HAL
open science

Efficient seeding techniques for protein similarity search

Mihkail Roytberg, Anna Gambin, Laurent Noé, Slawomir Lasota, Eugenia Furletova, Ewa Szczurek, Gregory Kucherov

To cite this version:

Mihkail Roytberg, Anna Gambin, Laurent Noé, Slawomir Lasota, Eugenia Furletova, et al.. Efficient seeding techniques for protein similarity search. Proceedings of the 2nd International Conference BIRD, Jul 2008, Vienna, Austria. pp.466-478, 10.1007/978-3-540-70600-7 . inria-00335564

HAL Id: inria-00335564

https://inria.hal.science/inria-00335564

Submitted on 29 Oct 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Efficient seeding techniques for protein similarity search

Mikhail Roytberg[1], Anna Gambin[2], Laurent Noé[3], Sławomir Lasota[2], Eugenia Furletova[1], Ewa Szczurek[4], and Gregory Kucherov[3]

[1] Institute of Mathematical Problems in Biology, Pushchino, Moscow Region, 142290, Russia, mroytberg@mail.ru,furletova@impb.psn.ru
[2] Institute of Informatics, Warsaw University, Banacha 2, 02-097, Poland, {aniag|S.Lasota}@mimuw.edu.pl
[3] LIFL/CNRS/INRIA, Bât. M3, Campus Scientifique, 59655 Villeneuve d'Ascq Cédex, France, {Gregory.Kucherov|Laurent.Noe}@lifl.fr
[4] Max Planck Institute for Molecular Genetics, Computational Molecular Biology, Ihnestr. 73, 14195 Berlin, Germany, ewa.szczurek@molgen.mpg.de

Abstract. We apply the concept of *subset seeds* proposed in [1] to similarity search in protein sequences. The main question studied is the design of efficient *seed alphabets* to construct seeds with optimal sensitivity/selectivity trade-offs. We propose several different design methods and use them to construct several alphabets. We then perform an analysis of seeds built over those alphabet and compare them with the standard Blastp seeding method [2,3], as well as with the family of vector seeds proposed in [4]. While the formalism of subset seed is less expressive (but less costly to implement) than the accumulative principle used in Blastp and vector seeds, our seeds show a similar or even better performance than Blastp on Bernoulli models of proteins compatible with the common BLOSUM62 matrix.

1 Introduction

Similarity search in protein sequences is probably the most classical bioinformatics problem, and a commonly used algorithmic solution is implemented in the ubiquitous Blast software [2,3]. On the other hand, similarity search algorithms for nucleotide sequences (DNA, RNA) underwent several years ago a significant improvement due to the idea of *spaced seeds* and its various generalizations [5,6,7,8,9,10,11]. This development, however, has little affected protein sequence comparison, although improving the speed/precision trade-off for protein search would be of great value for numerous bioinformatics projects. Due to a bigger alphabet size, protein seeds are much shorter (typically 2-5 letters instead of 10-20 letters in the DNA case) and also letter identity is much less relevant in defining hits than in the DNA case. For these reasons, the spaced seeds technique might seem not to apply directly to protein sequence comparison.

Recall that Blast applies quite different approaches to protein and DNA sequences to define a hit. In the DNA case, a hit is defined as a short pattern

of identically matching nucleotides whereas in the protein case, a hit is defined through an *accumulative* contribution of a few amino acid matches (not necessarily identities) using a given *scoring matrix*. Defining a hit through an additive contribution of several positions is captured by a general formalism of *vector seeds* proposed in [7]. On the other hand, it has been understood [6,12,13,14,15] that using simultaneously a *family* of seeds instead of a single seed can further improve the sensitivity/selectivity ratio. Papers [4,16] both propose solutions using a family of vector seeds to surpass the performance of Blast.

However, using the principle of accumulative score over several adjacent positions has an algorithmic cost. Defining a hit through a pattern of exact letter matches allows for a *direct hashing* scheme, where each key of the query sequence is associated with a *unique* hash table entry storing positions of the subject sequence (database) where the key can hit. On the other hand, defining a hit through an accumulative contribution of several positions leads to an additional pre-computed table storing, for each key, its *neighborhood* i.e., the list of subject keys (or corresponding hash table entries) with which it can form a hit. For example, in a standard Blastp setting (Blosum62 scoring matrix with threshold 11 for accumulative score of 3 contiguous positions) a 3-letter key hits on average 19.34 distinct keys, i.e. requires that many accesses to the hash table. For the family of vector seeds from [4] with an equivalent selectivity level (score 18), a (here 4-letter) key hits on average 15.99 keys. For some applications, for example in setting large-scale protein comparisons on a specialized computer architecture (see e.g. [17]) one might need to minimize the number of hash table accesses, and therefore to use another seeding formalism.

In [1], we proposed a new concept of *subset seeds* that can be viewed as an intermediate between ordinary spaced seeds and vector seeds: subset seeds allow one to distinguish between different types of mismatches (or matches) but still treat seed positions independently rather than cumulatively. Distinguishing different mismatches is not done by scoring them, but by extending the seed alphabet such that each seed letter specifies different sets of mismatches. For example, in the DNA case it is beneficial to distinguish between transition mutations ($\mathtt{A} \leftrightarrow \mathtt{G}$, $\mathtt{C} \leftrightarrow \mathtt{T}$) and others (transversions) [18].

