Computational Genomics Final Report

Kevin He, Chao Cheng Chuang, Jingkai Guo, Peijia Ye

December 2023

Abstract

Aligning sequencing reads against a protein reference database is a major computational bottleneck in metagenomics and other data-intensive evolutionary project. In this report, we developed a version of "Simple-DIAMOND" based upon "DIAMOND" and "PALADIN" in order to speed up the sequencing read protein alignment process while preserving the high accuracy crucial for gene identification and functional analysis. Our goal was to create a faster alternative to the more traditional Blastx method.

1 Introduction

As high-throughput DNA sequencing technologies improve, metagenomics has become an increasingly popular field [1]. In recent years, metagenomics has helped reveal many previously hidden details about functional homologies and new biological pathways [1,2]. These discoveries have had a huge impact in everything from improving crop yields to discovering new diagnosis pathways [3,4].

In order to study metagenomics, one of the critical problems that needs to be solved is the efficient alignment of DNA sequencing reads to a reference protein database [2]. On a small scale, BlastX can be used to solve this problem [2]. BlastX creates kmers from the query sequence and determines whether there are any seeds that can be extended [5]. However, due to the volume of DNA sequences that high-throughput sequencing generates for metagenomics, BlastX is oftentimes insufficient for processing high throughput DNA sequencing results [2]. In large-scale genomic projects, this limitation can significantly impede research progress. Therefore, enhancing the speed of these alignments without sacrificing accuracy is a critical objective in the field.

For our approach, we hoped to improve the alignment of DNA to protein by bringing together features of existing algorithms from DIAMOND and PALADIN, two successful software that align DNA to protein [1,2]. By doing so, we hope to create a new piece of software that can align DNA to protein faster. In particular, we wanted to incorporate the open reading frame filter from PALADIN, try different sketching methods for seed selection, and create a regional Smith-Waterman alignment.

2 Prior Work

2.1 Blastx

Blastx has long been a standard in the alignment of DNA to protein. Blastx works by creating 6 reading frames of proteins then aligning each reading frame against each reference protein sequence [5]. For each alignment, Blastx creates 3-mers of the query sequence and then iterates on a shifting window of 3 amino acids down each protein sequence [5]. For each amino acid sequence, Blastx checks whether or not there is a kmer match that will result in a score higher than than a threshold utilizing the Blosum62 penalty matrix. If there is a kmer which matches, the seed is extend by 3 on both sides. If it continues to remain

above the threshold, it is extended until the alignment falls below the threshold. The best alignment is the longest alignment. This method will provide for a very sensitive alignment, but will often be a very slow process.

2.2 DIAMOND

DIAMOND attempts to correct many of the problems that lead Blastx to be slow. One of the slowest part of the algorithm is the iterative comparison between the kmers against each 3 amino acid window within each reference protein sequence [2]. First, DIAMOND incorporates a double indexing scheme in order to quickly query the sequences of each protein [2]. Within this double indexing scheme DIAMOND uses a long seed in order to minimize the amount of kmer comparisons necessary [2]. Second DIAMOND only considers a fraction of all seeds possible by applying the minimizer sketching technique in order to find seeds. Only seeds that can extend 10 amino acids will be kept [2]. Afterwards, a Smith-Waterman alignment is applied to find the best possible alignments [2]. Given that DIAMOND claimed that it had improved the speed of DNA sequencing read alignment to protein by 20,000 times, we adopted DIAMOND to be our base model [2].

2.3 PALADIN

A year after DIAMOND was published, PALADIN was published and claimed to be 7-8x faster than DIAMOND [1]. PALADIN works by creating an Open Reading Frame filter first in order to remove the alignment of sequences that are unlikely to result in protein sequences [1]. Reads that have stop codons early on within all 6 reading frames are unlikely to translate into a significant protein and therefore are discarded [1]. Afterwards, this new fasta file containing all of the sequences that passed are retained [1]. Next, PALADIN applies a BWA Mem algorithm to align the 6 reading frames in protein space[1]. The results are subsequently queried against the Uniprot database [1]. Given that the BWA Mem algorithm and query against Uniprot database is fairly standardized, we decided to adopt the ORF filter into our model. We were also driven by the fact that BWA Mem was published in 2009 and given that DIAMOND was published in 2015. Given that the most novel contribution was the ORF filter, we decided to implement it into our own pre-processing [2,6].

