Improved alignment of protein sequences based on common parts

David Hoksza

Department of software engineering, Faculty of Mathematics and Physics,
Charles University in Prague
Malostranské nám. 25, 118 00, Prague 1, Czech Republic
david.hoksza@mff.cuni.cz

Abstract. In the last twenty years, protein databases have been growing exponentially. To speed up the search, heuristic approaches have been proposed and their accuracy has been steadily growing, but exact search is still needed in some cases. The only exact search algorithm remains SSEARCH (or it's clones) which sequentially scans database of protein sequences, and performs full alignment against each of the sequences.
Due to the need of the exact search, we focus on improving the sequential search algorithm. We decrease the costs needed to compute the alignment of pair of protein sequences when used with large databases. This is achieved by reusing alignment calculations of common parts of the sequences without loss of accuracy.
With this method, we reduced the computational costs by up to 20 % depending on the database size and subset used. We also implemented approximate search which further reduced computational costs for the the sake of some accuracy loss.

Key words: protein databases, Smith-Waterman algorithm

1 Introduction

In recent years, there has been an exponential growth of databases of protein sequences. One of the reasons for this growth is the fact that similarity between a protein sequence with an unknown function and a sequences in a database of protein sequences with known functions is used by biologists to help to determine the function of the inspected protein. Also protein sequences of different species can help (if they are sufficiently similar) in getting information about the new protein, and therefore it makes sense to keep at disposal as large repositories of sequences as possible, hence size of the repositories exponentially grows. This growth has an obvious consequence - searching the repositories becomes slower, especially when *Smith-Waterman* (SW) algorithm [16] (which is the most sensitive method for homology search) is used as the similarity measure. The *Smith-Waterman* algorithm is of quadratical complexity, and therefore use of this algorithm became unacceptable for large repositories. To decrease quadratic complexity of the search to linear, heuristic algorithms were proposed - most well known are *FASTA* [10] and *BLAST* [1][1].

Similarity of two sequences, when measured with Smith-Waterman algorithm, is called Smith-Waterman (SW) score. SW-score is applied only to a single pair of sequences and thus E-value (expected value computed out of SW-score) was proposed to incorporate some statistics into the output such as lengths of the aligned sequences, size of the database, etc. E-value for a query sequence q, a database sequence and a SW-score S expresses number of database sequences that will score with q equivalently or better than S with respect to the database size. Hence, lower E-value leads to more significant matches. In usual scenario user inputs an E-value and a protein sequence and gets on the output sequences that are similar to the given sequence with E-value lower than the given value[2]. Due to the rigorous nature of the algorithm, Smith-Waterman finds more distant matches than BLAST or FASTA does, thus SW can output also sequences that would have sufficient E-value but BLAST of FASTA would miss it. And there are still situations where accuracy is preferred over speed, e.g. database curation, finding structures by sequence alignment, etc. In these cases, rigorous algorithms are used.

Smith-Waterman algorithm is nowadays implemented in *ScanPS* [2] (original SW algorithm), *SSEARCH* (SW with SWAT optimizations [6]), or *MPsrch* (*MPsrch* is parallelized version of the true Smith and Waterman[3]).

2 Similarity in Sequence Databases

Since protein sequence is a linear sequence of letters upon alphabet of 20 amino-acids[4], algorithms and similarity measures for searching protein databases are similar to algorithms for searching databases of ordinary strings with the difference that the similarity measure used is a bit more complex (because of semantics of protein sequences). As for every database of objects we must foremost define a similarity measure which we will use to compare objects, and based on this measure we can define methods for searching it. In string databases, similarity measure is based on alignment of sequences (usually pairs of sequences[5]). Alignment of sequences is such an arrangement of it's letters (by inserting gaps to each of them) which holds order of the letters. Hence the simplest alignment is defined upon sequences of equal lengths (pairing letters at equal positions). Score of the alignment is number of positions where the sequences differ. Such a scoring system is called *Hamming distance* (Fig. 1a).

[1] Nowadays BLAST and it's clones are the most exploited heuristic algorithms for homology search among protein sequences.

[2] Similar sequences have low E-value but high SW-score.

[3] MPsrch, SSEARCH and ScanPS are operated by EBI and currently handle about 1100 jobs per month as stated by EBI support.

[4] Each of the amino-acids is coded by a triplet of nucleotides from DNA called codons.