Since the protein alphabet is much larger than the one of DNA, subset seeds provide a very attractive seeding option for protein alignment. The present study is then motivated by following general questions: *how far can we go with subset seeds applied to protein sequences? Can we reach the performance of* Blast *seeds? the one of vector seeds? or maybe even outperform them? ...*

In the paradigm of subset seeds, each seed letter specifies a set of amino acid pairs matched by this letter. Therefore, a crucial question is the design of an appropriate *seed alphabet*, which is one of the main problems we study in this paper. In Section 2, we introduce some probabilistic notions we need to reason about seed efficiency. Section 3 introduces the first simple approach to design a seed alphabet, which, however, does not lead to so-called *transitive* seeds, useful in practice. Section 4 presents three different approaches to designing transitive seed alphabets, based on a pre-defined (Section 4.1) or newly designed (Sec-

tion 4.2) hierarchical clustering of amino acids, as well as on a non-hierarchical clustering (Section 4.3). Section 5 describes comparative experiments made with the designed seeds on probabilistic models.

2 Preliminaries

Throughout the paper, $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{V}, \mathtt{W}, \mathtt{Y}\}$ denotes the alphabet of amino acids.

In most general terms, a *(subset) seed letter* α is defined as any symmetric and reflexive binary relation on Σ. Let $\mathcal{B}$ be a *seed alphabet*, i.e. a collection of subset seed letters. Then a *subset seed* $\pi = \alpha_1 \dots \alpha_k$ is a word over $\mathcal{B}$. π defines a symmetric and reflexive binary relation on words of Σ^k (called *keys*): for $s_1, s_2 \in \Sigma^k$, $s_1 \sim_\pi s_2$ iff $\forall i \in [1..k]$, we have $\langle s_1[i], s_2[i] \rangle \in \alpha_i$.

For practical reasons, we would like seed letters to define a *transitive* relation, in addition. This induces an equivalence relation on keys, which is very convenient and allows for an efficient indexing scheme (see Introduction). In this paper, we will be mainly interested in transitive seed letters, but we will also study the non-transitive case in order to see how restrictive the transitivity condition is.

The quality of a seed letter or of a seed is characterized by two main parameters: *sensitivity* and *selectivity*. They are defined through background and foreground probabilistic models of protein alignments. Foreground probabilities are assumed to represent the distribution of amino acids matches in proteins of interest, when two homologous proteins are aligned together. Background probabilities, on the other hand, represent the distribution of amino acid matches in *random alignments*, when two proteins are randomly aligned together.

In this paper, we restrict ourselves to Bernoulli models of proteins and protein alignments, although some of the results we will present can be extended to Markov models.

Assume that we are given background probabilities $\{b_1, \dots, b_{20}\}$ of amino acids in protein sequences under interest. The *background probability* of a seed letter α is defined by $b(\alpha) = \sum_{(a_i,a_j)\in\alpha} b_i b_j$. The *selectivity* of α is $1 - b(\alpha)$ and the *weight* of α is defined by

$$w(\alpha) = \frac{\log b(\alpha)}{\log b(\#)}, \tag{1}$$

where $\# = \{\langle a, a\rangle | a \in \Sigma\}$ is the "identity" seed letter. For a seed $\pi = \alpha_1 \dots \alpha_k$, the background probability of π is $b(\pi) = \prod_{i=1}^{k} b(\alpha_i)$, the selectivity of π is $1 - b(\pi)$ and the weight of π is $w(\pi) = \log_{b(\#)} b(\pi) = \sum_{i=1}^{k} w(\alpha_i)$. Note that the weight here generalizes the weight of of classical spaced seeds [19] defined as the number of "identity" letters it contains.

Let f_{ij} be the probability to see a pair $\langle a_i, a_j \rangle$ aligned in a target alignment. The *foreground probability* of a seed letter α is defined by $f(\alpha) = \sum_{(a_i,a_j)\in\alpha} f_{ij}$. The *sensitivity* of a seed π is defined as the probability to hit a random target

alignment[5]. Assume that target alignments are specified by a length N. Then the sensitivity of a seed $\pi = \alpha_1 \dots \alpha_k$ is the probability that a randomly drawn gapless alignment (i.e. string of pairs $\langle a_i, a_j \rangle$) of length N contains a fragment of length k which is matched by π. In [1] we proposed a general algorithm to efficiently compute the seed sensitivity for a broad class of target alignment models. This algorithm will be used in the experimental part of this work.

The general problem of seed design is to obtain seeds with good sensitivity/selectivity trade-off. Even within a fixed seed formalism, the quality of a seed is dependent on the chosen selectivity value. This is why we will always be interested in computing efficient seeds for a large range of selectivity levels.

3 Dominating seed letters

Our main question is how to choose seed letters that form good seeds? Intuitively, "good letters" are those that best distinguish foreground and background letter alignments.

For each letter α, consider its foreground and background probabilities $f(\alpha)$ and $b(\alpha)$ respectively. Intuitively, we would like to have letters α with large $f(\alpha)$ and small $b(\alpha)$. A letter α is said to *dominate* a letter β if $f(\alpha) \geq f(\beta)$ and $b(\alpha) \leq b(\beta)$. Observe that in this case, β can be removed from consideration, as it can be always advantageously replaced by α.

Consider all amino acid pairs (a_i, a_j) ordered by descending *likelihood ratio* $f_{ij}/b_i b_j$. Consider the set of pairs (a_i, a_j) such that $f_{ij}/b_i b_j > s$ for some threshold value s. Then one can show that this set forms a letter that cannot be dominated by any other letter[6] (proof omitted). This observation leads to defining seed letters that consist of those pairs (a_i, a_j) for which the ratio $f_{ij}/b_i b_j$ is above a given threshold.

