Efficient alternatives to PSI-BLAST

M. STARTEK¹, S. LASOTA¹, M. SYKULSKI¹, A. BULAK¹, L. NOÉ², G. KUCHEROV³, and A. GAMBIN¹, ², ³

¹ Institute of Informatics, University of Warsaw, 2 Banacha St., 02-097 Warszawa, Poland
² LIFL/CNRS/INRIA, Bât. M3, Campus Scientifique, Villeneuve d’Ascq, France
³ 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 the 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-PSI-BLAST tool. The experimental evaluation has been performed demonstrating a significantly higher sensitivity compared to the ordinary PSI-BLAST algorithm.

Key words: PSI BLAST tool, sequence alignment, seeding technique.

1. Introduction

Since the time complexity of the optimal alignment problem is quadratic (e.g., the Smith-Waterman algorithm [1]), 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 [2, 3], remarkably successful in uncovering close homologs. Its extended iterated variant called PSI BLAST [3], 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 [4], while the second one, CTX-PSI BLAST, benefits from the contextual alignment model proposed in [5].

Both tools have been developed on the basis of the source code of the NCBI BLAST tool¹.

1.1. 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 [6, 7], sensitive alignment tools like Pattern-Hunter [8–10] and Yass [11], automaton based theory for modeling and analyzing seeds [4, 12]. The idea of using multiple seeds is also widely recognized [9, 13–15]. In this paper we attempt to achieve similar results for protein homology search, using the approach of [16].

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 [16], with the conclusion that subset seeds [4] 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.

Recently the concept of hash seed for protein homology search has been proposed in [17]. It also applies amino acid grouping (in this case based on BLOSUM matrix) to designing good seeds that allow for a direct hashing scheme. The subset seeds, as considered in this paper, can be regarded as another (potentially more powerful) type of hash seeds.

Moreover, adaptive seeds (being matches that are chosen based on their rareness, instead of using fixed-length matches), has been implemented in the alignment tool called LAST [18], which enables fast and sensitive comparison of large sequences with arbitrarily nonuniform composition.

A fundamental notion in the 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 an exploration of even medium length sequences costly and difficult. A way of approaching the problem is to reduce the alignment alphabet, exploiting similarities among various amino acids [16]. By applying the subset seed the complexity of alignment description may be reduced, while maintaining the biological information content. The idea of subset seeds [4], 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. [19]) 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 [16]. 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 [20, 21]; 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 [4].

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 [22].

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%).

1.2. CTX-PSI BLAST: context-sensitive protein homology search. The contextual (context-sensitive) alignment of biological sequences was proposed in [5], as an extension of the classical alignment where the neighboring residues influence the cost of substitutions. The paper [5] introduced the rudiments of the theory of contextual alignment, and a prototype tool based on the dynamic algorithm of Smith and Waterman [1]. A complement of this work was the computation of biologically significant contextual substitution tables [23], based on BLOSUM family [24].

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 below.

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 [25] a contextual version of the famous BLAST [2] 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 [26].

**Organization of the paper.** In Sec. 2 we present the algorithm to design the seeds used for the protein alignment and then is Sec. 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 Secs. 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 [27].

**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 \( \Sigma \), and whose every internal (non-leaf) node has two children. An example of such a tree is shown in Fig. 1. 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. [21, 20].

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 \( \Sigma_v \). In particular, the root is labeled by the whole set \( \Sigma \). 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 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 \( \alpha \) of nodes such that:

(i) (maximality) \( \alpha \) contains all leaves and

(ii) (downward closedness) whenever \( v \in \alpha \) and \( v' < v \) then \( v' \in \alpha \).

