Efficient alternatives to PSI-BLAST

Michał Startek[1], Sławomir Lasota[1], Maciej Sykulski[1], Adam Bułak[1],
Anna Gambin[1,2]
[1] *Institute of Informatics, University of Warsaw, 2 Banacha, 02-097 Warsaw, Poland*
[2] *Mossakowski Medical Research Centre Polish Academy of Sciences, 5 Pawinskiego, 02-106 Warsaw, Poland*
Laurent Noé[3], Gregory Kucherov[4]
[3] *LIFL/CNRS/INRIA, Bât. M3, Campus Scientifique, Villeneuve d'Ascq, France.*
[4] *Laboratoire d'Informatique Gaspard-Monge, Marne-la-Valle, France*

Abstract

In this paper we present two algorithms that may serve as efficient alternatives to the well-known PSI BLAST tool: SeedBLAST and CTX-PSI Blast. Both may benefit from the knowledge about amino acid composition specific to a given protein family: SeedBLAST uses a advisedly designed seed, while CTX-PSI BLAST extends PSI BLAST with the context-specific substitution model.

The seeding technique became central in the theory of sequence alignment. There are several efficient tools applying seeds to DNA homology search, but not to protein homology search. In this paper we fill this gap. We advocate the use of multiple subset seeds derived from a hierarchical tree of amino acid residues. Our method computes, by an evolutionary algorithm, seeds that are specifically designed for a given protein family. The seeds are represented by deterministic finite automata (DFAs) and built into the NCBI-BLAST software. This extended tool, named SeedBLAST, is compared to the original BLAST and PSI-BLAST on several protein families. Our results demonstrate a superiority of SeedBLAST in terms of efficiency, especially in the case of twilight zone hits.

The contextual substitution model has been proven to increase sensitivity of protein alignment. In this paper we perform a next step in the contextual alignment program. We announce a contextual version of the PSI-BLAST algorithm, an iterative version of the NCBI-BLAST tool. The experimental evaluation has been performed demonstrating a significantly higher sensitivity compared to the ordinary PSI-BLAST algorithm.

1. Introduction

Since the time complexity of the optimal alignment problem is quadratic (e.g., the Smith-Waterman algorithm [41]), thus too large for everyday tasks, most of sequence aligning is done using heuristics. One of such heuristics is implemented in the ubiquitous BLAST software [3, 4], remarkably successful in uncovering close homologs. Its extended iterated variant called PSI BLAST [4], similarly popular as pure BLAST, is more sensitive in detecting the harder-to-find distant homologs. However it also suffers from some disadvantages, e.g. when non-homologous proteins are incorporated into the profiles the corrupted model leads to meaningless results. To prevent this phenomenon, the homology detection method should extract knowledge specific to a particular family of proteins. In this paper we propose two different extensions of BLAST method that fulfill this requirement. One of them, SeedBLAST, explores the seeding technique [23], while the second one, CTX-PSI BLAST, benefits from the contextual alignment model proposed in [14]. Both tools have been developed on the basis of the source code of the NCBI BLAST tool[1].

SeedBLAST: seeds in protein homology search

Standard BLAST runs in three phases, the first of them finding short initial alignments, so called *hot spots*. However, quite different methods are applied to define a hot spot for DNA and protein sequences.

In the case of DNA, a hot spot is a short sequence of identically matching nucleotides. Application of seeds enables the consideration of non-identical matchings as well, and thus finding out previously overlooked good initial alignments. This is why *spaced seeds* have been intensively investigated and have successful applications: improvements of BLASTN [7, 40], sensitive alignment tools like PatternHunter [29, 25, 21] and Yass [34], automaton based theory for modeling and analyzing seeds [23, 9]. The idea of using *multiple seeds* is also widely recognized [25, 8, 42, 24]. In this paper we attempt to achieve similar results for protein homology search, using the approach of [39].

In the case of protein sequences, a hot spot is defined through a *cumulative* contribution of amino acid matches, not necessarily identical. A short sequence of such matches is considered a hot spot if their additive contribution (score) exceeds a predefined threshold. It is thus not clear whether seed-based approaches may measure up with the cumulative scores in expressibility and effectiveness. A first attempt to compare the two approaches has been done in [39], with the conclusion that *subset seeds* [23] may offer an attractive alternative to the "cumulative" approach of BLAST (cf. also discussion and references therein concerning expressibility of different classes of seeds). It is also argued that the algorithmic cost may thus be reduced, as application of seeds allows the use of a direct indexing scheme based on hashing.

A fundamental notion in seed theory is an alignment alphabet, whose letters correspond to matching two residues. In the case of nucleotide sequences, the alignment alphabet has 6 (or 12, if directional) letters. In the case of amino acid sequences, however, the alignment alphabet has at least 200 letters, which makes exploration of even medium length sequences costly and difficult. A way of approaching the problem is to reduce the alignment alphabet, exploiting

[1] National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov.

similarities among various amino acids [39]. By applying the *subset seed* the complexity of alignment description may be reduced, while maintaining the biological information content. The idea of subset seeds [23], can be viewed as an intermediate concept between ordinary spaced seeds and vector seeds. In this approach different types of matches (or mismatches) are distinguished, as a seed letter corresponds to a subset of matches. In the case of protein sequences, for instance, it might be beneficial to distinguish mutations inside some predefined amino acid groups (like aliphatic, aromatic, tiny, etc. [28]) from mutations between these groups.

The aim of this paper is to experimentally confirm the value of applying seed-based hot spot search, using the approach of [39]. In short, as our technical contribution we propose a method of computing a well-performing multiple space seed, and present an implementation of a new seed-based hot spot search routine. Furthermore, we advocate the use of deterministic finite automata (DFAs) as a seed representation. Finally, we experimentally confirm a supremacy of this new approach over the original NCBI-BLAST hot spot search.

We investigate, and search for, reduced alignment alphabets, called *seed alphabets*, that can be derived from hierarchical trees of amino acids. Such trees were designed, e.g., in [26, 32]; for our purposes we compute, (by amino acids clustering), a specific tree for a given protein family. An advantage of using hierarchical trees is that the alphabets are always *transitive* (i.e., each letter corresponds to a transitive set of matching pairs) and thus enable application of the direct hashing scheme.

We search for a well-performing alphabet and a multiple subset seed over it with the use of an evolutionary algorithm. The fitness evaluation is based on computing the seed sensitivity and selectivity in the way suggested in [23].

The multiple seed, represented as a DFA, is then used in the hot spot search of BLAST. We have implemented an extension to the NCBI-BLAST software, called SeedBLAST, that accepts a multiple subset seed as its input parameter. The extension is written in **C++**, relies on the template mechanism, and is prone to compiler optimizations (most functions can be *inlined*). An important advantage of our implementation is that being developed within the NCBI-BLAST framework, it inherits all stable and tested features of this implementation.

The first test results can be perceived as promising: although our multiple seed selection method is rather simplistic, our tool returns more interesting hits than the standard BLAST with comparable settings. Some returned hits tend to be long although having only medium E-value, the type of hits known to be dimmed and not reported by BLAST. This kind of hits is termed *twilight zone* after [38].

