Trade-offs in Query and Target Indexing for the Selection of Candidates in Protein Homology Searches

Strahil Ristov[1⋆], Robert Vaser[2⋆⋆], and Mile Šikić[2,3]

[1] Ruđer Bošković Institute, Department of Electronics,
Bijenička 54, 10000 Zagreb, Croatia
[2] Faculty of Electrical Engineering and Computing,
Unska 3, 10000 Zagreb, Croatia
[3] Bioinformatics Institute,
Singapore 138671, Singapore
ristov@irb.hr, robert.vaser@fer.hr, mile.sikic@fer.hr

Abstract. We compare two recent similar and complementary indexing methods for fast seed discovery [10,12]. Both methods are based on the principle of counting matches on a diagonal with a goal to find the value and/or position of the best match between two sequences under Hamming distance on alphabet of k-mers, where k can equal 1. The matching k-mers in two sequences are found by scanning one sequence and using the index of the other. Indexing the shorter of the two sequences is easier to perform on-line; however, if the index is constructed off-line on the longer sequence, the number of comparison operation is potentially much smaller. We present the analysis of this effect for different real data sequence lengths in the context of protein search.

Keywords: protein sequence homology, string index, SWORD, Hamming distance vector, diagonal counting

1 Introduction

Finding the extent of the homology between two protein sequences is possibly the most important computational task in bioinformatics and biological sciences. The baseline results are obtained with the alignment algorithms such as Smith-Waterman (SW)[11] or Needleman-Wunsch [7]. Unfortunatelly, quadratic time complexity of the alignment algorithms renders them almost unusable when a large amount of sequences is compared. Therefore, different heuristic methods have been proposed throughout the years, from the early FASTA [8], BLAST [1] and BLAT [6] to the more recent ones such as Rapsearch2 [14] and DIAMOND [3]. They all share a common principle, i.e. decomposition of the alignment problem into seed and alignment phases. The former searches for seeds, i.e. subsequences of length k (k-*mers*) which are shared between a given pair of sequences, while the latter phase couples found seeds into local alignments based on different criteria. The seed phase is often implemented by creating an index containing all k-mers of the larger set of sequences, either stored on the hard drive or created on the fly. Only exception is the DIAMOND algorithm which uses double indexing, i.e. indexes are created for both query and target sequences and seeds are obtained by linearly traversing both indexes at the same time [3].

⋆ Supported in part by the Croatian Science Foundation under projects IP-11-2013-9623 and IP-11-2013-9070.

⋆⋆ Supported in part by the Croatian Science Foundation under project UIP-11-2013-7353.

Strahil Ristov, Robert Vaser, Mile Šikić: Trade-offs in Query and Target Indexing for the Selection of Candidates in Protein Homology Searches, pp. 118–125.
Proceedings of PSC 2017, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-06193-0

A recently developed method SWORD [12] applies a different approach, i.e. in the search stage equal k-mers are found in order to obtain a measure of similarity for a pair of sequences which further enables running the optimal alignment algorithms on only a small subset containing the most similar sequences. The measure of similarity between two sequences involves counting the number of matches at the best overlap of the sequences. To facilitate this, SWORD employs an index that is built on the fly on the smaller sequence, or the set of sequences (the query set). In another recent paper, a similar and complementary method has been proposed for the efficient Hamming vector calculation [10]. The two methods are similar, as they are both based on finding the best diagonal, i.e. the position at which two sequences should overlap in order to have the highest number of matching symbols aligned, and complementary in the choice of which sequence is indexed. The index is constructed on query in [12], and on target in [10]. In this paper we investigate the potential of combining the SWORD method with the indexing of the target.

2 Counting hits on a diagonal in sequence pattern matching

The principle of counting matches on a diagonal (or *diagonal counting* for short) was used in computational biology applications as early as in 1983 [13]. The goal is to find the greatest number of matches, without insertion or deletions, between two sequences and the corresponding position(s), i.e. the relative position, one or more of them, between the two sequences where the number of matches is the largest. This information can further be used for the location of possible good alignments as in [13,8] or to score the whole sequences as in [12]. Besides in bioinformatics applications, the underlying principle of counting matches at a given offset from the beginning of the sequence has been used in general string matching problems such as k-mismatch problem in [2] and Hamming distace vector calculation in [10]. Incidentally, finding the number of matches on all diagonals amounts to finding the inverse of the Hamming distance vector for two sequences.

Diagonal counting consist of sliding one sequence over the other and at each position counting the number of aligned symbols that are equal. The alphabet of symbols can consist of single characters, or of *k-grams*, i.e. k consecutive characters. In the context of protein sequences k-grams are usually called k-mers.