Hence, a single seed letter \( \alpha \) is defined as a lower set of a maximal antichain wrt. the tree ordering. This antichain contains the maximal elements of \( \alpha \) 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 \( \Theta \), 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 \( \alpha \) wrt. the tree ordering form a partition of \( \Sigma \). Thus \( \alpha \) represents naturally an equivalence relation on \( \Sigma \). \( A \) and \( B \) are related iff they belong jointly to some node of \( \alpha \); 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 \( \alpha_1, \alpha_2 \) that are incomparable by inclusion. A seed alphabet is a family \( \mathcal{A} \) of seed letters totally ordered by inclusion: for each \( \alpha_1, \alpha_2 \in \mathcal{A} \), either \( \alpha_1 \subseteq \alpha_2 \) or \( \alpha_2 \subseteq \alpha_1 \). Alphabets with this property are called hierarchical in [16]. We used this assumption
as it leads to a nice mathematical formalization, namely the family of seed alphabets forms a constrained independence system [28, 29]. 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 $\mathcal{A}$ as a finite word over $\mathcal{A}$. A multiple seed is a pair consisting of a seed alphabet and a set of seeds over that alphabet. We say that a seed $s = s_1 s_2 \ldots s_n$ aligns two amino acid sequences $a = a_1 a_2 \ldots a_n$, $b = b_1 b_2 \ldots b_n$, if and only if for all $i \in \{1, 2, \ldots, n\}$, $(a_i, b_i) \in s_i$.

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 [30] using GenRGenS Java tool [31]. The background sensitivity of a seed was computed with the use of Markovian probability transducer as described in [4]. 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 non-trivial task [12, 32, 33]. In the case of hierarchical subset seeds the combinatorial structure of seed alphabets suggests hardness of the optimization problem (see [16] 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 [34]. 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 $\leftarrow$ 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
    - $f \leftarrow fitness^M(M)$;
      - Randomly, based on $f$, choose one of the following:
        - Population $\leftarrow$ Population $\setminus \{M\}$; -- with increasing probability for low values of $f$
        - Population $\leftarrow$ Population $\setminus \{M\} \cup \{Mutate(M)\}$;
        - Population $\leftarrow$ Population $\cup \{Mutate(M)\}$; -- with increasing probability for high values of $f$
  - Population $\leftarrow$ Population
- return the member of Population that maximizes fitness

**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:

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

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
Efficient alternatives to PSI-BLAST

3.1. Hot spot searching 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 [35]. 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 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 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

\[ N = Q \bowtie S := \{ w \mid \text{for some } q \in Q \text{ and } s \in S, s \text{ aligns } q \text{ and } w \}. \]

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 ∈ Σ; 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

Bull. Pol. Ac.: Tech. 60(3) 2012

499

Unauthenticated Download Date | 5/28/18 10:30 PM
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 [25]. 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 $i$th 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 $i$th 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;
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. 2. 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. 2, the left context is thus chosen to be $A$ and the right one is $E$.

\(^2\)http://ctx-psi-blast.sourceforge.net/index.html

\[
P: \ldots \ 6 \ 7 \ 8 \ 9 \ 10 \ \ldots
\]
\[
\ldots \ 6 \ 7 \ D \ 9 \ 10 \ \ldots
\]
\[
S: \ldots \ C \ A \ D \ E \ A \ \ldots
\]

Fig. 2. A fragment of alignment of a subject sequence $S$ against 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 [36], statistics of optimal non-gapped contextual alignment follows the same extreme value distribution as in the non-contextual case [37]. Hence we have decided to adopt the island method [38], used also by [25], for estimation of the parameters $K$ and $\lambda$ required for E-value calculation for the alignment score $S$ of two sequences of length $m$ and $n$, respectively:

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

Using the method suggested in [38], 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 [39, 40].

A protein family, exhibiting a 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 wide-spread 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
Efficient alternatives to PSI-BLAST

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.

5.1. Comparing efficiency. Figure 3 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).

![Fig. 3. Symmetric difference between outputs of BLAST and SeedBLAST](image)

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 [22] – 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 Fig. 4.

![Fig. 4. Domain coverage by alignments found by BLAST and SeedBLAST](image)

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.

5.2. 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:

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

(cf. Sec. 1) that seems to suit best to this purpose: the parameter c corresponds 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.