Furthermore, this methodology can be useful for searching for particular type of alignments. Given a set of alignments, one can construct a specific seed automaton and perform database search for this certain type of alignments. Following this idea we investigated the ability to align known structurally homologous domains of the Rhodopsin family of G-protein coupled receptors (GPCRs). The outcome of our experiment showed a significant difference between NCBI-BLAST and SeedBLAST, in favor to the latter: our method yielded much longer alignments covering up to 70% of the entire domain, even for proteins sharing low sequence identity (20-30%).

CTX-PSI BLAST: context-sensitive protein homology search

The contextual (*context-sensitive*) alignment of biological sequences was proposed in [14], as an extension of the classical alignment where the neighboring residues influence the cost of substitutions. The paper [14] introduced the rudiments of the theory of contextual alignment, and a prototype tool based on the dynamic algorithm of Smith and Waterman [41]. A complement of this work was the computation of biologically significant contextual substitution tables [15], based on BLOSUM family [19].

The contextual approach is significantly more complex than the ordinary one, both with respect to the underlying theory as well as from algorithmic point of view. To only mention one additional new aspect brought by the approach, note that the final score of alignment typically depends on the order of substitutions performed, as illustrated in Fig. 1.

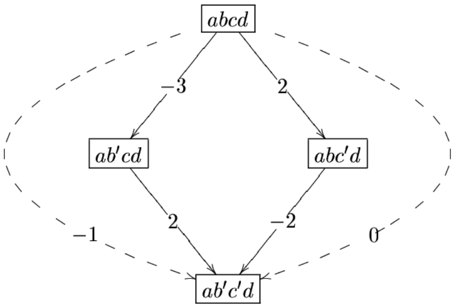

Figure 1: The substitution score may depend on the order of individual substitution. In the example, the score of substitution $b \mapsto b'$ in the context (a, c) is -3, but in the context (a, c') it is -2. On the other hand, the score of the substitution $c \mapsto c'$ equal 2 both in context (b, d) and (b', d).

On the positive side, it has been observed that the contextual approach increases sensitivity of alignment. These promising results encouraged for a further work on efficient implementation of the contextual alignment. A a natural next step, in [16] a contextual version of the famous BLAST [3] algorithm has been proposed. Significantly higher sensitivity of the contextual alignment was confirmed, while keeping the amount of additional computations needed on a reasonable level.

In this paper we do a next step in the program: we announce a contextual version of PSI-BLAST. We perform an experimental evaluation of accuracy of our algorithm compared to structural alignments, using the methodology of [13].

Organization of the paper. In Section 2 we present the algorithm to design the seeds used for the protein alignment and then is Section 3 we describe how the SeedBLAST tool uses the seeds. Section 4 is devoted to the presentation of the other tool, CTX-PSI BLAST, a contextual extension of PSI BLAST. Finally, in Sections 5 and 6 both tools are evaluated: their efficiency and sensitivity are compared with BLAST and PSI BLAST. The last section summarizes some directions for further work. An initial version of this paper, presenting SeedBLAST, has appeared as [17].

2. Subset seed design

General approach. Given a protein family, we assume that a small representative subset of this family has already been aligned well (for example manually by experts), and is available as a training set. The algorithm designing subset seed attempt to extract information about the structure of the family from this set, and use it to produce alignments for the entire family. In the first phase a hierarchical tree is constructed that represents similarities of amino acids. Then, a *seed alphabet* is designed, along with a set of *seeds.* This is a learning phase, and runs independently of our BLAST enhancement. Next, the seed alphabet along with the corresponding set of seeds is used by the SeedBLAST algorithm to find hot spots. Afterwards, the computation of SeedBLAST follows the standard BLAST scheme.

Hierarchical tree of amino acids. Let $\Sigma = \{A, C, D, \ldots\}$ be the amino acid alphabet ($|\Sigma| = 20$). A valid hierarchical tree of amino acids is a binary tree whose leaves are labeled bijectively by elements of Σ, and whose every internal (non-leaf) node has two children. An example of such a tree is shown in Figure 2. Such a tree constitutes a parameter in our approach; we assume that it corresponds to some biologically significant hierarchical clustering of amino acid residues, c.f. [32, 26]. Any non-leaf node v of T is represented by a set of (labels of) leaves in the subtree rooted in v. This set is denoted by Σ_v. In particular, the root is labeled by the whole set Σ. There are precisely $|\Sigma| - 1 = 19$ non-leaf nodes.

Our basic intuition is as follows. Think of a leaf labeled by $A \in \Sigma$ as a representation of the exact match A—A. Then a node v represents all matches A—B for $A, B \in \Sigma_v$.

The tree is obtained from the training set of alignments in the following way: first, for each pair of amino acids the number of times they have been aligned one with another is counted, and then, using those counts, the amino acids are hierarchically clustered through neighbor-joining method.

Seed alphabets and seeds. From now on we assume a fixed hierarchical tree T. The tree nodes are partially ordered by a natural ordering induced by the tree structure (we call it *tree ordering*). This coincides with the inclusion ordering of the labeling sets: $v_1 \leq v_2 \iff \Sigma_{v_1} \subseteq \Sigma_{v_2}$. We assume here for technical

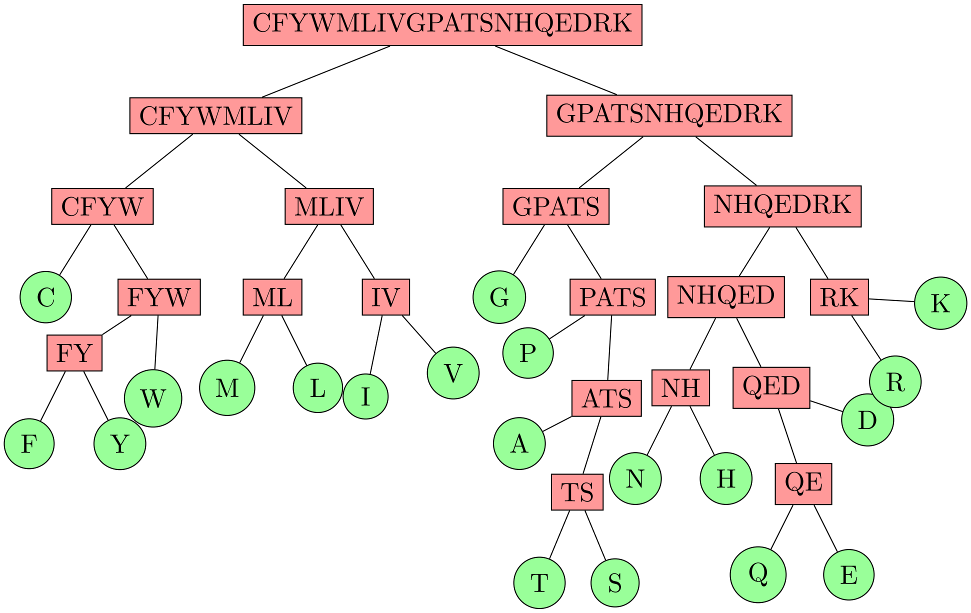

Figure 2: The hierarchical tree of amino acids proposed by [26].

convenience that the leaves are labeled by singletons $\{A\}$ instead of single amino-acids $A \in \Sigma$. Below we consider sets of nodes of T, ordered by inclusion as well. Certain sets of nodes will be *seed letters* (potential elements of a *seed alphabet*).