The brute force procedure would be to slide one sequence over the other and iteratively test for matches. If m and n denote the lengths of the two sequences that are matched, the complexity of brute force approach is strictly quadratic $\Theta(mn)$. However, instead of comparing the symbols at all positions in two sequences, it is possible to use indexing of one sequence to reduce the number of comparisons, as there is no need to check the positions where no match exists. Indeed, the indexing has been combined with the diagonal counting from the earliest works in bioinformatics [13,4].

In the two recent articles by the authors the diagonal counting with indexes has been used to optimize search for protein homology [12], and for efficient calculation of Hamming distance vector for protein sequences [10]. A salient difference between methods described in [12] and [10] lies in the choice of the object of indexing. The method described in the first article uses index on shorter (query) sequence and constructs it on-line, while the method described in the second article employs the index on the longer (target) sequence which is constructed off-line. We shall describe both algorithms in some detail in the following section.

3 Query vs. target indexing

3.1 SWORD algorithm and query indexing

The SWORD [12] is an efficient algorithm for protein database search that aims to offer better speed vs. sensitivity ratio than BLAST, a gold standard in sequence homology search [1]. Same as BLAST, and as well as most other algorithms for finding the homology in biological sequences, SWORD uses the two stage approach: the first stage is reducing the whole database to a subset of probable candidates based on some heuristics, and the second stage is performing the full scale alignment using the standard SW [11] or, in some cases, the Needleman-Wunsch [7] method. However, while BLAST in the first phase searches for hits - short matching segments in sequences, and then expands these hits into local alignments, the SWORD method finds the best whole candidate sequences from the database of sequences, and performs the SW alignment on the whole sequences. The effect is that less work is invested in processing of all potential positions for a good local alignment at the expense of performing SW on longer inputs. The increased work on SW alignment is compensated by using fast parallel processing.

The first phase of determining the best candidate sequences is performed using the diagonal counting. For a query sequence and each protein sequence in the target database the algorithm calculates the highest value of any diagonal when matching the two sequences at every position. The fixed number of sequences with the highest score are then forwarded to the alignment phase, which is a fast parallel SW implementation based on SIMD (Single Instruction Multiple Data) instructions [12].

For the purpose of this work we shall consider only the first, heuristic, stage of SWORD method. The diagonal counting in SWORD is performed on the alphabet of k-mers, with the different values of k producing different speed vs. sensitivity ratio. Larger values lead to greater speed and lower sensitivity. To increase sensitivity, matches can include not only the identical k-mers but also those that are similar enough according to a given amino acid similarity matrix. In [12] k is proposed to be 3 with included similar k-mers for the best sensitivity, or 5, with exact matches only, for the greatest speed. The published version of the code can be found at https://github.com/rvaser/sword. A threshold for similarity T, when $k = 3$, is set to $T = 13$ using BLOSUM62 substitution matrix [5].

To perform fast counting of matches, SWORD uses index on the query sequence. The index is constructed as a perfect hash table that for each different k-mer returns the list of the corresponding positions in the sequence. In case of sensitive search, for each k-mer in the query, similar k-mers are generated and stored in the index.

The SWORD uses diagonal counting to optimize the choice of candidates in search for protein homology, and the value of the highest score has proven to be an adequate criterion for the choice of candidates. According to the results presented in [12], SWORD can be regarded as a viable alternative to BLAST, it is overall considerably faster while remaining comparatively sensitive to distant homologies. However, the actual times needed for the processing of complex inputs can be considerable. For instance, matching the complete E. Coli proteome to NCBI NR protein data base requires approximately 4 hours with SWORD (an order of magnitude less than with BLAST) which is a motive for further research on a possible speed up of the algorithm.

3.2 The potential advantage of target indexing

The indexing of target strings in the context of diagonal counting has been proposed in [10] with the application in the Hamming vector calculation where query is, as a rule, much shorter than the target. The complexity of the matching in the average case is then given with:

$$O(mn \sum_{a \in A} p_a^2) \tag{1}$$

where A is the alphabet, and p_a is the probability of a symbol $a \in A$ in the target sequence. In the case when all symbol probabilities are equal (1) reduces to:

$$O(mn/|A|) \tag{2}$$

The effectiveness of this approach is based on the sparse distribution of symbols in shorter sequences when the alphabet is large. Solely the symbols that are present in query are accessed in the index. As long as the length of query remains small compared to $|A|$, indexing the target can considerably reduce the total number of operations. Only when all symbols from the alphabet are present in the query, the number of index access operations is the same as when the index is constructed on the query. The downside of target indexing is that it has to be done off-line since target is, as a rule, much larger than query.