[5] Techniques of multiple alignments of sequences are also widely investigated but it's not subject of this paper.

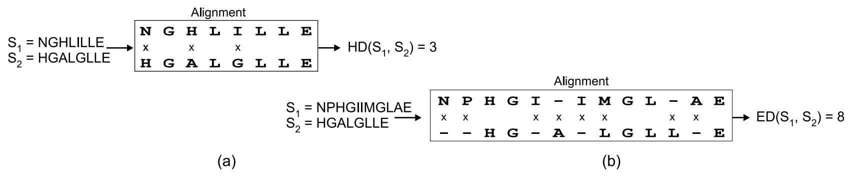

Fig. 1. Hamming (a) and Edit (b) distance

To allow to compare sequences of different lengths we need to insert gaps to them. The desired alignment minimizes number of gaps and positions at which the sequences differ (score of the alignment). Equivalently we can compute minimal number of editing operations needed to turn one sequence into the other where editing operations are inserting, deleting and modifying a single letter. Therefore the measure is called *edit* or *Levenshtein distance* (Fig. 1b).

Edit distance penalizes each of the editing operations with the same cost. *Weighted edit distance* is an extension to the edit distance which distinguishes editing operations by assigning different (but constant) costs to each of them.

For many applications weighted edit distance is sufficient, not so for protein sequence alignment where we need to incorporate a *weighting system* which assigns a specific cost to each pair of amino-acids which is used when a pair of amino-acids is aligned. The need arises from the fact that protein sequences comprise a kind of semantics which stems from the evolutionary history of protein sequences. Knowing evolution history of many sequences allows us statistically derive probability that an amino-acid will mutate into another one in a given time period. Based on this probability, we are able to assign score to each pair of amino-acids. In bioinformatics, we present this fact by $20x20$ *substitution matrix*. There are many different sets of matrices[6] nowadays and the most wide spread ones are PAM [3] and BLOSUM [7].

Final modification, specific to protein sequences alignment, is different costs for opening and extending a gap. This means that in the resulting alignment a position where a gap starts is penalized (usually noticeably) more then the consequent gap positions.

Moreover to these scoring systems, we differentiate between *local* and *global* alignment. The standard conception of aligning sequences is called global alignment. But if we are interested in finding substrings of pairs of sequences that express high level of similarity, we use local alignment. Thus the best local alignment is an alignment of such a pair of substrings that has the highest global alignment score among all possible pairs of substrings.

2.1 Dynamic Programming

There are quadratic complexity algorithms based on dynamic programming for computing both global and local alignments.

[6] Depending whether we are searching for evolutionary close or distant homologues we use different matrices of these sets.

Global Alignment Dynamic programming algorithm for global alignment of two sequences was published in 1970 by Needleman and Wunsch [12]. It is based on dynamic programing matrix G of size $(m+1) \times (n+1)$ where m and n are lengths of the sequences to be aligned. $G[i,j]$ stores score of the optimal alignment of prefixes of respective lengths (zero-th row and column are designated for initialization). The recursive formula for computing cells of the matrix is:

$$G[i,j] = max \begin{cases} G[i-1,j] + \sigma \\ G[i,j-1] + \sigma \\ G[i-1,j-1] + S[a_i, b_j] \end{cases} \tag{1}$$

In this formula σ stays for gap cost, $i \in \{1..m\}, j \in \{1..n\}$, a and b are sequences to be aligned and S is a substitution matrix. Ergo, at position $[i,j]$ we can align i-th and j-th letter or add a gap to one of the sequences. In the initialization phase, we fill the border cells with $G[i,0] = i * \sigma$, $G[0,j] = j * \sigma$, because alignment of a string of zero length with a string of length i demands inserting i spaces into the empty string.

From the fact that $G[i,j]$ contains score of the resulting optimal alignment of respective prefixes follows that score of the optimal global alignment can be fetched from $G[m,n]$. In this paper, the score is what matters to us but if we were interested in the alignment itself, we could backtrack the matrix in the same way it was filled according to the formula 1 (first line means adding space into 'vertical' sequence, second adding space into 'horizontal' sequence and third means aligning i-th and j-th letters of the sequences).