A *seed letter* is defined as any subset α of nodes such that:
(i) *(maximality)* α contains all leaves and
(ii) *(downward closedness)* whenever $v \in \alpha$ and $v' < v$ then $v' \in \alpha$.

Hence, a single seed letter α is defined as a lower set of a maximal antichain wrt. the tree ordering. This antichain contains the maximal elements of α wrt. the tree ordering and may be visualized by a *horizontal cut through the tree* T. Seed letters are naturally ordered by inclusion. The smallest one is the "exact match" seed letter #, containing only the leaves. The largest one is the "don't care" seed letter _, containing all the nodes of T. One particular seed letter, denoted by @, is obtained by removing from _ the root node. We place an additional restriction on alphabets that we use, that they must contain both # and _.

The maximal elements of a seed letter α wrt. the tree ordering form a partition of Σ. Thus α represents naturally an equivalence relation on Σ: A and B are related iff they belong jointly to some node of α; i.e., iff there exists some $v \in \alpha$ such that $A \in \Sigma_v$ and $B \in \Sigma_v$. We feel free to write $(A, B) \in \alpha$ in this case. The induced equivalence is identity relation in case of # and full relation in case of _. The inclusion ordering of seed letters coincides with the inclusion of the induced equivalences.

Certain families of seed letters will be allowed as seed alphabets. Essentially, we forbid two letters α_1, α_2 that are incomparable by inclusion. A *seed alphabet* is a family $\mathbb{A}$ of seed letters totally ordered by inclusion: for each $\alpha_1, \alpha_2 \in \mathbb{A}$, either $\alpha_1 \subseteq \alpha_2$ or $\alpha_2 \subseteq \alpha_1$. Alphabets with this property are called *hierarchical* in [39]. We used this assumption as it leads to a nice mathematical formalization, namely the family of seed alphabets forms a constrained independence system [22, 11]. We show that even with this restriction very efficient seeds can be obtained. Thus, in this paper we will not consider non-hierarchical alphabets. Note that, again, the seed alphabets may be naturally ordered by inclusion as well.

We define a *seed* over a seed alphabet $\mathbb{A}$ as a finite word over $\mathbb{A}$. A *multiple seed* is a pair consisting of a seed alphabet and a set of seeds over that alphabet. We say that a seed $s = s_1 s_2 ... s_n$ aligns two amino acid sequences $a = a_1 a_2 ... a_n$, $b = b_1 b_2 ... b_n$, if and only if for all $i \in \{1, 2, ..., n\}$, $(a_i, b_i) \in s_i$.

Foreground sensitivity (or just sensitivity) of a multiple seed M, denoted by $sens^F(M)$, is the number of positions in the training set of alignments matched by at least one of the seeds from M, divided by the total number of positions. Foreground sensitivity is computed directly from the training set.

Background sensitivity of a seed corresponds to the probability of matching two aligned random sequences. We assume that the background model for amino acid sequences is given as a Markov chain. For our experiments, the Markov chain models of orders $1, 2$ and 3 were learned from the TrEMBL database [6] using `GenRGenS` Java tool [37]. The background sensitivity of a seed was computed with the use of Markovian probability transducer as described in [23]. Background sensitivity of a multiple seed M, denoted by $sens^B(M)$, is estimated from above by the sum of background sensitivities of each of the individual seeds in M (the estimation is sharp only if seed occurrences are independent).

Evolutionary approach. Optimizing multiple seeds is recognized as a highly nontrivial task [43, 9, 30]. In the case of hierarchical subset seeds the combinatorial structure of seed alphabets suggests hardness of the optimization problem (see [39] for details). Therefore we decided to use an efficient heuristic algorithm.

In the proposed approach seed alphabets and seeds are simultaneously chosen through an application of a genetic algorithm. The genetic algorithms are used to solve various optimization problems [31]. They work by first generating a random multiset (*initial population*) of potential solutions, evaluating the function being optimized (*fitness function*) for each one of them, culling a percentage of them with low values of such function, cloning and slightly altering (mutating) the rest at random – and repeating this process until a satisfactory solution is obtained.

In our case, the potential solutions are pairs: a seed alphabet, and a set of seeds. A mutation applies thus either to the alphabet, or to one of the seeds. Mutating the alphabet is one of the following: deleting a randomly chosen letter (except for the top # and bottom _ one), altering a letter (by adding a tree node to it, or removing a tree node – but only if that would not violate the constraint that the alphabet must be hierarchical), or adding a random (non-conflicting) letter. While modifying the letter one has to respect its definition, i.e. the *(i) maximality* and *(ii) downward closedness* conditions. Mutating the set of seeds means either deleting one of the seeds, adding a random seed, or replacing a random letter in a random seed by one of its neighbors in the alphabet. Algorithm 1 explains the details.

A multiple seed may contain individual seeds of different lengths in general. However to simplify and speed-up the computations we have decided to fix the length; all individual seeds computed by the evolutionary algorithms have the same length W = 5.

Algorithm 1: Genetic Algorithm

```
Input: Protein family $F$
Output: A multiple seed for family $F$
begin
    Population ← a multiset of 100 randomly chosen multiple seeds (the
    initial population);
    while Not Run Out Of Time do
        foreach multiple seed $M \in$ Population do
            $f \leftarrow fitness_F(M)$;

            Randomly, based on f, choose one of the following:
                • Population ← Population $\setminus \{M\}$;
                  - with increasing probability for
                  low values of $f$

                • Population ← Population $\setminus \{M\} \cup$
                  $\{$Mutate$(M)\}$;

                • Population ← Population$\cup$
                  $\{$Mutate$(M)\}$; - with increasing
                  probability for high values of $f$
        end
    end
    return the member of Population that maximizes $fitness_F$
end
```

The most important aspect of every optimization algorithm, a genetic algorithm being no exception, is the fitness function chosen. Usually, what we want to obtain is a seed that has as low background sensitivity as possible, while at the same time having as high foreground sensitivity as possible. So, the first idea might be to choose the following function:

$$fitness_1(M) = \frac{sens^F(M)}{sens^B(M)}.$$

This, however, yields unsatisfactory results – the evolution just results in a smallest multiple seed possible, with minuscule foreground and background sensitivity.

The fitness function has to reflect the trade-off between foreground sensitivity and background sensitivity. It should be noted that both of these play similar role to NCBI-BLAST '-f' parameter (i.e. the threshold for the cumulative score of three hit positions). The '-f' parameter allows one to adjust the length of computation, and the quality of results. With SeedBLAST it has been split in two – the $sens^F(M)$ part is responsible for the quality of results, while $sens^B(M)$ is responsible for the length of computation. Keeping that in mind, we can select a fitness function that can match our needs – using it, we can in effect specify 'give me the best results you can achieve within a given timeframe' – or, the opposite – 'give me results at least this good, and I don't care how long it takes to compute them'. Or everything in-between.

An example of fitness function that adheres to the first approach might be as follows:

$$fitness_2(M) = \begin{cases} 0 & \text{if} \quad sens^B(M) > c \\ sens^F(M) & \text{otherwise} \end{cases}$$

The second approach is fulfilled by the following fitness function:

$$fitness_3(M) = \begin{cases} sens^F(M) & \text{if} \quad sens^F(M) < c \\ \frac{sens^F(M)}{sens^B(M)} & \text{otherwise} \end{cases}$$

For further tests, described in the rest of the paper, we have chosen the function $fitness_3$, with $c = 0.15$; except for the performance evaluation, where we prefer to use $fitness_2$ (in order to make the fair comparison with NCBI-BLAST).