3 Methods and software

3.1 Data and Data Preprocessing

For our data, we generated out synthetic data from the assembly of E. Coli K-12. We randomly sampled 5000 positions from the NCBI assembly of E. Coli each with 150 base pairs long to be our reads. We used the Uniprot Database for our reference protein database [12]. We stripped out all amino acids that were not part of the standard 20 amino acid bases due to Blossom62 being unable to score it and we ensured that our protein sequences were all at least 30 base pairs long due to the need to run a minimizer window of 30.

3.2 Open Reading Frame (ORF) Filter

The ORF filter is implemented as described in PALADIN. We first check through each of the 6 reading frames: 3 in the forward direction and 3 in the reverse direction for stop codons appearing within a certain threshold value. The threshold where we count stop codons not appearing early is calculated as the length of the sequence multiplied by the threshold value. If at least one reading frame doesn't have a stop codon before the threshold length, we accept that sequence for processing. ORFs are crucial for potentially coding proteins. Long ORFs usually indicate protein-coding genes. Sequences with stop codons in all frames,

suggesting non-coding regions, are excluded to focus resources on more promising protein-coding sequences.

Integrating the ORF filter into our algorithm optimizes the identification of protein-coding sequences and improves overall efficiency. We process the DNA query sequence data from a FASTA file, focusing on filtering out sequences that lack an Open Reading Frame (ORF) in any of the six frames. Additionally, a threshold is established to analyze only specific parts of the sequence.

3.3 Reduced Alphabet

In alignment with the DIAMOND configuration, we used an 11-group reduced amino acid alphabet to boost speed. This method grouped the 20 standard amino acids based on similar properties, enhancing the detection of conserved regions in protein sequences, particularly in distantly related proteins. [6] Seed selection in the alignment process relies on exact matches to this condensed alphabet, followed by a Smith-Waterman [7] alignment for matched seeds to identify precise sequence similarities.

3.4 Spaced Seed

In our implementation, we adopted the concept of spaced seeds[8,9], a sophisticated improvement in the seeding step of sequence alignment. Spaced seeds differ from traditional contiguous seeds in that they are longer and only a selected subset of positions within these seeds are utilized. This approach is characterized by two key aspects: the 'weight', which is the number of positions used, and the 'shape', which refers to the specific layout of these positions.

Theoretical analyses have demonstrated that a single spaced seed, if optimally shaped, can outperform a contiguous seed of the same weight. The reason for this enhanced performance lies in the unique arrangement of the spaced seed, which allows for a more efficient and effective identification of potential alignments, especially in cases of subtle or complex genetic similarities.

To further amplify sensitivity, multiple seed shapes can be employed. Each additional shape incrementally improves sensitivity while only causing a sublinear increase in computational time. This balance between sensitivity and computational efficiency is crucial for handling large-scale genomic data.

In line with DIAMOND's default settings, we utilized a set of four seed shapes in our implementation. These shapes vary in length (ranging from 15 to 24) and have a consistent weight of 12. By adopting these predefined shapes, we leverage their proven efficiency and effectiveness in sequence alignment.

3.5 Sketching

1. **Uniform sketching** The first sketching method we implemented is Uniform sketching. Uniform sketching;s benefit is that it allows for an even spread of seeds. Since we had seed masks, we only needed to match some of the positions within the seed to be called a match. We ended up picking seeds every 24 base pairs. Although picking every 24 base pairs to be a seed didn't reduce the size of the kmer index, it was certainly more spare than minhash and minimizer.

2. **Minhash sketching** The second sketching method we implemented is Minhash sketching. Minhash sketching allows a user to pick the a number of seeds with the smallest hashed values. This approach allows for a random spreading of seeds and is oftentimes a bit denser than uniform hash. We ensured there were always 5 seeds on both the protein reference and the query sequence.