The original formula 1 lacks distinction for different costs for opening and extending a gap. In order to express it, we must incorporate two matrices for vertical and horizontal gaps [5]. Role of these matrices is to decide whether it makes sense to start a new gap, or it would be better to continue an already started gap[7]:

$$H[i,j] = max \begin{cases} G[i,j-1] + \sigma \\ H[i,j-1] + \delta \end{cases} \tag{2}$$

$$V[i,j] = max \begin{cases} G[i-1,j] + \sigma \\ V[i-1,j] + \delta \end{cases}, \tag{3}$$

where δ stays for the cost of continuing a gap. And finally, we also need to change the recursion for G to incorporate the H and V matrices:

$$G[i,j] = max \begin{cases} V[i,j] \\ H[i,j] \\ G[i-1,j-1] + S[a_i, b_j] \end{cases}. \tag{4}$$

[7] We use matrices here but (as follows from the recursion) one-dimensional arrays would be sufficient if we would align the sequences line by line or column by column (for that matter, this is also true for the G matrix where two one-dimensional array would be sufficient).

Local Alignment As mentioned earlier, in bioinformatics we are usually more interested in searching common or highly conserved sections of protein sequences. For this purpose, global alignment is not optimal since relatively short but highly similar parts can be 'lost' in order to properly align rest of the sequence. In 1981, Smith and Waterman [16] published dynamic programming algorithm solving local alignment problem. This algorithm is actually a slight modification of Needleman and Wunsch. If at any position score of the global alignment algorithm is positive, the optimal local alignment will certainly not start at that position because the already aligned parts score above zero. But if at any position the score should be negative, it makes sense to start a new local alignment from that position because the final score will be higher by the absolute value of the particular score. Let's add just stated decision into the recursive equation:

$$L[i,j] = max \begin{cases} V[i,j] \\ H[i,j] \\ L[i-1,j-1] + S[a_i,b_j] \\ 0 \end{cases} .^{8} \quad (5)$$

To get the best local alignment we also need to stop aligning at the position with the highest score. Therefore, optimal local alignment score is not at position $L[m,n]$ but at position $[i_{max}, j_{max}]$ with the highest $L[i_{max}, j_{max}]$ value. If we start backtracking from the position $[i_{max}, j_{max}]$ to position $[i_0, j_0]$ where $L[i_0, j_0] = 0$, we get route of the optimal local alignment.

3 Speed-up by Using Common Parts

As mentioned earlier, there exist areas where rigorous alignment by Smith-Waterman algorithm is needed. Here, we can not sake accuracy for speed and thus the remaining ways how to speed up the search are indexing, parallelism, or speeding up the distance computation itself.

Following the knowledge of distances among objects which are precomputed and stored, the indexing methods filter out as many objects as possible without even fetching them from the database. Methods with primary target to decrease number of distance computations are called metric access methods (MAM) because of incorporating axioms of metric to filter out groups of objects. Nevertheless, MAM's are not applicable to protein alignment problem since it is not metric at all, and it is difficult to turn it into metric, as has been show in [8]. There have also been attempts to turn the distance into metric [18] by modifying the substitution matrix which is based on observation done in [15]. However, this method is applicable only to global alignment and just to q-grams (not the whole sequences). Other approaches have similar drawbacks and up to date there is no indexing method that could replace BLAST (as far as we know).

There are two possible approaches how to parallelize the database search with Smith-Waterman. First one is to parallelize the whole database search by running

[8] This method is applied to initialization phase too, so the border cells are filled with zero values.

multiple alignments at the same time on more than one computational unit. The speed-up is directly proportional to the number of the computational units involved (example of this approach is MPSrch [11][9]). Second approach demands hardware modifications and focuses on speeding up single distance computation (computing the dynamic programming matrix). Hardware platform enabling this solution is FPGA (Field Programmable Gate Array) [13] [4] or standard CPU enabling certain degree of parallelism [14].

As far as we know, there have not been many attempts to improve the alignment of protein sequences itself without a specialized hardware. However, interesting improvement was achieved in [9].

3.1 Basic Algorithm

Method presented in this paper saves computation costs as indexing methods do, but instead of omitting distance computations it reuses parts of the distance matrix computed by Smith and Waterman. This decreases number of operations needed for virtually every alignment done during the whole database search.