This decision was taken through trial and error - there is no guarantee that this is the optimal choice. The multiple seed that was computed and used for further experiments exhibits foreground sensitivity equal to 0.179906, and background sensitivity equal to 0.01047971. The whole multiple seed, consisting of 3686 individual seeds, is not subject to a concise presentation.

3. SeedBLAST: seed-based extension of BLAST

Given a query, the goal of the first phase of the BLAST algorithm is to index all subwords of length W (chosen as a parameter). Not only exact subwords are indexed but also their predefined neighborhoods, with respect to a metric determined by the cumulative score according to the BLOSUM matrix. With each query, the occurrences of the neighborhoods are stored in a dictionary-type data structure; current version of NCBI-BLAST uses a hash table.

The size of neighborhood is crucial as it must be stored in a dictionary. BLAST uses a threshold on the BLOSUM score of an alignment of a segment pair. The threshold represents the trade-off between sensitivity and time and memory efficiency since it has a direct impact on the number of analyzed hits.

The default threshold was adjusted experimentally by the BLAST developers and currently equals 11 in protein NCBI-BLAST.

We seek to describe the neighborhood using our selected multiple seed. In principle, the method may be applied to any multiple seed, possibly containing words of different lengths. However, in the case study described in the following section, all the seeds have the same length W = 5. Moreover, all individual seeds are constructed over the same alphabet. This assumption greatly simplifies the seed design and allows to construct a single automaton for looking for all hot spots simultaneously.

3.1. Hot Spot Search Using DFA

A *trie*, or a *prefix tree*, is a dictionary with a tree-structured transition graph, in which the start node is the root and all the leaves are final nodes [27]. Tries are especially convenient when the keys are short strings: the tree edges are labeled by letters, and retrieving a value assigned to a given key w is done by following the w-labeled path in the tree, thus very efficient.

It is assumed that labels of edges outgoing from a node are all different. A trie may be thus seen as an acyclic DFA recognizing a finite language (the language contains labels of all the paths going from the root to a leaf). Upon acceptance, the automaton in addition returns the value assigned to a word read (being a key). In our case, the value will be typically a set of positions in a query.

In our algorithm, to be described below, we construct a number of different tries (automata). To optimize for memory, on the implementation level we always conform to the *Mealy paradigm* of keeping values attached to transitions, not vertices.

In a preprocessing phase a trie S is constructed to represent the multiple seed. Its input alphabet is the seed alphabet $\mathbb{A}$.

Next, we proceed with constructing a trie Q, over the input alphabet Σ, that keeps all subwords of length W from a given query. For each such word we store in Q pointers to all positions in query where it appears. This will reduce operations in the following phases. It is worth noting that Q may be used to process jointly multiple queries. Analogously, NCBI-BLAST also permits many queries to be stored jointly in its hash table.

As a consecutive step, a trie N is built to store neighborhoods. Its alphabet is Σ, and language is given by

$$N = Q \propto S :=$$

$$\{w \mid \text{for some } q \in Q \text{ and } s \in S, s \text{ aligns } q \text{ and } w\}.$$

The trie N is constructed by systematically traversing a *product* of Q and S. The value assigned to a word w in N denotes, similarly as in Q, a set of positions in the query. It is given by the union of values assigned to q in Q, for all q ranging over

$$\{q \in Q \mid \text{for some } s \in S, s \text{ aligns } q \text{ and } w\}.$$

On the implementation level, the union is represented by a suitable pointer data structure.

Finally we construct an automaton H over the alphabet Σ, whose aim is to find hot spots in the subject sequences. Operation of H is similar to a pattern-matching automaton. It is built on the basis of the automaton N, by adding additional edges outgoing from the final (leaf) nodes. To easily explain the construction, we recall that each node of N is uniquely determined by the labeling of the path from the root to that node. Fix a leaf determined by w and a letter $a \in \Sigma$; the outgoing a-labeled edge will point to a node determined by the longest suffix of wa that belongs to N. Clearly, in contrast to all other automata, H may have cycles.

Having constructed H, next BLAST phases remain unchanged. Each subject sequence is traversed along, starting from the root of H. At each step, the value assigned to the current node (state) of H informs whether any hot spots are found at the current position in a subject. If so, the hot spots are stored for further processing in the following phases of BLAST.

4. CTX-PSI BLAST: contextual extension of PSI-BLAST

CTX-PSI-BLAST, the contextual extension of PSI-BLAST, has been implemented as an extension of the NCBI BLAST tool. We have also exploited the CTX-BLAST source code [16]. The web page of the contextual PSI-BLAST project[2] contains further informations, user documentation, and the complete source code.

CTX-PSI-BLAST, similarly as PSI-BLAST, constructs in each phase a PSSM (Position-Specific Scoring Matrix) based on a multi-alignment found in that iteration. Assume that the alphabet is of size m; typically $m = 20$ as we primarily consider protein sequences here. In standard PSI BLAST, a PSSM is an $n \times m$ matrix, where n corresponds to the length of the query sequence. In entry (i, x) the matrix stores the score of aligning the ith column with x.

In our contextual approach, a PSSM is an $n \times m^3$ matrix, whose entry indexed by (i, x, a, b) contains the score of aligning the ith column with x, denoted $i \mapsto x$, *in the context* (a, b). Thus the context consists of the two neighboring symbols, the left neighbor a and the right neighbor b, and the score may depend on these two symbols.

Here are the major extensions we have introduced to the original PSI-BLAST code:

1. replacing the classical BLAST alignment routine with the one provided by CTX-BLAST;
2. contextual extension of the PSSM (Position-Specific Scoring Matrix) data structure;
3. adaptation of the method of computing a PSSM;

[2] http://ctx-psi-blast.sourceforge.net/index.html

4. adaptation of the alignment algorithm against a PSSM, to work with the contextual PSSMs.

In the first iteration a CTX-BLAST routine is called, and a contextual PSSM is computed. In every subsequent iteration, an adapted CTX-BLAST routine is run, that aligns, instead of the input sequence, a PSSM computed in the previous iteration. However, unlike in CTX-BLAST, it is sometimes not clear how to choose a context of a substitution, as a PSSM is considered instead of a query sequence. As an illustration, consider the example given in Fig. 4. For a distinguished substitution $8 \mapsto D$, it is clear what is its left context only if the position 7 is already substituted. The same applies to the right context of $8 \mapsto D$. We have decided to apply a simplifying solution in such situations, and to choose as context the neighboring symbols from the subject sequence S. In the example from Fig. 4, the left context is thus chosen to be A and the right one is E.

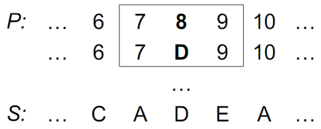

Figure 3: A fragment of alignment of a subject sequence S agains a PSSM. A substitution $8 \mapsto D$ is distinguished, in the context $(7, 9)$.