3. **Minimizer sketching** The third sketching method we implemented is Minimizer sketching. The original DIAMOND implementation used a minhash sketching scheme. Minimizer sketching takes a window of size k and then runs a smaller window w within the window of size k to find the seed that match the minimal hash value of the rolling smaller window. Combined, these seeds represent the minimizer. We chose 30 for our large window size due to it being slightly bigger than our largest seed.

3.6 Sort-Merge Join Algorithm, Parallel Computing

In our research, we explored the implementation of the sort-merge join algorithm, a cornerstone of DIAMOND's double-indexing strategy. The sort-merge join is an efficient method for combining two sorted data sets and is particularly effective for large-scale data, common in genomic analysis.

The algorithm's process involves several key steps:

1. **Sorting:** Both data sets are initially sorted based on specific keys. In sequence alignment, these could be genetic markers or sequence identifiers.

2. **Merging and Joining:** Sequential comparison of the sorted data sets is performed. The datasets are merged at points where the keys match, determining the combination of data from the two sources.

3. **Efficiency:** The sort-merge join is generally more efficient for large datasets than other methods, such as hash-table approaches, due to minimized random data access.

4. **Genetic Analysis Application:** This method facilitates rapid and effective comparison of query sequences against reference databases, identifying matches based on sorted sequence identifiers.

In our study, we endeavored to adopt the sort-merge join algorithm, pivotal in DIAMOND's double-indexing method. This approach, inspired by database sort-merge joins, targets sorting seed sets from both queries and references. The essence of this method is to prepare and sort two seed lists, a task amenable to parallel processing. Typically, it combines radix clustering with a swift sorting mechanism to optimize large data set management.

However, our attempts to implement this algorithm faced challenges. We struggled with parallelizing the process, particularly in integrating the radix clustering[10] with an efficient sorting algorithm, hindering our ability to achieve the time and computational efficiencies characteristic of DIAMOND's sort-merge join.

For our implementation, we sorted our seed dictionary, keyed by seed sequences, using a multi-threaded sort - a parallel version of merge sort. In our tests with 10,000,000 seeds of length 4, the multi-threaded sort took 17.8 seconds, outperforming the single-threaded version (48 seconds) but lagging behind Python's built-in sort (5.3 seconds). The relative slowness of the multi-threaded sort is attributed to a few factors: Python's multi-threading is more of a simulation rather than true parallelism, leading us to use multi-processing which incurs significant initialization and context-switching overhead. Additionally, Python's built-in sort is implemented in CPython, offering a performance edge. Moreover, Python employs Timsort, a hybrid sorting algorithm generally faster than merge sort.

3.7 Smith-Waterman using Blosum62

In the context of our study, where we implemented the Smith-Waterman[7] algorithm for protein sequence alignment, the choice of the BLOSUM62[11] matrix as our scoring matrix plays a pivotal role. The BLOSUM62 matrix is particularly beneficial because it is based on empirically derived substitutions from protein blocks. This matrix effectively represents the likelihood of substitution between different amino acids, making it highly suitable for comparing proteins and detecting evolutionary relationships.

Using the Smith-Waterman algorithm with the BLOSUM62 matrix allows for a more nuanced and accurate alignment. The Smith-Waterman algorithm is renowned for its precision in local sequence alignment, identifying the most similar region between two sequences without being affected by other regions of lesser similarity. This specificity is crucial in protein sequence analysis, especially when detecting functional or evolutionary connections between sequences that may only have short regions of high similarity.

Furthermore, by configuring the gap opening penalty at 11 and the gap extension penalty at 1, as per DIAMOND's default settings, we strike a balance between penalizing gaps in the alignment and allowing for the natural variability seen in protein sequences. This setup helps in accurately aligning sequences with insertions or deletions, which are common in evolutionary divergent proteins.

3.8 Finding the best match

In our process of aligning a specific query sequence to a protein sequence in the reference database, we consider all six reading frames via the seeds within each reading frame to ensure we select the best reading frame score.