So, the idea is to store parts of the matrix and use it later. But almost every submatrix in the dynamic programming matrix is context dependent - its content is dependent not only on the amino-acids to be aligned, but also on the calculation made so far. This makes it impossible to store any inner submatrix for later use unless content of the border cells are the same in a future alignment. The only set of submatrices having the same left and top context are submatrices starting at position [0,0]. While the query sequence stays the same in the search, if two sequences share a common prefix, their Smith-Waterman matrix will be identical up to the point where they start to differ. Let's imagine the query sequence to be at the top side of the matrix and the database sequence at the left side then if two sequences s_1, s_2 share common prefix of length n then the main idea is as follows:

1. Align s_1 with the query sequence (fill the dynamic programming matrix).
2. Replace s_1 in the matrix by s_2.
3. Start Smith-Waterman with s_2 from the $(n+1)$-th row.

Of course, we need to repeat the same technique with the H and V matrices to make the whole thing work[10].

Notice that we do not need to change the existing algorithm at all and we even do not need a persistent storage for the submatrices corresponding to the common prefixes. All we need is to sort the database according to the prefixes. Then we can traverse the database in prefix order, and if two consequent sequences share a common prefix, we save portion of distance computations proportional to the common prefix length. When traversing this way, the method has no additional memory demands at all.

[9] Since MPsrch is a commercial product, its exact algorithm is not known.

[10] If we store just one-dimensional arrays instead of matrices L, H and V, the algorithm still works fine we just need to store the respective one-dimensional arrays (located at the position where sequences start to differ) and use it in the next step.

We define so called *prefix ratio* which is the proportion between overall length of the prefixes (i.e. if we sort the database according to the prefixes, then each sequence contributes to the overall length with length of the shared prefix with the previous sequence) and the length of the database (sum of lengths of individual sequences). The speed-up is then equivalent to the prefix ratio of the database being searched, and is independent on the query sequence.

3.2 Improvement by Using Inversed Sequences

Further speed-up might be achieved if we would find another parts that are common to some set of sequences and are context independent or the context is the same among them. We realized that the score of the optimal alignment is independent on the direction of the alignment, hence if we denote $r(s)$ inversion of sequence s, then score of the optimal alignment of s_1 and s_2 is equivalent to the score of the optimal alignment of $r(s_1)$ and $r(s_2)$:

Theorem 1. *Let's denote an optimal alignment of strings* $\underline{s_1}$ *and* $\underline{s_2}$ *as* $\underline{a(s_1, s_2)}$, *score of the alignment as* $\underline{opt} = \underline{s(a(s_1, s_2))}$, $\underline{r(s)}$ *the reverse string of* $\underline{s}$. *Then* $\underline{opt} = \underline{s(a(s_1, s_2))} = \underline{s(a(r(s_1), r(s_2)))}$.

Proof. First, we show that the score of an arbitrary alignment does not depend on the direction in which it is computed. At each position of an alignment, two letters are aligned or there is a space in one of the sequences. In the first case, since aligning of two letters is not context dependent, the direction does not matter. In the latter case, each position inside a longer gap is scored equally independently of the direction. For border positions of the gaps, the first position in the direction of the alignment is scored differently than the last one, but sum of both is again equal independently of the direction.

Hence, score of an alignment is independent of the direction. Let opt_r be the score of the optimal alignment of the reversed sequences. If $opt < opt_r$, than $s(a(s_1, s_2)) = opt < opt_r = s(a(r(s_1)), r(s_2))$ which is in conflict with the proved fact that $s(a(s_1, s_2)) = s(a(r(s_1), r(s_2)))$. The same is true for $opt > opt_r$. Hence, $opt = opt_r$.

We can use the knowledge of the score of the alignment of reversed strings to improve the presented method. For each sequence in the database we can decide whether to align it with the query sequence in the standard way or whether to reverse both, the query and database sequence and do the alignment. If we *appropriately* divide the database into two groups (sequences to be aligned in the standard way and sequences to be aligned reversely) we might increase the prefix ratio and thus speed-up the whole search.

Partitioning of the database can be done in two stages:

1. Divide a given percent of the database into 2 groups randomly, and then add each of the remaining sequences into a group so that the overall prefix ratio increases.
2. Repeat following step n-times - move a random sequence from one group into the other one if it would increase the overall prefix ratio.