The crucial point of the CTX-PSI-BLAST algorithm is the calculation of statistical significance of the alignment. According to [18], statistics of optimal non-gapped contextual alignment follows the same extreme value distribution as in the non-contextual case [2]. Hence we have decided to adopt the island method [1], used also by [16], for estimation of the parameters K and λ required for E-value calculation for the alignment score S of two sequences of length m and n, respectively:

$$\text{E-value}(S) = Kmne^{-\lambda S}.$$

Using the method suggested in [1], we have obtained the values $K = 0.008$ and $\lambda = 0.211$.

5. Evaluation of SeedBLAST

Datasets. We used a dataset extracted from the Pfam database, that contains expert-made protein structural families and their multi-alignments, later extended to larger families using *profile-HMMs* [5, 12].

A protein family, exhibiting low identity percentage, has been selected from Pfam (namely PF00001). This family contains, amongst other G-protein-coupled receptors (GPCRs), members of the opsin family, which have been considered to be typical members of the rhodopsin superfamily. They share several motifs, mainly the seven transmembrane helices (7tm_1 domain). This domain will be the main focus of our experiment.

The rhodopsin-like GPCRs themselves represent a widespread protein family that includes hormone, neurotransmitter and light receptors, all of which transduce extracellular signals through interaction with guanine nucleotide-binding (G) proteins. Although their activating ligands vary widely in structure and character, the receptors are believed to adopt a common structural framework comprising 7 transmembrane helices.

The expert-made multi alignment of 7tm_1 domains from 64 of the family members was downloaded (the whole family contains 16975 proteins), and used as a training set to obtain a multiple seed. The latter was subsequently used by the SeedBLAST algorithm to compute pair-wise alignments of the 7tm_1 domain of all the family members. The results were compared with those obtained by the standard BLAST algorithm.

For a fair comparison, it had to be ensured that both algorithms actually run with the same background sensitivity. Thus, the '-f' parameter of NCBI-BLAST was adjusted in the course of the experiment to obtain similar background sensitivity to that of the multiple seed used by SeedBLAST. The table shows typical values of the '- f' parameter together with the corresponding values of background sensitivity:

-f parameter of NCBI-BLAST	SeedBLAST background sensitivity
11 (default)	0.002195
10	0.005342
9	0.00816
8	0.012276
7	0.018163

The background sensitivity of the seed used by SeedBLAST was 0.01047971; this corresponds to 8 or 9 as the value of the '-f' parameter in the NCBI-BLAST invocations.

As the alignment concerned only the domain fragment of each protein, and the domain is already known to be the same in each protein, every alignment found should be considered biologically significant.

Comparing efficiency. Figure 4 shows the symmetric difference between hits found by NCBI-BLAST, and those found by SeedBLAST (that is, alignments found by one of the algorithms but not the other).

We observe that alignments found by SeedBLAST are in general longer than those found by BLAST, and thus provide better coverage of the domain. Especially many alignments that have not been found by BLAST lie in the so-called twilight zone [38] – namely long alignments with low identity percentage, and thus low E-value, that nevertheless are biologically significant. The reason why SeedBLAST constructs longer alignments is that it detects much more biologically significant hot-spots. A supremacy of SeedBLAST becomes more apparent in view of Figure 5.

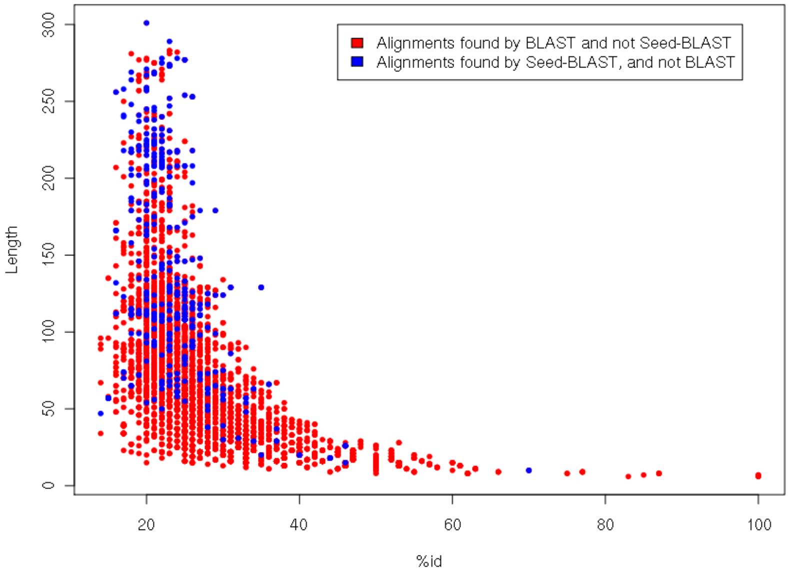

Figure 4: Symmetric difference between outputs of BLAST and SeedBLAST.

To obtain this diagram, pairs of domains for which BLAST and SeedBLAST found different alignments were chosen from the set of all alignments, and the coverage of domains by these alignments was computed. We conclude that SeedBLAST is much more efficient in providing biologically significant alignments than the standard BLAST algorithm. The reason for SeedBLAST's improved efficiency is the inclusion of subset seeds specifically tuned for the domain under consideration. When one aims at aligning different family of proteins appropriate multiple seed should be designed and used in SeedBLAST. We conclude that SeedBLAST appears much more efficient than BLAST in recognizing protein domains, as it is both more effective in covering the entire domain as well as much less likely to cover anything beyond the sought-for domain.

Comparing running time. In addition, SeedBLAST and NCBI-BLAST were compared with respect to their running time (preprocessing, i. e. seed design phase is not included). For a fair comparison, again, we had to ensure that both algorithms actually run with the same background sensitivity.

In case of SeedBLAST, we had to be able to control the background sensitivity of the multiple seed used. This led us to choose the fitness function:

$$fitness_2(M) = \begin{cases} 0 & \text{if} \quad sens^B(M) > c \\ sens^F(M) & \text{otherwise} \end{cases}$$

(cf. Section 2) that seems to suit best to this purpose: the parameter c corre-

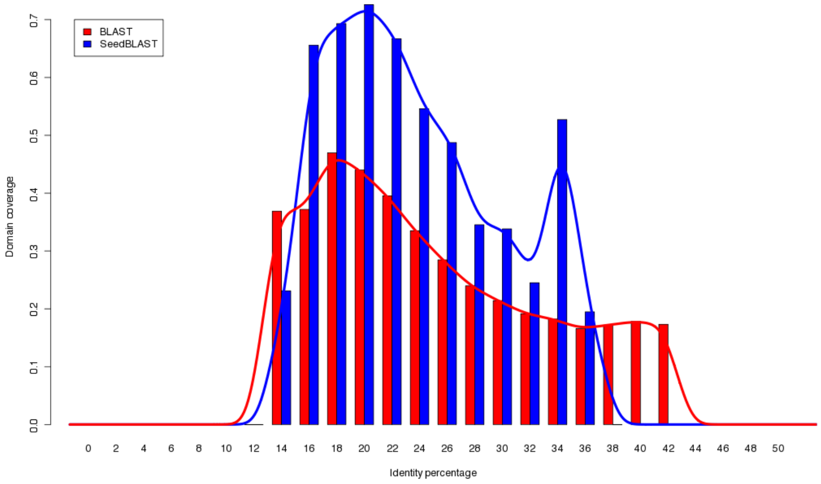

Figure 5: Domain coverage by alignments found by BLAST and SeedBLAST.

sponds directly to the desired background sensitivity of the multiple seed.