Let us consider an alphabet of k-mers formed from 20 different amino acids. At the first sight it would appear that with the sizes of the alphabet that equal 20^k there is a great probability that a significant number of symbols will not be present in the shorter query sequence. However, with protein sequences, and when using the SWORD method that generates similar k-grams at each position, the number of different symbols in the query quite rapidly reaches $|A|$.

3.3 Generalized procedure for counting matches on a diagonal

Generalized statement of the best diagonal problem is: Finding the position(s) of the highest scoring match between two sequences under Hamming distance metrics on the alphabets of $k-mers$. If $k = 1$, the problem reduces to a simple inverse Hamming distance vector calculation. In order to avoid quadratic matching of every position in both sequences, the simple solution is to index one sequence and scan the other. Since the mechanism is the same regardless of which sequence is indexed, we give a general pseudo code in Algorithm 1 where query and target can be freely interchanged. The notation used in Algorithm 1 is as follows: S_{idx} and S_{scan} denote the indexed and the scanned sequence, respectively; the respective lengths of the sequences are denoted with $|S_{idx}|$ and $|S_{scan}|$; $|A|$ is the size of alphabet A, where the symbols in the alphabet can be single characters or k-mers; $PositionList[a]$ stores positions of all instances of a in S_{idx}, for all symbols $a \in A$; $MatchVector$ stores the scores for $|S_{idx}| + |S_{scan}| - 1$ diagonals in an alignment array between the two sequences.

After the construction of $MatchVector$ it is easy to find, in one pass, the value and the position(s) of the best match(es). Index is an inverted file of indexed sequence and its construction is linear with $|S_{idx}|$. If the scanned sequence includes symbols that are not present in the indexed sequence, there will be unnecessary index accesses. In the next section we give experimental results on how long can a query protein sequence be to justify the target indexing approach.

Algorithm 1: *PositionList* and $MatchVector$ are initialized to zeros; $S_{idx}[i]$ and $S_{scan}[i]$ denote character at position i in S_{idx} and S_{scan}, respectively

```
1 int* PositionList[|A|];
2 int MatchVector[|S_idx| + |S_scan| − 1];
  /* pre-processing phase                                   */
3 for i = 0 to |S_idx|-1 do
4     add i to PositionList[S_idx[i]];
  /* Match vector computation                               */
5 for i = 0 to |S_scan|-1 do
6     for j in PositionList[S_scan[i]] do
7         MatchVector[j - i + |S_scan| − 1] ++;
```

4 Experimental results

Off-line indexing of a longer target sequence is justified when alphabet symbols are sparse in the query sequence. In such cases the reduced number of index access operations compensates for longer processing time needed for the index construction. The size of the fraction of the target alphabet that is present in the query depends on the size of the alphabet and the query length. Obviously, as the length of the query increases, more and more symbols are present. Let $pA_{t/q}$ denote the percentage of target alphabet symbols present in query sequence. In order to asses the possible speedup of SWORD algorithm that could be achieved with target indexing we have performed experimental analysis to obtain insight on what are the real values of $pA_{t/q}$, with different query lengths, on the alphabet of amino acids and real protein sequence data. We have employed the heuristic part of SWORD algorithm that scans the target and looks up the found k-mers in the index of query sequence. We counted the number of target k-mers that were found in the index and calculated the $pA_{t/q}$ percentage. The fraction of unsuccessful index accesses is the direct measure of the gain (i.e. reduction of the number of index accesses) that can be achieved using target indexing. In all our experiments all k-mers present in query were also present in the target, therefore with target indexing every index access would be successful.

The targets were different size subsets of the UniProt database [9], and the queries were random protein sequences of three different sizes, taken from the target. In Tables 1-6 we report the results for query lengths of 100, 500 and 1500 amino acids, and target data sets that are comprised of 10^4, 10^5 and $5{\times}10^5$ lines, where each target line is a different protein from UniProt database. The sizes of the targets are $3.7{\times}10^6$, $37.5{\times}10^6$, and $181{\times}10^6$ amino acids, respectively. Tables are organized according to the value of k, and present results for the exact k-mer matching as well as the matching with the expanded number of k-mers that include all similar k-mers within the given similarity threshold. The values in column *tested* represent the number of k-grams present in the target that are tested against the index of a query. These values are roughly equal to the sizes of the target sets. The values in column *matches* are the numbers of k-mers found in the query index. The values of $pA_{t/q}$, i.e. the percentage of tested k-mers that are actually matched are given under % sign.