3.3 Inexact Search

Method presented in this paper is dependent on the amount of the prefix ratio of the database, so one of the goals is to increase this ratio. With growing size of the database, the prefix ratio should increase because the probability of sequences having common prefixes increases too. By chopping sequences into more parts, we get bigger database with shorter sequences, hence the prefix ratio should grow. But this method brings a serious drawback - if an optimal alignment of an unbroken sequence will be spanned over the point of split then the sequence might not occur in the result set any more[11].

4 Experimental Results

In our experiments, we focused on how various parameters and methods influence prefix ratio (PR). Remember that PR corresponds to percentage of the speed increase according to the full scan. Experiments were performed on subset of UniProt database [17] with restricted lengths of the sequences to 3000 letters which makes 99.9% (5,340,227 sequences) of the whole database.

In all the experiments where subset of database were used, five independent random subsets of given size were generated and experiments where carried out against each of them. The results were averaged in order to avoid random subsets with higher PR than the average.

4.1 Prefix Ratio

For the first experiment we generated subsets of size 1000, 5000, 10000, 15000, 30000, 50000, 80000, 100000, 200000, 50000 and 1000000. These datasets were used to find out how size influences PR. We expected that when dealing with bigger number of sequences, the probability of the sequences having more common prefixes increases. In Tab. 1, we can see that this assumption is correct, e.g. subset of size 1,000,000 has PR 9.1 whilst subset of size 1000 only 0.9. To receive the highest achievable prefix ratio (according to todays database size), we carried out the same test also against the whole UniProt which gave us 18% speed-up. This speed-up should grow (not unlimitedly) with increasing size of the protein databases.

We also performed PR tests against few semantic based subsets which might lead to higher prefix ratio since sequences from similar organisms might show higher similarity in general. For these test we used subsets of UniProt based on taxonomic divisions[12]. Results, supporting the assumption of higher prefix ratio of semantically closed datasets, can be seen in Tab. 2. Majority of the sets have noticeably higher prefix ratio than random sets of comparable size from Tab. 1 (especially bacteria and viruses datasets).

[11] Sequence appears in the result set if one of it's parts align with the query with score higher than the threshold.

[12] Can be downloaded from EBI FTP.

Table 1. PR (in %) of random subsets.

Subset Size	*Prefix ratio*
1000	0.9
5000	1.4
10000	1.7
15000	1.9
30000	2.4
50000	2.8
80000	3.2
100000	3.6
200000	4.6
500000	6.8
1000000	9.1
5340227	18

Table 2. PR (in %) based on taxonomic divisions.

Taxonomic Division	*Count*	*Prefix ratio*
archaea	11489	2.4
bacteria	140132	18.7
fungi	19520	2.1
human	17599	1.1
invertebrates	14001	3.8
mammals	16807	8.8
plants	23718	8.7
rodents	22280	4.2
vertebrates	12213	4.4
viruses	11525	9.2

4.2 Reversed Sequences

In order to see how we can improve PR by using reversed sequences as explained in section 3.2, we performed experiments on subsets up to the size of 200,000. First, experiments concerned the initial build of the two groups. In the building stage, a portion of the database is divided into the groups randomly, and the rest is added in the way that would increase the overall PR. Hence we focused on finding the appropriate percentage of the database that should be inserted randomly. In Fig. 2a, we can see that there is no given percentage that would be optimal for all cases. However, in most cases to insert about 50% of the database randomly works just fine. In absolute numbers, after the building stage the PR is worse than the PR with basic algorithm.

In the next experiment, we were shifting sequences randomly between the two groups to improve PR. Results in Fig. 2b clearly show that the ratio increases just up to a particular value for smaller datasets. We believe that the absolute possible prefix ratio was not reached at this point, but the distribution reached a local optimum. What lead us to this assumption? We limited size of each group [13] because sequences tend to be cumulated in one of the groups (the bigger one). In the point where the optimum was achieved, sizes of the groups were limitary which might be an indication of a local optimum. Even though, we managed to increase PR with this method about 20% according to the basic method. For convenience, in Tab. 3 you can compare PR after building stage, PR achieved by shifting sequence between groups [14] and finally, PR achieved without using reversed sequences.

[13] The size of group A can be at most 2/3 of size of B.

[14] These numbers slightly differ from numbers in Fig 2b because here we performed more shifts (to reach the local optimum).