In case of NCBI-BLAST, its background sensitivity can be adjusted by the '-f' parameter. For the test, we picked several different values of the '-f' parameter, and then calculated the background sensitivities induced by these values. These background sensitivities were taken as the value of the c parameter in the above fitness function, exploited in the computation of multiple seeds used by SeedBLAST.

The results are summarized in the Table 1.

In fact, because of the unpredictable nature of multiple seed evolution, we can't control the background sensitivity of a multiple seed exactly. This gives SeedBLAST a slight advantage over the other, represented by the difference between the c parameter (equal to the actual background sensitivity NCBI-BLAST runs with), and the background sensitivity of an obtained multiple seed. Still, even accounting for this slight difference, the results show that SeedBLAST algorithm is over two times faster than NCBI-BLAST on average. As the SeedBLAST is an extension of NCBI-BLAST we conclude, that the speed-up is achieved due to faster hot-spot identification stage. We argue that the multiple seed approach enables to detect biologically significant hot-spots, e.g. those corresponding to functional residues [35].

It is worth mentioning here that the performance of SeedBLAST (being the extension of standard NCBI-BLAST implementation) is comparable with the performance of subset seed based tools that use parallel implementation or specialized hardware [36, 33].

Comparison with PSI-BLAST. One can see the close similarity between the seed approach and position specific scoring matrices used to improve homol-

Table 1: f: -f parameter of NCBI-BLAST c: corresponding background sensitivity (parameter c) used in seed design, $sens^B(M)$: actual background sensitivity of the multiple seed SeedBLAST, NCBI-BLAST: running time of NCBI-BLAST,SeedBLAST: running time of SeedBLAST.

f	c	$sens^B(M)$	NCBI-BLAST (sec.)	Seed-BLAST (sec.)
15	0.000306	0.00030564	1.20	0.45
11	0.002195	0.00211789	3.32	1.40
8	0.012276	0.01156471	12.17	4.74
5	0.026406	0.02622019	23.84	12.40

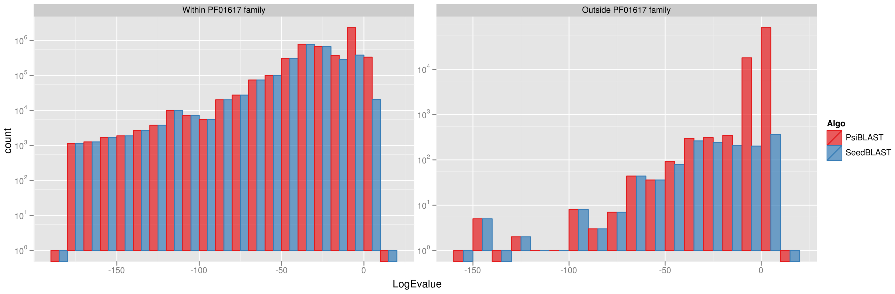

Figure 6: Histogram of found alignments grouped by logarithm of their E-value. We can observe that within the Antigen family SeedBLAST finds all of the alignments that PSI-BLAST does (except for a small fraction of some non-significant ones with E-values of 1 and more). Histogram on the right side shows results of alignment of proteins which we know to be unrelated to Antigens with Antigens. We can see that SeedBLAST finds less non-homology-related alignments than PSI-BLAST does.

ogy search. Therefore we decided to compare the selectivity and sensitivity of SeedBLAST to the efficiency of popular PSI-BLAST algorithm. The experiment was performed on two protein families (Surface antigen - PF01617 and Globin - PF00042). The performace of both algorithms on Globin family was almost identical (data not shown). On the other hand on Antigen family SeedBLAST achieved much better selectivity while keeping the same level of sensitivity (cf. Fig. 6).

6. Evaluation of CTX-PSI-BLAST

The experimental evaluation of the CTX-PSI-BLAST algorithm relies on a methodology applied in [13] for PSI-BLAST. The authors of [13] selected 123 pairs of evolutionary distinct structural homologous protein sequences. PSI-BLAST was run for each of the sequences, in order to check if the other sequence from the pair will be selected from the database. For all hits, a degree of similarity to the structural alignment has been computed. Out of 123 pairs, 36 have been found, which amounts to 29.3%. For 16 pairs, both sequences of the pair have been found.

Since this experiment has been performed, the PSI-BLAST tool undergone many improvements and optimizations. Our implementation has been done based on the BLAST version denoted as `Mar_17_2008`[3]. For the purpose of reliable comparison, we have repeated the experiment of [13] using the same version of PSI-BLAST. Then, the same experiment has been performed using CTX-PSI-BLAST.

6.1. Input data

The 123 sequence pairs were selected from the DAPS (Distant Aligned Protein Sequences) database according to the following criteria:

1. lengths of both sequences are at least 30
2. resolution of both sequences at least 350 pm
3. the difference of length is at most 50% of the smaller length
4. the length of the structural alignment is at least 60% of the greater length
5. the similarity is not identified with the Smith-Waterman algorithm.

The average similarity ratio of the selected pairs was 12%.

As in [13], for tests we have used the NR database (NonRedundant nucleotide database). In our experiments, the PSI-BLAST and CTX-PSI-BLAST were run, similarly as PSI-BLAST in [13], with the default parameters and 5 iterations. The only parameter that was set explicitly was the number of sequences yielded (parameter `num_alignments`), by default set to 250. We have chosen to switch this parameter off – in such case all sequences are yielded with e-value smaller than 10.

For estimating quality of the results we have used two ratings:

- *sensitivity* - the ratio of properly located amino-acid pairs to the length of the DAPS alignment
- *specificity* - the ratio of properly located amino-acid pairs to the length of alignment found by CTX-PSI-BLAST.

6.2. Results

PSI-BLAST identified 46 pairs (37.4%) out of 123. In case of 22 pairs, both sequences have been found. For CTX-PSI-BLAST, the corresponding numbers of hits are 43 (35%) and 21. Notably, the results of PSI-BLAST were significantly better than those reported in [13] – this is implied by recent improvements of the algorithm, as well as by continuous updates of the database used. The following table summarizes the results:

[3] The source code is accessible at: ftp://ftp.ncbi.nih.gov/toolbox/ncbi_tools++/2008/Mar_17_2008/

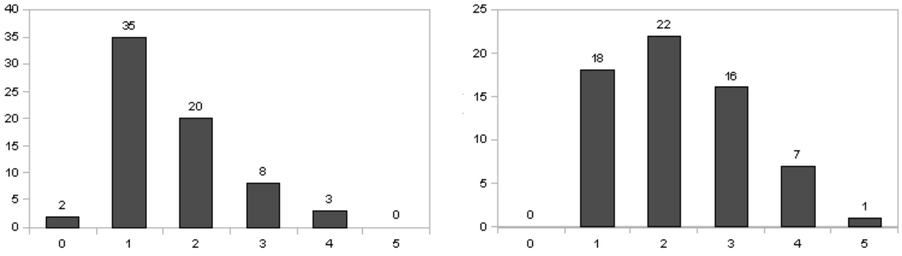

Figure 7: The iteration that found a pair. 0 denotes the initial iteration that runs without a PSSM. PSI-BLAST (on the left-had side) compared to CTX-PSI BLAST (on the right-hand side).