### Table 1

<table>
<thead>
<tr>
<th>f</th>
<th>c</th>
<th>sens^B(M)</th>
<th>sens^F(M)</th>
<th>NCBI-BLAST (sec.)</th>
<th>Seed-BLAST (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.000306</td>
<td>0.00030564</td>
<td>1.20</td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0.002195</td>
<td>0.00211789</td>
<td>3.32</td>
<td>1.40</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.012276</td>
<td>0.01156471</td>
<td>12.17</td>
<td>4.74</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.026406</td>
<td>0.02622019</td>
<td>23.84</td>
<td>12.40</td>
<td></td>
</tr>
</tbody>
</table>

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.
Comparison with PSI-BLAST. One can see the close similarity between the seed approach and position specific scoring matrices used to improve homology 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 performance 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. From the analysis of found alignments grouped by logarithm of their E-value 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 [41].

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 [42, 43].

Alphabet used for experiments. The alphabet contains 8 letters. We provide its representation as a hierarchy of nested equivalence relations over amino acid alphabets.

\[
\begin{align*}
&{\text{CFYWMLIVGPATSNHQEDRK}} \\
&5: \{RK, Q, E, D, IVL, F,M, A,S,Y,T,G,N,H,C,P,W\} \\
\end{align*}
\]

5.3. Usage. To run compiled file blast in SeedBLAST mode please provide the parameter: -pblastp - seedfa.

There is one more command line parameter, options are read from a config file.

\[-x \text{ File to read SeedFA extension configuration from.}\]

(default ./seedfa.cfg) [String]

\[-M \text{ --seed-file-homit }\]

shall first line in the seed file be omitted flag

\[-S \text{ --seed-file }\]

name of a file with the alignment seed

\[-A \text{ --alignalph-file }\]

name of a file with the alignment alphabet

\[-K \text{ --seed-file-kind }\]

parameter threshold for log(P/Q) based choice:: 2 - seedCover, 5 - Pprob, 6 - Qprob; see manual for more info

Configuration:

\[-A \text{ --alignalph-file }\] name of a file with the alignment alphabet

\[-S \text{ --seed-file }\] name of a file with the alignment seed

\[-M \text{ --seed-file-homit }\] shall first line in the seed file be omitted flag

6. Evaluation of CTX-PSI-BLAST

The experimental evaluation of the CTX-PSI-BLAST algorithm relies on a methodology applied in [26] for PSI-BLAST. The authors of [26] selected 123 pairs of evolutionary distinc 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 [26] 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

efficient alternatives to PSI-BLAST

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 [26], 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 [26], 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 [26] – 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:

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

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 [26] 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 [26]. Figure 5 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.

![Fig. 5. The iteration that found a pair. 0 denotes the initial iteration that runs without a PSSM. PSI-BLAST (top) compared to CTX-PSI BLAST (bottom)](image)

Other interesting phenomenon can be observed: 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 [25].

### 6.3. Usage

The following command runs the standard version of PSI-BLAST:

```
./psiblast -in query -db database -num_iterations iter \ -out output_file
```

where query is the input query sequence in FASTA format; database is the name of database to scan for significant alignments; iter is the number of iterations to be run (PSI-BLAST may run less iterations then specified if a search converges); output_file is a file to which the program output is written.
will be written (if a specified file exists, it will be overwritten). The contextual version of PSI-BLAST requires two additional parameters: ctx_matrix defines the contextual substitution table (e.g., CTX-BLOSUM62, the contextual counterpart of BLOSUM62); and ctx_stat defines the file containing the parameters for statistical significance calculation in the following order: $\alpha$, $\beta$, $\lambda$, and $K$. Parameters must be separated with a single space. The exemplary usage of CTX-PSI-BLAST on the NR database performing 6 iterations is the following:

```
./psiblast -in some.seq -db nr -num_iteration 6 -out some.out \-ctx_matrix blosum62.nrm
-ctx_stat stat.txt
```

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 [44], 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 [45]. 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 [36]) 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 the Polish Ministry of Science and Higher Education grant nr N N301 013436.

REFERENCES

Efficient alternatives to PSI-BLAST


[33] B. Ma and H. Yao, “Seed optimization is no easier than optimal golomb ruler design”, APBC 1, 133–144 (2008).