Table 3. Prefix ratio growth by using reversed sequences.

subset	*prefix ratio*			*subset*	*prefix ratio*		
	build	*shifts*	*no reverse*		*build*	*shifts*	*no reverse*
1000	0.6	1	0.9	50000	2	3.2	2.8
5000	1	1.8	1.4	80000	2.4	3.4	3.2
10000	1.2	2.1	1.7	100000	2.6	4.8	3.6
15000	1.4	2.4	1.9	200000	3.4	6.2	4.6
30000	1.7	3	2.4				

4.3 Splitting

Last experiments investigated impact of splitting on PR and accuracy. In all the experiments, 5,000,000 alignments were performed and the results were averaged. Tab. 4 [15] presents first part of the experiment - for different E-values [16] shows number of sequences that pass the cutoff score, average lengths of these sequences, their average SW score, average score that they need to pass, and number of cells in the dynamic programming matrix that reach the maximal value (hence number of possible optimal alignments). Next rows in the table show number of sequences that pass the cutoff score and (at the same time) span over given number of splits [17]. Inspecting number of sequences spanning over a split might lead us to a conviction that also real inaccuracy will show similarly poor numbers (inaccuracy about 90%) which was not confirmed, as Tab. 5 certifies. The accuracy loss (number of sequences that would normally score above a given threshold but do not because of the split) is about 17%. SW score and cutoff score from Tab. 4, together with number of maximal values in one alignment, are responsible for this contradiction. When an optimal

[15] Experiments in Tab. 4 and Tab. 5 were performed against the whole UniProt database.

[16] In practice usually E-value 10 is used.

[17] Splitting occurs equally along the sequence.

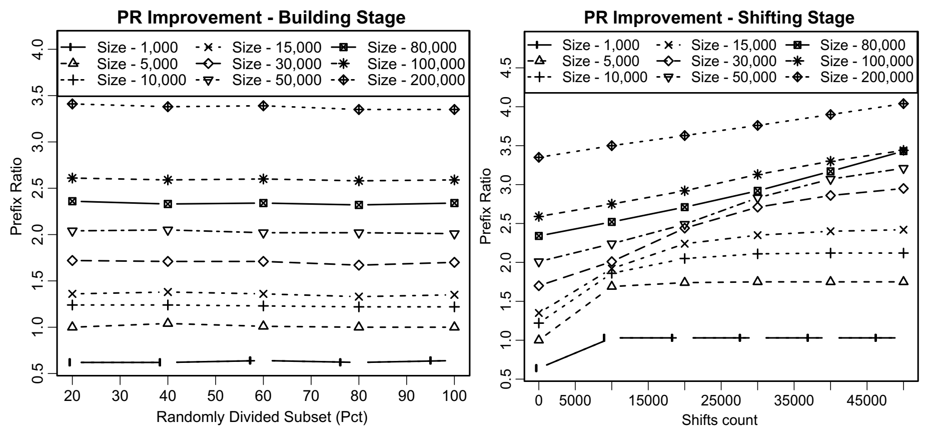

Fig. 2. Prefix ratio - a) after building stage according to different sizes being divided randomly, b) in shifting stage according to increasing number of shifts.

alignment is split, it has such a high score that even its parts may score above the cutoff score alone. Moreover, in some cases there are more than one optimal alignment and thus one of them might not be split at all.

Table 4. Alignment statistics.

	E-value											
	6		*8*		*10*		*12*		*14*		*max*	
result size	7252		7455		7607		7685		7771		7906	
∅ length	171		171		172		173		174		25	
∅ SW score	384		389		396		402		411		31	
∅ cutoff score	77.5		76.5		75.6		75		74.2		0	
∅ max values	1.3		1.3		1.3		1.3		1.3		1.5	
	abs	pct	abs	pct	abs	pct	abs	pct	abs	pct	abs	pct
1 splits	5887	81.2	6001	80.5	6086	80	6173	79.4	6245	79	1159830	23.2
2 splits	6694	92.3	6860	92	6984	91.8	7112	91.5	7216	91.3	2072025	41.4
3 splits	6993	96.4	7172	96.2	7305	96	7450	95.9	7570	95.8	2700590	54
4 splits	7092	97.7	7278	97.6	7420	97.5	7572	97.4	7695	97.3	3204691	64

Table 5. Splitting accuracy loss.