Resulting alphabet We computed the likelihood ratio for all amino acid pairs, based on practical values of background and foreground probabilities computed in accordance with the BLOSUM62 matrix (see Section 5.1). Not surprisingly, amino acid identities (pairs $\langle a, a \rangle$) have highest likelihood scores varying from 38.11 for tryptophan down to 3.69 for valine. Among distinct pairs, only 25 have a score greater than 1 (Figure 1). A quick analysis shows that those do not form transitive relations, and therefore do not verify the transitivity requirement. The alphabet containing those 25 pairs is denoted `Non-transitive`. It will be used in the experimental part of the paper (Section 5) in order to study how restrictive

[5] Note that our definitions of sensitivity and selectivity are not symmetric: sensitivity is defined with respect to the entire alignment and selectivity with respect to a single alignment position. These definitions capture better the intended parameters we want to measure. However, selectivity could also be defined with respect to the entire alignment. We could suggest the term *specificity* for this latter definition.

[6] It is interesting to point out the relationship to the well-known Neyman-Pearson lemma which is a more general formulation of this statement.

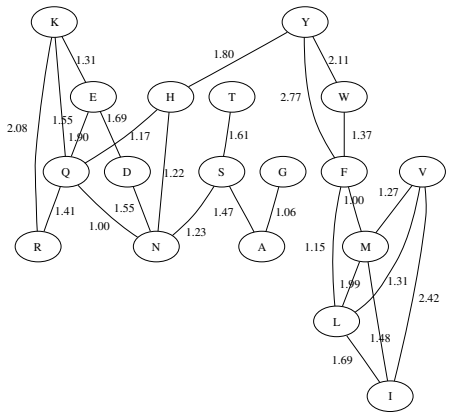


Fig. 1. Alphabet `Non-transitive`: amino acid pairs with likelihood ratio > 1

is the requirement of transitive letters, i.e. how much better are general seeds than those obtained with the restriction of transitivity.

4 Transitive seed alphabets

In the case of transitive seed alphabets, every letter $\alpha \in \mathcal{B}$ is a partition of the amino acid alphabet Σ. In other words, the binary relation associated with each letter (cf Section 2) is an equivalence relation. Transitive alphabets represent the practical case when each amino acid is uniquely mapped to its equivalence class. This, in turn, allows for an efficient hashing scheme during the stage of seed search, when different entries of the hash table index non-intersecting subsets of keys.

In Sections 4.1,4.2, we explore transitive seed alphabets verifying an additional condition: for any two seed letters $\alpha_1, \alpha_2 \in \mathcal{B}$ corresponding to partitions $P_{\alpha_1}, P_{\alpha_2}$ respectively, one of $P_{\alpha_1}, P_{\alpha_2}$ is a refinement of the other. Formally, for any $\alpha_1, \alpha_2 \in \mathcal{B}$,

$$\text{either every set } \sigma \in P_{\alpha_1} \text{ is a subset of some } \delta \in P_{\alpha_2}, \text{ or vice versa.} \quad (2)$$

The purpose of the above requirement is to define seed letters using a biologically significant hierarchical clustering of amino acids represented by a tree. In Section 4.1, we will use a pre-defined hierarchical clustering to design efficient seed alphabets. Then in Section 4.2, we construct our own clustering based on appropriate background and foreground models of amino acids distribution. Finally, in Section 4.3 we lift condition (2) and study "non-hierarchical" seed alphabets.

4.1 Transitive alphabets based on a pre-defined clustering

Assume we have a biologically significant hierarchical clustering tree which is a rooted binary tree T with 20 leaves labelled by amino acids. Such trees have been proposed in [20,21], based on different similarity relations. The hierarchical tree derived from [20] is shown on Figure 2. The tree, obtained with a purely bioinfor-

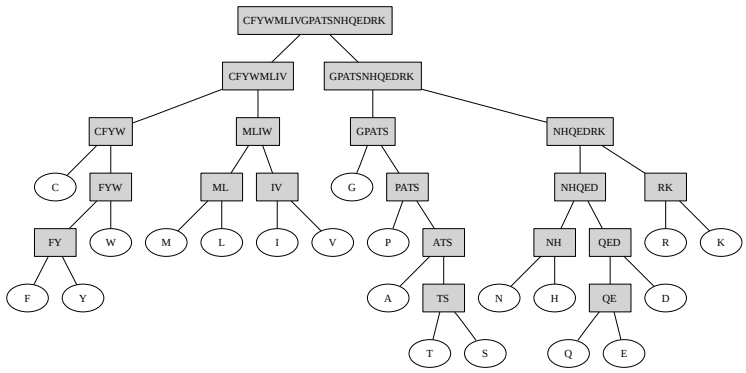


Fig. 2. Hierarchical tree derived from [20].

matics analysis, groups together amino acids with similar biochemical properties, such as hydrophobic amino acids `L,M,I,V`, hydrophobic aromatic amino acids `F,Y,W`, alcohols `S,T`, or charged/polar amino acids `E,D,N,Q`. A similar grouping is obtained in [21].

A *seed letter* is defined here as a subset α of nodes of T such that

(i) α contains all leaves,
(ii) for a node v, if $v \in \alpha$, then all descendants of v belong to α too.

In other words, a seed letter can be thought of as a "horizontal cut" of the tree. For example, for the tree of Figure 2 there are 1597 different seed letters. Seed letters are naturally ordered by inclusion. The smallest one is the "identity" seed letter #, containing only the leaves. The largest one is the "joker" seed letter _, containing all the nodes of T. One particular seed letter is obtained by removing from _ the root node. We denote it by @.

Observe that each seed letter α represents naturally an equivalence relation on Σ: a_i and a_j are related iff their common ancestor belongs to α. It is identity relation in case of # and full relation in case of _.