To prevent repetitive Smith Waterman Alignments, we designed a regional smith-waterman method which only performs an alignment between seed index - seed length to seed index + seed length. Any seeds falling within this region get eliminated. We continue this process until we run out of seeds. Once we have explored all six frames for all proteins, we select the highest Blosum score.

The chosen frame is then reported as the optimal alignment, along with its alignment score and the starting position of the match within the specific protein sequence.

This detailed information is recorded in the output file for each query sequence. Essentially, we repeat this thorough alignment process for each query sequence against every sequence in the reference database. This methodical approach ensures that we identify the most accurate and relevant alignments.

4 Results

4.1 Overall result

In Table 1, we present our overall results. Notably, our implementation achieves a matching score closely aligned with the original Diamond algorithm, reaching an impressive 97.34 percent. This performance indicates a high degree of accuracy and efficiency in our method, almost paralleling that of a well-established tool in the field. When compared to Blastx, our approach shows a minimal discrepancy, with only a two percent difference in the matching score. This highlights the effectiveness of our algorithm, demonstrating its capability to maintain high accuracy while potentially offering improvements in other aspects like speed or resource utilization.

	us vs blastx	**us vs diamond**	**diamond vs blast**
Number of Matches	38491	38356	39178
Matching Score	86.77%	97.34%	88.31%

Table 1: Comparison of matching scores and number of matches

4.2 Open Reading Frame (ORF) Filter

In PALADIN, applying an Open Reading Frame (ORF) filter effectively streamlines the dataset by eliminating sequences that do not contain ORFs, which are potential protein-coding regions. As evident from the data in the table, increasing the threshold for the ORF

filter leads to a reduction in the number of sequences, or 'reads', that require analysis. This filtering process is critical for focusing computational resources on sequences with a higher likelihood of biological significance.

The execution times for both Blastx and Diamond demonstrate a general trend of decrease as the threshold increases. This trend indicates that with higher ORF thresholds, fewer sequences pass the filter, thereby reducing the computational workload. As a result, both Blastx and Diamond require less time to process the data, highlighting the efficiency gains achieved through the application of an ORF filter. The table below encapsulates the relationship between the ORF filter threshold and the corresponding execution times for Blastx and Diamond, illustrating the effectiveness of ORF filtering in optimizing sequence analysis pipelines.

threshold	**number of reads**	**Blastx**	**Diamond**
0	10000	4min10s	1.653s
30	9971	4min7s	1.408s
40	9898	4min30s	1.103s
50	9752	4min48s	1.066s
60	9552	4min56s	0.992s
70	9293	4min47s	0.977s
80	9051	4min41s	1.127s
100	8618	4min26s	1.135s

Table 2: Comparison of threshold filtering percentages and their impact on the number of reads and processing time by Blastx and Diamond.

4.3 Sketching Techniques

Sketching Scheme	**Time**	**Accuracy against DIAMOND**
Uniform	80.89s	0
Minhash	198.29s	93.62
Minimizer	262.84s	87.23

Table 3: Processing time and accuracy for different sketching techniques

In this table, we present querying on a shortened version of our sequences with different sketching scheme. In our query version, we have 20 sequences querying against our protein reference database. The full version of randoms sequence takes nearly an hour to process and better takes advantage of the double indexing scheme. Overall, our accuracy for uniform hashing are very low and this doesn't come as much of a surprise as the uniform sketching technique has the lowest density of seeds. Without it's correct match present, we ended up picking many low scoring sequences that weren't correct matches. Minhash performed much better since we are guranteed 5 seeds for the reference. Minimizer did slightly worse, but that can be attributed to minimizers not guaranteeing 5 seeds like minhash does.

4.4 Smith-Waterman Efficiency

The regional Smith-Waterman algorithm enhances the efficiency of the seeding and extension phases compared to the conventional full Smith-Waterman approach, which involves comparing each query (across all six reading frames) against the complete protein database. The traditional method requires constructing a vast matrix to ascertain the optimal local alignment between the query and the entire sequence. This process is not only exceedingly time-consuming but also often results in slower performance than BLAST. When we originally ran the Smith-Waterman we took 1150 seconds in order to process one sample.