	E-value				
	6	*8*	*10*	*12*	*14*
1 splits	14.3%	16.2%	15.8%	17.2%	17.2%
2 splits	13.4%	15.1%	15.8%	15.6%	15.9%
3 splits	17.1%	17.2%	17%	18%	18.2%
4 splits	15.9%	16.9%	17.1%	17.6%	18.1%

Interesting observation is that the accuracy is higher for three splits than it is for two. Such an observation has already been made and taken into account in computing the E-value - probability of an alignment in the middle of a sequence is higher than on its edges.

Finally, we investigated how size of the database influences PR when splitting. In Fig. 3, we can see that with increasing size of the database also the PR gain increases.

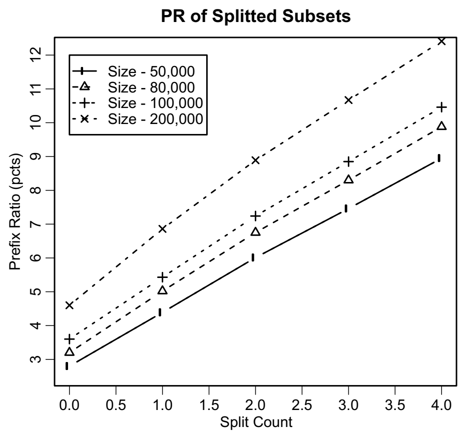

Fig. 3. Relation of number of splits and prefix ratio.

5 Conclusion

We implemented and tested a modification of Smith-Waterman algorithm for large datasets which benefits from shared prefixes and suffixes of the sequences. This modification can be incorporated into existing implementations, thus increasing speed for the price of just slight modifications. If used with methods aligning parallely more sequences in one moment, the speed-up might be lower since neighboring sequences (in the sense of prefix order) might be evaluated concurrently. Nevertheless, the speed-up might be up to 20% without any accuracy loss. When inacuraccy included, the speed-up increases approximately two times.

Further speed-up might be achieved by incorporating a kind of lookahead when categorizing sequences into the two groups. This might help to (partially) avoid the observed stagnation in a local optimum.

Acknowledgments. This research has been supported by grant GAUK 57907 provided by the Grant Agency of Charles University.

References

1. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI–BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, 25:3389–3402, 1997.
2. G. J. Barton. An efficient algorithm to locate all locally optimal alignments between two sequences allowing for gaps. *Computer Applications in the Biosciences*, 9(6):729–734, 1993.
3. M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model for evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:345–352, 1978.
4. S. Dydel and P. Bała. Large scale protein sequence alignment using fpga reprogrammable logic devices. In *Field Programmable Logic and Application*, volume 3203, pages 23–32. Springer, 2004.
5. O. Gotoh. An improved algorithm for matching biological sequences. *J Mol Biol*, 162(3):705–708, December 1982.
6. Green. `http://www.genome.washington.edu/UWGC/analysistools/Swat.cfm`.
7. S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl Acad. Sci. USA.*, 89:10915–10919, 1992.
8. D. Hoksza and T. Skopal. Index-based approach to similarity search in protein and nucleotide databases. In *DATESO*, pages 67–80, 2007.
9. M. Itoh, S. Goto, T. Akutsu, and M. Kanehisa. Fast and accurate database homology search using upper bounds of local alignment scores. *Bioinformatics*, 21(7):912–21, 2005.
10. D. J. Lipman and W. R. Pearson. Rapid and Sensitive Protein Similarity Searches. *Science*, 227:1435–1441, Mar. 1985.
11. MPSrch. `http://www.ebi.ac.uk/MPsrch/`.
12. S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.
13. T. Ramdas and G. Egan. A survey of fpgas for acceleration of high performance computing and their application to computational molecular biology. In *TENCON 2005 2005 IEEE Region 10*, pages 1–6, 2005.
14. T. Rognes and E. Seeberg. Six-fold speed-up of smith-waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16(8):699–706, 2000.
15. P. H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *J. Algorithms*, 1(4):359–373, 1980.
16. T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–197, March 1981.
17. C. Wu, R. Apweiler, A. Bairoch, D. Natale, W. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. Martin, R. Mazumder, C. O'Donovan, N. Redaschi, and B. Suzek. The universal protein resource (uniprot): an expanding universe of protein information. *Nucleic Acids Res.*, 34(Database issue)(1):D187–D191, 2006.
18. W. Xu and D. Miranker. A metric model of amino acid substitution. *Bioinformatics*, 20(8):1214–1221, 2004.