Following condition (2), a *hierarchical seed alphabet* is a family $\mathcal{B}$ of seed letters such that

$$\text{for every } \alpha_1, \alpha_2 \in \mathcal{B}, \text{ either } \alpha_1 \subseteq \alpha_2 \text{ or } \alpha_2 \subseteq \alpha_1. \tag{3}$$

Hence, a seed alphabet is a chain in the inclusion ordering of seed letters. Let us analyze what are the maximal seed alphabets wrt. inclusion. Clearly each

maximal seed alphabet $\mathcal{B}$ always contains the smallest and the largest seed letters # and _. Interestingly, each maximal $\mathcal{B}$ contains also @, as @ is comparable (by inclusion) to any other seed letter.

It can be shown that any maximal seed alphabet contains exactly 20 letters that can be obtained by a stepwise merging of two subtrees rooted at immediate descendants of some node v into the subtree rooted at v. Therefore, since a binary tree with n leaves contains $n-1$ internal nodes, a maximal seed alphabet contains precisely 20 letters and can be specified by a permutation of internal nodes in tree T.

Resulting alphabet Figure 3 shows alphabet `Transitive-predefined` designed through the approach of this Section. The alphabet has been designed from the tree of Figure 2. Each line corresponds to a letter (amino acid partition). Among the alphabets obtained with different parameter values, alphabet `Transitive-predefined` produced better seeds and will be used in the experimental part of this work (Section 5).

$$\{CFYWMLIVGPATSNHQEDRK\}$$
$$\{CFYWMLIV\}\ \{GPATSNHQEDRK\}$$
$$\{CFYWMLIV\}\ \{GPATS\}\ \{NHQEDRK\}$$
$$\{CFYW\}\ \{MLIV\}\ \{GPATS\}\ \{NHQEDRK\}$$
$$\{CFYW\}\ \{MLIV\}\ \{G\}\ \{PATS\}\ \{NHQEDRK\}$$
$$\{C\}\ \{FYW\}\ \{MLIV\}\ \{G\}\ \{PATS\}\ \{NHQEDRK\}$$
$$\{C\}\ \{FYW\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{ATS\}\ \{NHQEDRK\}$$
$$\{C\}\ \{FY\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{ATS\}\ \{NHQEDRK\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{ATS\}\ \{NHQEDRK\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{A\}\ \{TS\}\ \{NHQEDRK\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{NHQEDRK\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{NHQED\}\ \{RK\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{NHQED\}\ \{R\}\ \{K\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{NH\}\ \{QED\}\ \{R\}\ \{K\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{N\}\ \{H\}\ \{QED\}\ \{R\}\ \{K\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{N\}\ \{H\}\ \{QE\}\ \{D\}\ \{R\}\ \{K\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{MLIV\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{N\}\ \{H\}\ \{Q\}\ \{E\}\ \{D\}\ \{R\}\ \{K\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{ML\}\ \{IV\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{N\}\ \{H\}\ \{Q\}\ \{E\}\ \{D\}\ \{R\}\ \{K\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{M\}\ \{L\}\ \{IV\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{N\}\ \{H\}\ \{Q\}\ \{E\}\ \{D\}\ \{R\}\ \{K\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{M\}\ \{L\}\ \{I\}\ \{V\}\ \{G\}\ \{P\}\ \{A\}\ \{T\}\ \{S\}\ \{N\}\ \{H\}\ \{Q\}\ \{E\}\ \{D\}\ \{R\}\ \{K\}$$

Fig. 3. Alphabet `Transitive-predefined` designed using the tree of Figure 2. Each line corresponds to a seed letter (amino acid partition)

4.2 Transitive alphabets using an *ab initio* clustering method

Hierarchical clustering of amino acids A prerequisite to the approach of Section 4.1 is a given tree describing a hierarchical clustering of amino acid based on some similarity measure. In this section, we describe an *ab initio* approach that constructs a hierarchical clustering of amino acids from scratch, using a likelihood measure.

As in Section 4, our goal here is to construct a family of seed letters verifying (3). This family will be obtained with a simple greedy neighbor-joining clustering algorithm, starting with the family of twenty amino acid singletons.

We start with the partition of amino acids into 20 singletons. This partition corresponds to the # letter. For a current partition $P = \{C_1, \ldots, C_n\}$, iteratively apply the following procedure.

1 For each pair of sets C_k, C_ℓ,
  1.1 consider the set $Bridge(C_k, C_\ell) = \{(a_i, a_j) | a_i \in C_k,\ a_j \in C_\ell\}$.
  1.2 compute $ForeProb(k, \ell) = \sum\{f_{ij} | a_i \in C_k,\ a_j \in C_\ell\}$
  and $BackProb(k, \ell) = \sum\{b_i b_j | a_i \in C_k,\ a_j \in C_\ell\}$,
  1.3 compute $L(k, \ell) = ForeProb(k, \ell)/BackProb(k, \ell)$
2 Find the pair of sets (C_k, C_ℓ) yielding the maximal $L(k, \ell)$,
3 Merge C_k and C_ℓ into a new set, obtaining a new partition.

The rationale behind this simple procedure is that those two sets of amino acid are merged together which produce the maximal increment in the likelihood $f(\alpha)/b(\alpha)$. An alternative method, when the likelihood of the whole resulting set is maximized, yields biased results, as sets with a high likelihood tend to "absorb" other sets.