	PSI-BLAST	CTX-PSI-BLAST
total number of hits	68 (28.8%)	64 (26%)
number of identified pairs	46 (37.4%)	43 (35%)
mutual hits	22	21
average sensitivity	42.96%	44.78%
average sensitivity in the best hit	48.8%	47.64%
number of lost hits	4	7

The last row contains the number of pairs that have been located by the algorithm at some intermediate iteration but were not included in the final result yielded after the last iteration.

With respect to the number of pairs identified, PSI-BLAST slightly out-performed the contextual one. With respect to the quality of results, the two algorithms were very comparable. For instance, the fraction of hits above the 50% sensitivity threshold was around 45% in both cases. Interestingly, this is a major progress compared to the experiments in [13] where this number was only around 35%.

A valuable observation is that the contextual PSI-BLAST identified 6 pairs not located by the original PSI-BLAST. As a conclusion, we argue that CTX-PSI-BLAST, although itself does not out-perform PSI-BLAST with respect to sensitivity, may be considered a valuable tool used in combination with PSI-BLAST.

In the remainder of this section we present detailed results yielded during the experiment by PSI BLAST and CTX-PSI BLAST. For comparison, we stick to the same format as in [13]. Figure 7 illustrates the performance of both algorithms in subsequent iterations. Notice, that contextual algorithm identifies much more structural homologs during the second and subsequent iterations than the standard PSI BLAST. The probable reason for this behavior may be greater resistance to non-homologous proteins which are more rarely incorporated into the profile.

Other interesting phenomenon can be observed in Figures 8 and 9: sensitivity and specificity for selected iterations are closely correlated in the CTX-PSI

BLAST algorithm, while in the standard approach, no such correlation can be observed. This is mainly due to the fact that contextual alignment is usually longer and has size comparable to the structural alignment size. Similar observations were made in [16].

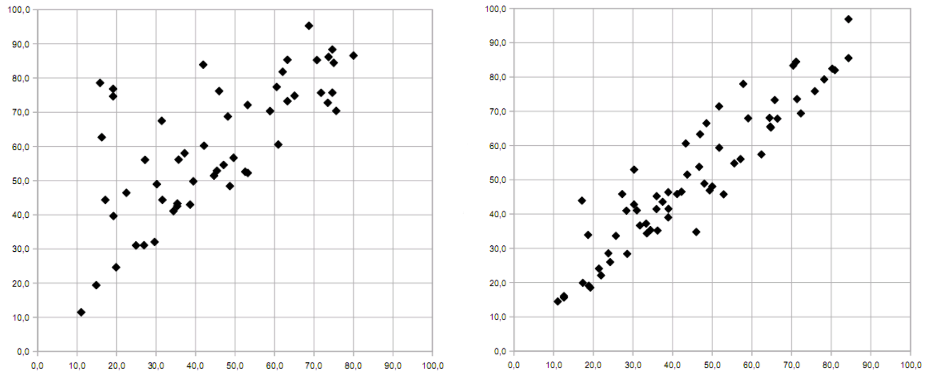

Figure 8: Sensitivity on X axis vs specificity (Y axis) for the iteration that yielded the first hit. PSI-BLAST (on the left-had side) compared to CTX-PSI BLAST (on the right-hand side).

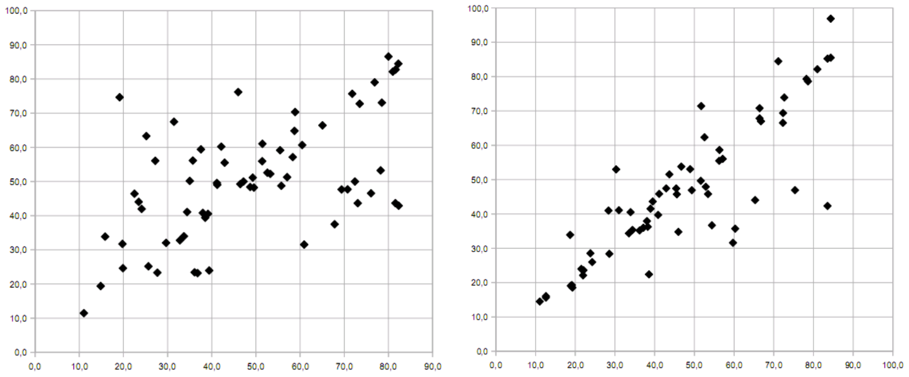

Figure 9: Sensitivity on X axis vs specificity (Y axis) for the iteration with the best sensitivity. PSI-BLAST (on the left-had side) compared to CTX-PSI BLAST (on the right-hand side).

7. Further research

To confirm usability of both the tools, further experiments using in-sample, out-of-sample tests and larger families would be useful. In case of SeedBLAST, its current performance is comparable with BLAST running with low threshold for hot spots. We hope that this result can be improved if more advanced methods for construction of multiple seeds are used, and longer seeds (i.e., of length > 5) are included. The remaining goal is to develop a method of seed construction that would keep sensitivity high and improve selectivity. That would eventually reduce the running time and memory requirements for greater seed lengths.

Protein homology search requires unavoidably storing a large dictionary in memory. Hence it seems to be worth pursuing ideas from [20], where a novel method of compression, based on *wavelets*, was proposed for dictionaries of words. Another possible improvement could be integration of the *cache-conscious* hashing DFA to improve efficiency of page-swapping, as described in [10]. However, we would like to recall here that our overall goal of investigating the seed-based hot spot search was to reduce the need for large information storage by choosing only those hits that seem important.

In case of CTX-PSI BLAST, there also remains a room for improvements. First, by now we did not apply the contextual model to the hot spot search. We believe that this extension could yield a further increase of sensitivity. Second, the use of reduced contextual substitution tables (see [18]) may reduce the complexity, while keeping sensitivity and specificity on the high level.

Finally, a possible continuation is to combine both improvements, namely application of seeds in hot spot search with the contextual extension of alignment procedure. As interesting question is whether this would bring additional increase of accuracy compared to joint but independent application of both tools, SeedBLAST and CTX-PSI BLAST.

Acknowledgements

The research reported in this paper has been partially supported by Polish Ministry of Science and Higher Education grant nr N N301 013436.

References

[1] S.F. Altschul, R. Bundschuh, R. Olsen, and T. Hwa. The estimation of statistical parameters for local alignment score distributions. *Nuclear Acids Res.*, 29(2):351–361, 2001.

[2] S.F. Altschul and W. Gish. Local alignment statistics. *Methods Enzymol.*, 266:460–480, 1996.

[3] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[4] S.F. Altschul, T.L. Madden, A.A. Schäffer, 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 Research*, 25:3389–3402, 1997.

[5] A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S.R. Eddy, S. Griffiths-Jones, K.L. Howe, M. Marshall, and E.L.L. Sonnhammer. The Pfam Protein Families Database. *Nucl. Acids Res.*, 30(1):276–280, 2002.

[6] B. Boeckmann, A. Bairoch, R. Apweiler, M.C. Blatter, A. Estreicher, E. Gasteiger, M.J. Martin, K. Michoud, C. O'Donovan, I. Phan, et al. The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003. *Nucl. Acids Res.*, 31(1):365–370, 2003.