Following the findings in the SWORD paper we have experimented with $k = 3$ and $k = 5$. The results for $k = 3$ are presented in Table 1 for the exact matching and in Table 2 for the expanded matching with similarity threshold used in SWORD algorithm. The results for exact matching with $k = 5$ are presented in Table 3. The

Table 1. k = 3, exact matching

	10^4 lines			10^5 lines			5×10^5 lines		
query length	tested	matches	%	tested	matches	%	tested	matches	%
100	3751688	95199	2.5	37557230	1008318	2.7	181015261	5013396	2.8
500	""	508787	13.6	""	5313058	14.1	""	26098594	14.4
1500	""	1272292	33.9	""	13257314	35.3	""	64529580	35.6

Table 2. k = 3, expanded matching with similar k-mers (BLOSUM62 similarity ≥ 13)

	10^4 lines			10^5 lines			5×10^5 lines		
query length	tested	matches	%	tested	matches	%	tested	matches	%
100	3751688	194925	5.2	37557230	1980520	5.3	181015261	9724152	5.4
500	""	895241	23.9	""	9044466	24.1	""	44277325	24.5
1500	""	3293569	87.8	""	37331149	99.4	""	159200790	87.9

Table 3. k = 5, exact matching

	10^4 lines			10^5 lines			5×10^5 lines		
query length	tested	matches	%	tested	matches	%	tested	matches	%
100	3731688	550	0.0	37357230	5259	0.0	180015261	27047	0.0
500	""	3113	0.1	""	24729	0.0	""	128832	0.1
1500	""	6883	0.2	""	59175	0.2	""	269872	0.1

Table 4. k = 5, expanded matching with similar k-mers (BLOSUM62 similarity ≥ 20)

	10^4 lines			10^5 lines			5×10^5 lines		
query length	tested	matches	%	tested	matches	%	tested	matches	%
100	3731688	7759	0.2	37357230	80798	0.2	180015261	394591	0.2
500	""	35800	1.0	""	357282	1.0	""	1789459	1.0
1500	""	181056	4.9	""	1788243	4.8	""	8616060	4.8

Table 5. k = 5, expanded matching with similar k-mers (BLOSUM62 similarity ≥ 22)

	10^4 lines			10^5 lines			5×10^5 lines		
query length	tested	matches	%	tested	matches	%	tested	matches	%
100	3731688	2516	0.1	37357230	24954	0.1	180015261	123087	0.1
500	""	9644	0.3	""	93423	0.25	""	469165	0.3
1500	""	54624	1.5	""	544963	1.5	""	2584125	1.4

Table 6. k = 5, expanded matching with similar k-mers (BLOSUM62 similarity ≥ 24)

	10^4 lines			10^5 lines			5×10^5 lines		
query length	tested	matches	%	tested	matches	%	tested	matches	%
100	3731688	951	0.0	37357230	9830	0.0	180015261	49270	0.0
500	""	4154	0.1	""	36207	0.1	""	189052	0.1
1500	""	20002	0.5	""	178632	0.5	""	854227	0.5

results in Tables 1-3 cover all indexing variants used in SWORD. Expanded matching with $k = 5$ would incur considerable processing overhead for on-line construction of query index. Especially so if the similarity threshold is set for higher sensitivity. However, if we accept the necessity of building the index off-line, we are not constrained to expanding only 3-mers. Off-line index can be constructed for any k and threshold that may result in good speed vs. sensitivity ratio. As an example, in Tables 4-6 we

have included results for the expanded matching with $k = 5$ and similarity thresholds of 20, 22, and 24, respectively.

The results show that with 3-mers the query alphabet comes close to saturation with the expanded matching when the length of the query is 1500 amino acids (Table 2). In such cases indexing of the target would not be economical. On the other hand, with the exact matching on 3-mers (Table 1), and both the exact and expanded matchings on 5-mers (Tables 3-6) the results are much more promising. Using target index in those cases can considerably reduce the number of index accesses. We find of particular interest the results presented in Table 4. Even with the expanded 5-mers, and a low threshold of similarity, indexing the target can reduce the number of index accesses by the factor of 20 with 1500 amino acids long query.

4.1 Discussion of the results

The heuristic phase of SWORD consumes approximately half of the total processing time. In this paper we present the preliminary findings on which we will base further investigation of possible speedup. To exploit the benefits of target indexing the query must be short. In the current version of SWORD algorithm multiple queries are combined in one query index. This could possibly be modified in a way to reduce the query alphabet saturation. Therefore, the potential for the improvement exists but it has to be investigated further.