Resulting alphabet An alphabet, called `Transitive-ab-initio`, obtained with this greedy neighbor-joining approach is given in Figure 4. It will be used in experiments presented later in Section 5.

$$\{CFYWHMLIVPGQERKNDATS\}$$
$$\{CFYWHMLIV\}\ \{PGQERKNDATS\}$$
$$\{C\}\ \{FYWHMLIV\}\ \{PGQERKNDATS\}$$
$$\{C\}\ \{FYWHMLIV\}\ \{P\}\ \{GQERKNDATS\}$$
$$\{C\}\ \{FYWH\}\ \{MLIV\}\ \{P\}\ \{GQERKNDATS\}$$
$$\{C\}\ \{FYWH\}\ \{MLIV\}\ \{P\}\ \{GATS\}\ \{QERKND\}$$
$$\{C\}\ \{FYWH\}\ \{MLIV\}\ \{P\}\ \{G\}\ \{ATS\}\ \{QERKND\}$$
$$\{C\}\ \{FYWH\}\ \{MLIV\}\ \{P\}\ \{G\}\ \{ATS\}\ \{QERK\}\ \{ND\}$$
$$\{C\}\ \{FYW\}\ \{H\}\ \{MLIV\}\ \{P\}\ \{G\}\ \{ATS\}\ \{QERK\}\ \{ND\}$$
$$\{C\}\ \{FYW\}\ \{H\}\ \{MLIV\}\ \{P\}\ \{G\}\ \{A\}\ \{TS\}\ \{QERK\}\ \{ND\}$$
$$\{C\}\ \{FYW\}\ \{H\}\ \{MLIV\}\ \{P\}\ \{G\}\ \{A\}\ \{TS\}\ \{QE\}\ \{RK\}\ \{ND\}$$
$$\{C\}\ \{FYW\}\ \{H\}\ \{ML\}\ \{IV\}\ \{P\}\ \{G\}\ \{A\}\ \{TS\}\ \{QE\}\ \{RK\}\ \{ND\}$$
$$\{C\}\ \{FYW\}\ \{H\}\ \{ML\}\ \{IV\}\ \{P\}\ \{G\}\ \{A\}\ \{TS\}\ \{QE\}\ \{RK\}\ \{N\}\ \{D\}$$
$$\{C\}\ \{FYW\}\ \{H\}\ \{ML\}\ \{IV\}\ \{P\}\ \{G\}\ \{A\}\ \{T\}\ \{S\}\ \{QE\}\ \{RK\}\ \{N\}\ \{D\}$$
$$\{C\}\ \{FY\}\ \{W\}\ \{H\}\ \{ML\}\ \{IV\}\ \{P\}\ \{G\}\ \{A\}\ \{T\}\ \{S\}\ \{QE\}\ \{RK\}\ \{N\}\ \{D\}$$
$$\{C\}\ \{FY\}\ \{W\}\ \{H\}\ \{ML\}\ \{IV\}\ \{P\}\ \{G\}\ \{A\}\ \{T\}\ \{S\}\ \{Q\}\ \{E\}\ \{RK\}\ \{N\}\ \{D\}$$
$$\{C\}\ \{FY\}\ \{W\}\ \{H\}\ \{M\}\ \{L\}\ \{IV\}\ \{P\}\ \{G\}\ \{A\}\ \{T\}\ \{S\}\ \{Q\}\ \{E\}\ \{RK\}\ \{N\}\ \{D\}$$
$$\{C\}\ \{FY\}\ \{W\}\ \{H\}\ \{M\}\ \{L\}\ \{I\}\ \{V\}\ \{P\}\ \{G\}\ \{A\}\ \{T\}\ \{S\}\ \{Q\}\ \{E\}\ \{RK\}\ \{N\}\ \{D\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{H\}\ \{M\}\ \{L\}\ \{I\}\ \{V\}\ \{P\}\ \{G\}\ \{A\}\ \{T\}\ \{S\}\ \{Q\}\ \{E\}\ \{RK\}\ \{N\}\ \{D\}$$
$$\{C\}\ \{F\}\ \{Y\}\ \{W\}\ \{H\}\ \{M\}\ \{L\}\ \{I\}\ \{V\}\ \{P\}\ \{G\}\ \{A\}\ \{T\}\ \{S\}\ \{Q\}\ \{E\}\ \{R\}\ \{K\}\ \{N\}\ \{D\}$$

Fig. 4. Alphabet `Transitive-ab-initio` obtained with the method of Section 4.2

4.3 Non-hierarchical alphabets

Previous approaches (Sections 4.1 and 4.2) were based on requirement (3) specifying that letters of the seed alphabet should be embedded one into another to form a "nested" hierarchy. This requirement is biologically motivated and, on the other hand, computationally useful as it reduces considerably the space of possible letters. However, this requirement is not necessary to implement the direct indexing (see Introduction). Therefore, we also designed non-hierarchical alphabets in order to compare them to hierarchical ones. We used the following heuristic consisting in generating first a large number of seed candidates, and selecting the ones with (1) high likelihood ratio, (2) a range of different weights.

Resulting alphabet An alphabet obtained with the above heuristic, called `Non-tree-transitive`, is shown in Figure 5. This alphabet will be used in the experiments reported in Section 5.