[7] B. Brejova, D. G. Brown, and T. Vinar. Optimal spaced seeds for homologous coding regions. *Journal of Bioinformatics and Computational Biology*, 1(4):595–610, 2004.

[8] Brona Brejová, Daniel G. Brown, and Tomás Vinar. Vector seeds: An extension to spaced seeds. *J. Comput. Syst. Sci.*, 70(3):364–380, 2005.

[9] J. Buhler, U. Keich, and Y. Sun. Designing seeds for similarity search in genomic DNA. *J. Comput. Syst. Sci.*, 70(3):342–363, 2005.

[10] M. Cameron, H.E. Williams, and A. Cannane. A deterministic finite automaton for faster protein hit detection in BLAST. *J. Comput. Biol.*, 13(4):965–78, 2006.

[11] S.W. Cheng and Y-F. Xu. Constrained independence system and triangulations of planar point sets. In *Computing and Combinatorics*, pages 41–50, 1995.

[12] Robert D. Finn, John Tate, Jaina Mistry, Penny C. Coggill, Stephen John Sammut, Hans-Rudolf Hotz, Goran Ceric, Kristoffer Forslund, Sean R. Eddy, Erik L. L. Sonnhammer, and Alex Bateman. The pfam protein families database. *Nucl. Acids Res.*, 36(suppl_1):D281–288, 2008.

[13] I. Friedberg, T. Kaplan, and H. Margalit. Evaluation of PSI-BLAST alignment accuracy in comparison to structural alignments. *Protein Science*, 9:2278–2284, 2000.

[14] A. Gambin, S. Lasota, R. Szklarczyk, J. Tiuryn, , and J. Tyszkiewicz. Contextual alignment of biological sequences. In *Proc. ECCB'02, Bioinformatics*, volume 18, pages 116–127. Oxford University Press, 2002.

[15] A. Gambin and J. Tyszkiewicz. Substitution matrices for contextual alignment. In *Journees Ouvertes Biologie Informatique Mathematique*, pages 227–238, 2002.

[16] A. Gambin and P. Wojtalewicz. CTX-BLAST: context sensitive version of protein blast. *Bioinformatics*, 23(13):1686–1688, 2007.

[17] Anna Gambin, Sławomir Lasota, Michał Startek, and Maciej Sykulski. Subset seed extension to protein blast. In *Proc. Intl Conf. on Bioinformatics Models, Methods and Algorithms* Bioinformatics'11, pages 149–158, 2011.

[18] Anna Gambin, Jerzy Tiuryn, and Jerzy Tyszkiewicz. Alignment with context dependent scoring function. *Journal of Computational Biology*, 13(1):81–101, 2006.

[19] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89:10915–10919, 1992.

[20] T. Kahveci and A.K. Singh. An efficient index structure for string databases. *Proceedings of the 27th VLDB*, pages 352–360, 2001.

[21] Derek Kisman, Ming Li, Bin Ma, and Li Wang. tPatternHunter: gapped, fast and sensitive translated homology search. *Bioinformatics (Oxford, England)*, 21(4):542–544, February 2005. PMID: 15374861.

[22] B. Korte and D. Hausmann. An analysis of the greedy heuristic for independence systems. *Ann. Discrete Math.*, 2:65–74, 1978.

[23] G. Kucherov, L. Noé, and M. Roytberg. A unifying framework for seed sensitivity and its application to subset seeds. *Journal of Bioinformatics and Computational Biology*, 4(2):553–570, 2006.

[24] Gregory Kucherov, Laurent Noe, and Mikhail Roytberg. Multiseed lossless filtration. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 2(1):51–61, 2005.

[25] M. Li, B. Ma, D. Kisman, and J. Tromp. PatternHunter II: Highly sensitive and fast homology search. *Journal of Bioinformatics and Computational Biology*, 2(3):417–439, 2004.

[26] T. Li, K. Fan, and W. Wang, J. Wang. Reduction of protein sequence complexity by residue grouping. *Protein Engineering*, 16(5):323–330, 2003.

[27] Franklin Mark Liang. Word hy-phen-a-tion by com-put-er. Technical report, Departament of Computer Science, Stanford University, 1983.

[28] C D Livingstone and G J Barton. Protein sequence alignments: a strategy for the hierarchical an alysis of residue conservation. *Computer Applications in the Biosciences: CABIOS*, 9(6):745–756, December 1993. PMID: 8143162.

[29] Bin Ma, John Tromp, and Ming Li. PatternHunter: faster and more sensitive homology search. *Bioinformatics (Oxford, England)*, 18(3):440–445, March 2002. PMID: 11934743.

[30] Bin Ma and Hongyi Yao. Seed optimization is no easier than optimal golomb ruler design. In *APBC*, pages 133–144, 2008.

[31] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[32] L.R. Murphy, A. Wallqvist, and R.M. Levy. Simplified amino acid alphabets for protein fold recognition and implications for folding. *Protein Engineering*, 13:149–152, 2000.

[33] Van Hoa Nguyen and Dominique Lavenier. Speeding up subset seed algorithm for intensive protein sequence comparison. In *RIVF*, pages 57–63, 2008.

[34] Laurent Noe and Gregory Kucherov. YASS: enhancing the sensitivity of DNA similarity search. *Nucl. Acids Res.*, 33(suppl_2):W540–543, July 2005.

[35] L. Oliveira, A. C. M. Paiva, and G. Vriend. A common motif in g-protein-coupled seven transmembrane helix r eceptors. *Journal of Computer-Aided Molecular Design*, 7(6):649–658, December 1993.

[36] Pierre Peterlongo, Laurent No, Dominique Lavenier, G illes Georges, Julien Jacques, Gregory Kucherov, and Mathieu Giraud. Protein similarity search with subset seeds on a dedicated reco nfigurable hardware. In *Parallel Processing and Applied Mathematics*, pages 1240–1248. Springer, 2008.

[37] Y. Ponty, M. Termier, and A. Denise. GenRGenS: software for generating random genomic sequences and structures, 2006.

[38] B. Rost. Twilight zone of protein sequence alignments. *Protein Engineering Design and Selection*, 12(2):85–94, 1999.

[39] M. Roytberg, A. Gambin, L. Noé, S. Lasota, E. Furletova, E. Szczurek, and G. Kucherov. On subset seeds for protein alignment. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(3):483–494, 2009.

[40] A. S. Shiryev, J. S. Papadopoulos, A. A. S chaffer, and R. Agarwala. Improved BLAST searches using longer words for protein seedin g. *Bioinformatics*, 23(21):2949–2951, November 2007.

[41] T.F. Smith and M.S. Waterman. The identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[42] Yanni Sun and Jeremy Buhler. Designing multiple simultaneous seeds for DNA similarity search. In *RECOMB*, pages 76–84, 2004.

[43] I-Hsuan Yang, Sheng-Ho Wang, Yang-Ho Chen, Pao-Hsian Huang, Liang Ye, Xiaoqiu Huang, and Kun-Mao Chao. Efficient methods for generating optimal single and multiple spaced seeds. In *BIBE '04: Proceedings of the 4th IEEE Symposium on Bioinformatics and Bioengineering*, page 411, Washington, DC, USA, 2004. IEEE Computer Society.