The results obtained on 5-mers open the possibility of further investigation to establish the level of sensitivity with expanded k-mers with larger k and different thresholds. The speed of SWORD algorithm is equally the result of a careful implementation regarding the cache efficiency. Indexing expanded k-mers increases the index size by an order of magnitude i.e., 7 to 13 times in our experiments. This rises requirements on the design of the data structures for index storage and access. Obviously, cache friendly, succinct and localized data structures should be employed for storing the index.

5 Conclusions

We have compared two complementary methods of indexing for finding the best match between two sequences under Hamming distance: one where the index is constructed on a query sequence and scan is performed on a target, and one where the index is constructed on a target sequence and scan is performed on a query. Both approaches reduce the number of comparisons with respect to the brute force approach and produce the same final result. A query is, as a rule, much shorter than the target, and the advantages of query indexing are the possibility to perform it on-line and the low space requirements for the index. On the other hand, if the target is indexed off-line, the number of index accesses is restricted to the length of the query, which can significantly reduce the total number of matching operations. This effect is dependent on the length of the query and the percentage $pA_{t/q}$ of alphabet symbols represented in the query. We have performed the experiments to obtain the insight into actual values of the query length and $pA_{t/q}$ within the framework of the heuristic part of SWORD algorithm for protein search. Somewhat surprisingly, we have found out that in the core SWORD variant, when amino acid triplets are expanded with similar 3-mers, almost all of the alphabet is present in a query of length 1500. As a result, target indexing cannot be straightforwardly implemented. On the other hand, the

results on the longer k-mers show much more promise. It is apparent that using longer k-mers with larger similarity neighborhood can lead to strong reduction in the number of total matching operations. However, to explore this fact, further work is required in finding the appropriate cache efficient data structures for storage of the larger index, as well as the modification of SWORD algorithm in order to work with shorter queries.

The actual possible speedup of SWORD method will probably have more to do with cache efficiency related data handling than with string algorithms. Nevertheless, the underlying mechanism could very well be based on the findings presented in this investigation. To our knowledge, this is the first analysis of that kind and we hope it may be useful to designers of algorithms based on protein sequence indexing.

References

1. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman: *Basic local alignment search tool.* Journal of Molecular Biology, 215(3) 1990, pp. 403 – 410.
2. A. Amir, L. M., and P. E.: *Faster algorithms for string matching with k mismatches.* Journal of Algorithms, 50(2) 2004, pp. 257 – 275, Soda 2000 Special Issue.
3. B. Buchfink, C. Xie, and D. H. Huson: *Fast and sensitive protein alignment using DIAMOND.* Nature Methods, 12(1) 2015, pp. 59–60.
4. J. P. Dumas and J. Ninio: *Efficient algorithms for folding and comparing nucleic acid sequences.* Nucleic Acids Research, 10(1) 1982, pp. 197–206.
5. S. Henikoff and J. Henikoff: *Amino acid substitution matrices from protein blocks.* Proceedings of the National Academy of Sciences of the United States of America (PNAS), 89 1992, pp. 10915–10919.
6. W. J. Kent: *BLAT - The BLAST-Like Alignment Tool.* Genome Research, 12(1) 2002, pp. 656–664.
7. S. B. Needleman and C. D. Wunsch: *A general method applicable to the search for similarities in the amino acid sequence of two proteins.* Journal of Molecular Biology, 48(3) 1970, pp. 443 – 453.
8. W. R. Pearson and D. J. Lipman: *Improved tools for biological sequence comparison.* Proceedings of the National Academy of Sciences of the United States of America, 85(8) 1988, pp. 2444–2448.
9. The UniProt Consortium: *Uniprot: a hub for protein information.* Nucleic Acids Research, 43(D1) 2015, pp. D204–D212.
10. S. Ristov: *A fast and simple pattern matching with hamming distance on large alphabets.* Journal of Computational Biology, 23(11) 2016, pp. 874–876.
11. T. Smith and M. Waterman: *Identification of common molecular subsequences.* Journal of Molecular Biology, 147(1) 1981, pp. 195 – 197.
12. R. Vaser, D. Pavlovic, and M. Sikic: *SWORD - a highly efficient protein database search.* Bioinformatics, 32(17) 2016, pp. 680–684.
13. W. Wilbur and D. Lipman: *Rapid similarity searches of nucleic-acid and protein data banks.* Proceedings of the National Academy of Sciences of the United States of America - Biological Sciences, 80(3) 1983, pp. 726–730.
14. Y. Zhao, H. Tang, and Y. Ye: *RAPSearch2: a fast and memory-efficient protein similarity search tool for next-generation sequencing data.* Bioinformatics, 28(1) 2012, pp. 125–126.