$$\{ARNDCQEGHILMKFPSTWYV\}$$
$$\{ARNDQEGHILMKFPSTWYV\}\ \{C\}$$
$$\{ARNDCQEHILMKFPSTWYV\}\ \{G\}$$
$$\{ARNDQEHILMKFSTYV\}\ \{CGPW\}$$
$$\{ARCQEHILMKFSTYV\}\ \{NDGPW\}$$
$$\{ARNDCQEGHKPST\}\ \{ILMFWYV\}$$
$$\{ARNDQEGHKST\}\ \{CILMFWYV\}\ \{P\}$$
$$\{ARNDQEHKPST\}\ \{CW\}\ \{G\}\ \{ILMFYV\}$$
$$\{ARNDQEKST\}\ \{CP\}\ \{GHW\}\ \{ILMFYV\}$$
$$\{AGPST\}\ \{RNDQEHK\}\ \{C\}\ \{ILMFWYV\}$$
$$\{APST\}\ \{RNDQEHK\}\ \{CW\}\ \{G\}\ \{ILMFYV\}$$
$$\{AGST\}\ \{RNDQEK\}\ \{C\}\ \{HFWY\}\ \{ILMV\}\ \{P\}$$
$$\{AST\}\ \{RNDQEK\}\ \{CH\}\ \{G\}\ \{ILMV\}\ \{FWY\}\ \{P\}$$
$$\{AST\}\ \{RQEHK\}\ \{ND\}\ \{CP\}\ \{G\}\ \{ILMV\}\ \{FWY\}$$
$$\{AST\}\ \{RQK\}\ \{NH\}\ \{DE\}\ \{C\}\ \{G\}\ \{ILMV\}\ \{FWY\}\ \{P\}$$
$$\{A\}\ \{RQK\}\ \{N\}\ \{DE\}\ \{C\}\ \{G\}\ \{H\}\ \{ILMV\}\ \{FY\}\ \{P\}\ \{ST\}\ \{W\}$$
$$\{A\}\ \{RK\}\ \{N\}\ \{DE\}\ \{C\}\ \{QH\}\ \{G\}\ \{ILV\}\ \{M\}\ \{FY\}\ \{P\}\ \{ST\}\ \{W\}$$
$$\{A\}\ \{RQK\}\ \{ND\}\ \{C\}\ \{E\}\ \{G\}\ \{H\}\ \{IV\}\ \{LM\}\ \{FWY\}\ \{P\}\ \{ST\}$$
$$\{A\}\ \{RK\}\ \{ND\}\ \{C\}\ \{Q\}\ \{E\}\ \{G\}\ \{H\}\ \{IV\}\ \{LM\}\ \{FWY\}\ \{P\}\ \{S\}\ \{T\}$$
$$\{A\}\ \{RK\}\ \{N\}\ \{D\}\ \{C\}\ \{Q\}\ \{E\}\ \{G\}\ \{H\}\ \{IV\}\ \{L\}\ \{M\}\ \{FY\}\ \{P\}\ \{S\}\ \{T\}\ \{W\}$$
$$\{A\}\ \{R\}\ \{N\}\ \{D\}\ \{C\}\ \{QE\}\ \{G\}\ \{H\}\ \{I\}\ \{L\}\ \{K\}\ \{M\}\ \{FWY\}\ \{P\}\ \{S\}\ \{T\}\ \{V\}$$
$$\{A\}\ \{R\}\ \{N\}\ \{D\}\ \{C\}\ \{Q\}\ \{E\}\ \{G\}\ \{H\}\ \{I\}\ \{L\}\ \{K\}\ \{M\}\ \{F\}\ \{P\}\ \{S\}\ \{T\}\ \{W\}\ \{V\}$$

Fig. 5. Non-hierarchical alphabet `Non-tree-transitive`.

5 Experiments

This section describes the experiments we made to test the efficiency of seeds we designed with different methods of previous sections. Sections 5.1-5.3 describe the experimental protocol, from the assignment of background and foreground probabilities, to the seed design. In Section 5.4, we analyze the power of different seed models proposed in Sections 3-4 with respect to probabilistic models.

5.1 Probability assignment and alphabet generation

First of all, we derived probabilistic models in accordance with the BLOSUM62 data from the original paper [22]. We obtained the BLOCKS database (version 5) [23] and the software of [22] to infer Bernoulli probabilities for the background and foreground alignment models. These probabilities have been used throughout the whole pipeline of experiments.

Different seed alphabets have then been generated by the methods presented in Section 3 (alphabet `Non-transitive`), Section 4.1 (alphabet `Transitive-predefined`), Section 4.2 (alphabet `Transitive-ab-initio`) and Section 4.3 (alphabet `Non-tree-transitive`).

5.2 Seed design

To each alphabet, we applied a seed design procedure that we briefly describe now. Since each seed (or seed family) is characterized by two parameters – sensitivity and selectivity – it can be associated with a point on a 2-dimensional plot.

Best seeds are then defined to be those which belong to the *Pareto* set among all seeds, i.e. those than cannot be strictly improved by increasing sensitivity, selectivity, or both.

For different selectivity levels, we designed good seed families containing one to six individual seeds, among which the best family was selected. In each seed family, each individual seed has been assumed to have approximately the same weight, within 5% tolerance. This requirement is natural as in the case of divergent weights, seeds with lower weight would dominantly affect the performance. In practice, having individual seeds of similar weight allows an efficient parallel implementation (see e.g. [17]).

Estimation of sensitivity of individual seeds or seed families has been done with the algorithm described in [1] and implemented in the IEDERA software, available at `http://bioinfo.lifl.fr/yass/iedera.php`. The selectivity of an individual seed has been computed according to the definition (Section 2). For a seed family, its selectivity has been lower-estimated by summing the background probabilities of individual seeds.

Seed family design has been done using a hill climbing heuristics (see [24,25,11]) alternating seed generation and seed estimation steps. All experiments were conducted for alignment lengths 16 and 32.