The regional Smith-Waterman approach focuses on extending seeds that indicate regions in the protein sequence with a higher likelihood of an optimal match. This targeted strategy significantly saves time by concentrating on the most promising areas for alignment.

5 Conclusions

From our results, we noticed a few things. First, the ORF filter was fairly ineffective. We didn't see the speed up that was published in the PALADIN paper. The biggest reason is probably because we selected reads from the Assembly which has minimal amounts of noise. In real life, we would likely see more sequencing errors and the ORF filter would likely be more efficient. Additionally, due to this sample being E.Coli, we have 1 contiguous ORF per gene that passes the filter rather than genes split up and being filtered out.

Second, we saw how different sketching techniques performed differently. Uniform sketching has an accuracy of 0 due to there being close to no seed hits. It's very rare to have seeds start at a multiple of 24 to match the seeds exactly. Minhash performed a bit better due to having a bit higher density of 5 seeds guaranteed. The minimizer had higher density than Uniform, but didn't always have 5 seeds. In order for an alternative sketching technique to work, there has to be enough density. We may even have to resample seeds as BWA Mem does to get better accuracy.

Moving forward, the aim is to enhance the overall efficiency of the program through the integration of parallel computing techniques. The strategy will involve distributing subsets of the query to multiple threads during the alignment process with the database. This distribution is anticipated to considerably accelerate the execution speed, paralleling the approach taken with minimizers. By leveraging the concurrent processing capabilities of modern computing architectures, we expect to achieve a substantial reduction in computational time and an increase in throughput for large-scale data analyses.

References

[1] Westbrook, Anthony, et al. "PALADIN: protein alignment for functional profiling whole metagenome shotgun data." *Bioinformatics* 33.10 (2017): 1473-1478.

[2] Buchfink, Benjamin, Chao Xie, and Daniel H. Huson. "Fast and sensitive protein alignment using DIAMOND." *Nature methods* 12.1 (2015): 59-60.

[3] Jansson, Janet. "Towards tera terra: Terabase sequencing of terrestrial metagenomics." (2011).

[4] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. "Basic local alignment search tool." *J Mol Biol.* 1990 Oct 5;215(3):403-10. doi: 10.1016/S0022-2836(05)80360-2. PMID: 2231712.

[5] Govender, Kumeren N., et al. "Metagenomic sequencing as a pathogen-agnostic clinical diagnostic tool for infectious diseases: a systematic review and meta-analysis of diagnostic test accuracy studies." *Journal of clinical microbiology* 59.9 (2021): 10-1128.

[6] Murphy, Lynne Reed, Anders Wallqvist, and Ronald M. Levy. "Simplified amino acid alphabets for protein fold recognition and implications for folding." *Protein engineering* 13.3 (2000): 149-152.

[7] Smith, Temple F., and Michael S. Waterman. "Identification of common molecular subsequences." *Journal of molecular biology* 147.1 (1981): 195-197.

[8] Burkhardt, Stefan, and Juha Kärkkäinen. "Better Filtering with Gapped Õ-Grams." *Fundamenta informaticae* 23 (2003): 1001-1018.

[9] Ma, Bin, John Tromp, and Ming Li. "PatternHunter: faster and more sensitive homology search." *Bioinformatics* 18.3 (2002): 440-445.

[10] Boncz, Peter A., Stefan Manegold, and Martin L. Kersten. "Database architecture optimized for the new bottleneck: Memory access." *VLDB*. Vol. 99. 1999.

[11] Henikoff, Jorja G., and Steven Henikoff. "[6] Blocks database and its applications." *Methods in enzymology*. Vol. 266. Academic Press, 1996. 88-105.

[12] UniProt Consortium. "UniProt: the universal protein knowledgebase in 2023." *Nucleic Acids Research*, 2023, 51.D1: D523-D531.