides that may determine the translation from a primary sequence into a tertiary structure.

List of References

[1] B. Kuhlman and P. Bradley, “Advances in protein structure prediction and design,” *Nature Reviews Molecular Cell Biology*, vol. 20, no. 11, pp. 681–697, Aug. 2019, doi: 10.1038/s41580-019-0163-x.

[2] D. N. Woolfson, “A Brief History of De Novo Protein Design: Minimal, Rational, and Computational,” *Journal of Molecular Biology*, vol. 433, no. 20, p. 167160, Oct. 2021, doi: 10.1016/j.jmb.2021.167160.

[3] A. N. Lupas, J. Pereira, V. Alva, F. Merino, M. Coles, and M. D. Hartmann, “The breakthrough in protein structure prediction,” *Biochemical Journal*, vol. 478, no. 10, pp. 1885–1890, May 2021, doi: 10.1042/bcj20200963.

[4] M. D. Finucane, M. Tuna, J. H. Lees, and D. N. Woolfson, “Core-Directed Protein Design. I. An Experimental Method for Selecting Stable Proteins from Combinatorial Libraries,” *Biochemistry*, vol. 38, no. 36, pp. 11604–11612, Aug. 1999, doi: 10.1021/bi990765n.

[5] L. Wei and Q. Zou, “Recent Progress in Machine Learning-Based Methods for Protein Fold Recognition,” *International Journal of Molecular Sciences*, vol. 17, no. 12, p. 2118, Dec. 2016, doi: 10.3390/ijms17122118.

[6] S. M. Kandathil, J. G. Greener, and D. T. Jones, “Recent developments in deep learning applied to protein structure prediction,” *Proteins: Structure, Function, and Bioinformatics*, vol. 87, no. 12, pp. 1179–1189, Oct. 2019, doi: 10.1002/prot.25824.

[7] L. Riechmann and G. Winter, “Novel folded protein domains generated by combinatorial shuffling of polypeptide segments,” *Proceedings of the National Academy of Sciences*, vol. 97, no. 18, pp. 10068–10073, Aug. 2000, doi: 10.1073/pnas.170145497.

[8] M. H. Hecht, A. Das, A. Go, L. H. Bradley, and Y. Wei, “De novo proteins from designed combinatorial libraries,” *Protein Science*, vol. 13, no. 7, pp. 1711–1723, Jul. 2004, doi: 10.1110/ps.04690804.

[9] N. Ferruz *et al.*, “Identification and Analysis of Natural Building Blocks for Evolution-Guided Fragment-Based Protein Design,” *Journal of Molecular Biology*, vol. 432, no. 13, pp. 3898–3914, Jun. 2020, doi: 10.1016/j.jmb.2020.04.013.

[10] C. Jäckel, P. Kast, and D. Hilvert, “Protein Design by Directed Evolution,” *Annual Review of Biophysics*, vol. 37, no. 1, pp. 153–173, Jun. 2008, doi: 10.1146/annurev.biophys.37.032807.125832.

[11] G. J. Rocklin *et al.*, “Global analysis of protein folding using massively parallel design, synthesis, and testing,” *Science*, vol. 357, no. 6347, pp. 168–175, Jul. 2017, doi: 10.1126/science.aan0693.

[12] S. C.-H. . Pegg, J. J. Haresco, and I. D. Kuntz, “A genetic algorithm for structure-based de novo design,” *Journal of Computer-Aided Molecular Design*, vol. 15, no. 10, pp. 911–933, 2001, doi: 10.1023/a:1014389729000.

[13] A. N. Adhikari, K. F. Freed, and T. R. Sosnick, “De novo prediction of protein folding pathways and structure

using the principle of sequential stabilization," *Proceedings of the National Academy of Sciences*, vol. 109, no. 43, pp. 17442–17447, Oct. 2012, doi: 10.1073/pnas.1209000109.

[14] S. Kondra, F. Chen, Y. Chen, Y. Chen, C. J. Collette, and W. Xu, "A study of a hierarchical structure of proteins and ligand binding sites of receptors using the triangular spatial relationship‐based structure comparison method and development of a size‐filtering feature designed for comparing different sizes of protein structures," *Proteins: Structure, Function, and Bioinformatics*, vol. 90, no. 1, pp. 239–257, 2022.

[15] P. Fatourou, N. D. Kallimanis, and T. Ropars, "An Efficient Wait-free Resizable Hash Table," *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures*, Jul. 2018, doi: 10.1145/3210377.3210408.

[16] "TreeMap (Java Platform SE 7)," *docs.oracle.com*. https://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html

[17] F. Fauberteau and S. Midonnet, "Worst case analysis of TreeMap data structure," *2nd Junior Researcher Workshop on Real-Time Computing (JRWRTC'08)*, pp. 33–36, 2008.

[18] P. Fuchs, D. Margan, and J. Giceva, "Sortledton," *Proceedings of the VLDB Endowment*, vol. 15, no. 6, pp. 1173–1186, Feb. 2022, doi: 10.14778/3514061.3514065.

[19] D. Comer, "Ubiquitous B-Tree," *ACM Computing Surveys*, vol. 11, no. 2, pp. 121–137, Jun. 1979, doi: 10.1145/356770.356776.

[20] M. Besta *et al.*, "Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries," *arXiv preprint arXiv:1910.09017*, 2019.

[21] H. Shen, L. Xue, H. Wang, L. Zhang, and J. Zhang, "B+-Tree Based Multi-Keyword Ranked Similarity Search Scheme Over Encrypted Cloud Data," *IEEE Access*, vol. 9, pp. 150865–150877, 2021, doi: 10.1109/access.2021.3125729.