5.3 BLASTP and the vector seed family from [4]

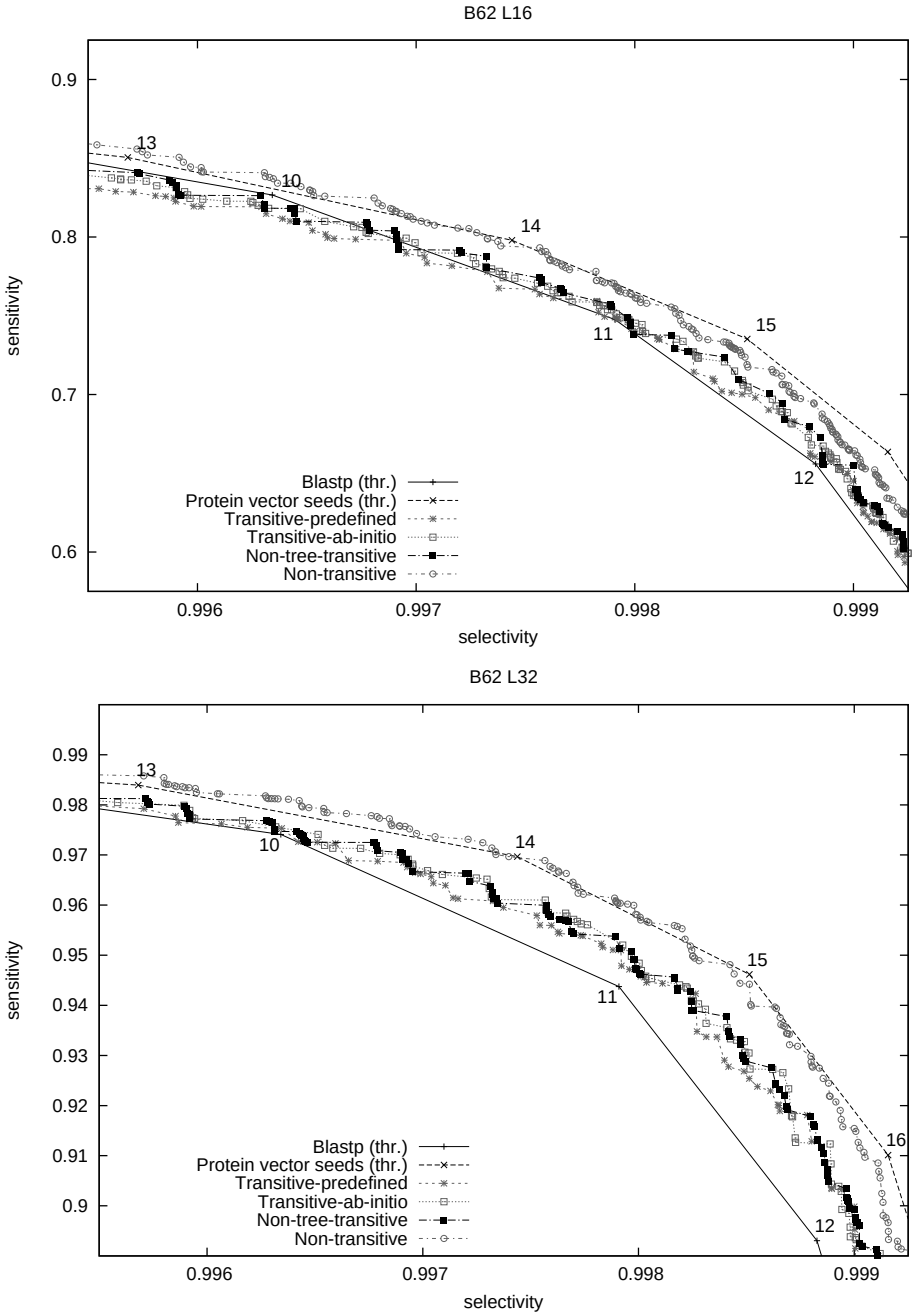
Our goal is to compare between different seed design approaches proposed in this paper, but also to benchmark them against other reference seeding methods. We used two references: the BLASTP seeding method and the family of vector seeds proposed in [4]. Both of them use a score (or weight) resulting from the accumulative contribution of several neighboring positions to define a hit (see Introduction). Therefore, they use a more powerful (and also more costly to implement) formalism of seeding.

To estimate the sensitivity and selectivity of those seeds, we modified our methods described in the previous section by representing an alignment by a sequence of possible individual scores. Foreground and background probability of each score is easily computed from those for amino acid pairs. After that, sensitivity and selectivity is computed similarly to the previous case.

5.4 Results

We compare the performance of the different approaches by plotting ROC curves of Pareto-optimal sets of seeds on the selectivity/sensitivity graph. The two plots in Figure 6 show the results for alignment length 16 and 32 respectively. The two first polylines show the performance of BLASTP with word size 3 and the vector seed family from [4], for different score thresholds. The other curves show the performances of different seed alphabets from Sections 3-4 represented by the Pareto-optimal seeds (seed families) that we were able to construct over those alphabets. As mentioned earlier in Section 5.2, each time we selected the best seed family among those with different number of individual seeds. Typically

Fig. 6. ROC curves of seed performance measured on the probabilistic model



(but not exclusively), points on the plots correspond to seed families with 3 to 5 seeds. Typically, the seed span ranges between 3 and 5 (respectively, 3 and 6) for alignment length 16 (respectively, 32). Seeds with langer span (> 4) tend to occur in seed families with larger number of seeds (> 3).

We observe that non-transitive seeds over the alphabet of Section 3 are comparable in performance with the vector seed family from [4] and clearly outperforms seeds over other alphabets. This result is interesting in itself, although this alphabet is unpractical in many cases, due to its incompatibility with the transitivity condition.

As for the other alphabets, they roughly show a comparable performance among them. Note that using non-hierarchical alphabet (Section 4.3) does not bring much of improvement, which justifies condition (3). For the alignment length 16, our seeds perform comparably to Blastp, with a slightly better performance for high thresholds and a slightly worse performance for low thresholds. On the other hand, for alignments of length 32, our seeds clearly outperform Blastp.

6 Conclusion

The main conclusion of our work is that although the subset seed model is less expressive than the method of accumulative score used in Blastp, carefully designed subset seeds can reach the same or even a higher performance. To put it informally, the use of the accumulative score in defining a hit can, without loss of performance, be replaced by a careful distinction between different amino acid matches without using any scoring system. From a practical point of view, subset seeds can provide a more efficient implementation, especially for large-scale protein comparisons, due to a much smaller number of accesses to the hash table. In particular, this can be very useful for parallel implementations or specialized hardware (see e.g. [17]).

Note that the seed design heuristic sketched in Section 5.2 does not guarantee to compute optimal seeds, and therefore our seeds could potentially be further improved by a more advanced design procedure, possibly bringing a further increase in performance. This is especially true for seeds of large weight (due to a bigger number of those), for which our seed design procedure could produce non-optimal seeds, thus explaining some "drop-offs" in high-selectivity parts of plots of Figure 6.

As far as further research is concerned, the question of efficient seed design remains an open issue. Improvements of the hill climbing heuristics used in this work are likely to be possible.

Acknowledgements Parts of this work have been done during visits to LIFL of Ewa Szczurek (June-August 2006), Anna Gambin and Sławomir Lasota (August 2006) and Mikhail Roytberg (October-December 2006). Those visits were supported by the ECO-NET and Polonium programs of the French Ministry of Foreign Affairs.

References

1. Kucherov, G., Noé, L., Roytberg, M.: A unifying framework for seed sensitivity and its application to subset seeds. JBCB **4**(2) (2006) 553–570
2. Altschul, S., Gish, W., Miller, W., Myers, E., Lipman, D.: Basic Local Alignment Search Tool. Journal of Molecular Biology **215** (1990) 403–410
3. Altschul, S. *et al.*: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. **25**(17) (1997) 3389–3402
4. Brown, D.: Optimizing multiple seed for protein homology search. IEEE/ACM TCBB **2**(1) (2005) 29–38 (earlier version in WABI 2004).
5. Ma, B., Tromp, J., Li, M.: PatternHunter: Faster and more sensitive homology search. Bioinformatics **18**(3) (2002) 440–445
6. Li, M., Ma, B., Kisman, D., Tromp, J.: PatternHunter II: Highly sensitive and fast homology search. JBCB **2**(3) (2004) 417–439 (earlier version in GIW 2003).
7. Brejova, B., Brown, D., Vinar, T.: Vector seeds: an extension to spaced seeds. Journal of Computer and System Sciences **70**(3) (2005) 364–380
8. Noé, L., Kucherov, G.: YASS: enhancing the sensitivity of DNA similarity search. Nucleic Acid Res. **33** (2005) W540–W543
9. Mak, D., Gelfand, Y., Benson, G.: Indel seeds for homology search. Bioinformatics **22**(14) (2006) e341–e349
10. Csürös, M., Ma, B.: Rapid homology search with neighbor seeds. Algorithmica **48**(2) (2007) 187–202
11. Zhou, L., Stanton, J., Florea, L.: Universal seeds for cDNA-to-genome comparison. BMC Bioinformatics **9**(36) (2008)
12. Sun, Y., Buhler, J.: Designing multiple simultaneous seeds for DNA similarity search. In: RECOMB. (2004) 76–84
13. Kucherov, G., Noé, L., Roytberg, M.: Multi-seed lossless filtration. In: CPM. Volume 3109 of LNCS., Springer Verlag (2004) 297–310
14. Yang, I.H. *et al.*: Efficient methods for generating optimal single and multiple spaced seeds. In: IEEE BIBE. (2004) 411–416
15. Xu, J., Brown, D., Li, M., Ma, B.: Optimizing multiple spaced seeds for homology search. In: CPM. Volume 3109 of LNCS., Springer (2004) 47–58
16. Kisman, D., Li, M., Ma, B., Wang, L.: tPatternHunter: gapped, fast and sensitive translated homology search. Bioinformatics **21**(4) (2005) 542–544
17. Peterlongo, P. *et al.*: Protein similarity search with subset seeds on a dedicated reconfigurable hardware. In: PBC. Volume 4967 of LNCS. (2007)
18. Noé, L.and Kucherov, G.: Improved hit criteria for DNA local alignment. BMC Bioinformatics **5**(149) (2004)
19. Keich, U., Li, M., Ma, B., Tromp, J.: On spaced seeds for similarity search. Discrete Applied Mathematics **138**(3) (2004) 253–263 (earlier version in 2002).
20. Li, T., Fan, K., Wang, J., Wang, W.: Reduction of protein sequence complexity by residue grouping. Journal of Protein Engineering **16** (2003) 323–330
21. Murphy, L., Wallqvist, A., Levy, R.: Simplified amino acid alphabets for protein fold recognition and implications for folding. J. of Prot. Eng. **13** (2000) 149–152
22. Henikoff, S., Henikoff, J.: Amino acid substitution matrices from protein blocks. Proc. Natl. Acad. Sci. USA **89** (1992) 10915–10919
23. Henikoff, S., Henikoff, J.: Automated assembly of protein blocks for database searching. Nucleic Acids Res. **19**(23) (1991) 6565–6572
24. Buhler, J., Keich, U., Sun, Y.: Designing seeds for similarity search in genomic DNA. In: RECOMB. (2003) 67–75
25. Ilie, L., Ilie, S.: Long spaced seeds for finding similarities between biological sequences. In: BIOCOMP. (2007) 3–8