On Subset Seeds for Protein Alignment

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

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 a comparative analysis of seeds built over those alphabets 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 seeds is less expressive (but less costly to implement) than the cumulative 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. Finally, we perform a large-scale benchmarking of our seeds against several main databases of protein alignments. Here again, the results show a comparable or better performance of our seeds versus BLASTP.

Index Terms—Protein sequences, protein databases, local alignment, similarity search, seeds, subset seeds, multiple seeds, seed alphabet, sensitivity, selectivity.

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 and 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]. 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 a *cumulative* 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 [11]. On the other hand, it has been understood [7], [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 cumulative 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 pointing to the positions of the subject sequence (database) where the key can hit. Usually these positions are stored in consecutive memory cells within the hash table.

On the other hand, defining a hit through a cumulative contribution of several positions leads to an additional precomputed table that stores, 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 cumulative score of three positions), the expectation, computed according to the Bernoulli sequence model, of the number of neighbors of a key is 19.34, i.e., that many accesses to the hash table are required for each key. For four positions and threshold 18, as in the case of seeds from [4], a key hits expectedly 15.99 keys and this number grows up to 45.59 when the score threshold decreases to 16. This raises an obvious memory problem, for example, for key size 4 and score threshold 18, the total size of neighborhoods is 7,609,575, and for key size 5, the neighborhood table may simply not fit into the memory. Another related implementation problem is cache usage: different keys of a neighborhood generally correspond to remote segments of the hash table and their processing gives rise to cache misses that cause additional latencies.

These implementation issues may become a bottleneck in large-scale protein comparisons. Furthermore, solving these problems may be very helpful in different specific

- *M. Roytberg and E. Furletova are with the Institute of Mathematical Problems in Biology, Pushchino, Moscow Region 142290, Russia. E-mail: mroytberg@mail.ru, janny51@rambler.ru.*
- *A. Gambin and S. Lasota are with the Institute of Informatics, Warsaw University, Banacha 2, 02-097, Poland. E-mail: {aniag, s.lasota}@mimuw.edu.pl.*
- *L. Noé and G. Kucherov are with LIFL/CNRS/INRIA, Bât. M3, Campus Scientifique, 59655 Villeneuve d'Ascq Cédex, France. E-mail: {laurent.noe, gregory.kucherov}@lifl.fr.*
- *E. Szczurek is with the Max Planck Institute for Molecular Genetics, Computational Molecular Biology, Ihnestr. 73, 14195 Berlin, Germany. E-mail: ewa.szczurek@molgen.mpg.de.*

Manuscript received 11 Apr. 2008; revised 21 Oct. 2008; accepted 21 Dec. 2008; published online 8 Jan. 2009.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2008-04-0068. Digital Object Identifier no. 10.1109/TCBB.2009.4.

experimental setups, such as in mapping protein comparison algorithms to specialized computer architecture (see, e.g., [17], [18]) where memory usage may be a crucial issue.

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 the others (transversions) [19], [20]. This leads (at least in the case of *transitive* seed alphabets defined in this paper) to the possibility of using the direct hashing.

Since the protein alphabet is much larger than that of DNA, subset seeds provide a very attractive seeding option for protein alignment. In this paper, we study the performance of subset seeds applied to protein sequences and compare it to existing seeding techniques of BLASTP and vector seeds.

Note again that subset seeds are less expressive than BLAST seeds or vector seeds in general, but in return, admit a more efficient implementation. Besides treating positions independently, subset seeds replace amino acid substitution scores by simply distinguishing different classes of mismatches. Therefore, another way to state the motivation of this work is to ask whether scores are really necessary at the seeding stage of protein alignment. We will show that with a reasonable level of precision, the answer to this question is negative.

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 fine, the quality of an alphabet is determined by the quality of the best seeds that can be constructed over this alphabet. The latter is already a complex optimization problem that is usually solved in practice by heuristic methods. (For a formal analysis of seed design problem, we refer to the recent paper [21] and references therein.) The problem of alphabet design studied in this paper presents an additional complexity as it introduces an additional dimension of the search space (set of possible alphabets), and additionally requires a study of selectivity/sensitivity dependencies rather than simply maximizing the sensitivity for a class of seeds with a given selectivity. In this paper, we propose several heuristic methods that lead to the design of efficient seed alphabets and corresponding seeds.

The paper is organized as follows: 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, wich are useful in practice. Section 4 presents three different approaches to designing transitive seed alphabets, based on a predefined (Section 4.1) or newly designed (Section 4.2) hierarchical clustering of amino acids, as well as on a nonhierarchical clustering (Section 4.3). Section 5 describes comparative experiments made with the designed seeds, obtained both on probabilistic models and on different protein data banks.

2 Preliminaries

Throughout the paper, we denote by

$$\begin{aligned}\Sigma &= \{\mathtt{A},\mathtt{C},\mathtt{D},\mathtt{E},\mathtt{F},\mathtt{G},\mathtt{H},\mathtt{I},\mathtt{K},\mathtt{L},\mathtt{M},\mathtt{N},\mathtt{P},\mathtt{Q},\mathtt{R},\mathtt{S},\mathtt{T},\mathtt{V},\mathtt{W},\mathtt{Y}\}\\ &= \{a_i\}_{i=1..20}\end{aligned}$$

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 $\mathcal{B}$, where k is called the *span* of π. 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 $\langle s_1[i], s_2[i]\rangle \in \alpha_i$. In this case, we say that seed α *hits* the pair s_1, s_2.

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 Section 1). In this paper, we will be mainly interested in transitive seed letters, but we will also study the nontransitive 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 acid 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 classical spaced seeds [22] defined as the number of "identity" letters it contains.

Let f_{ij} be the probability to see the 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.[1] Assume that target alignments are specified by

1. 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 better capture 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.

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 always be 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 $R(t) = \{(a_i, a_j) \mid f_{ij}/b_i b_j > t\}$. Then the following statement holds.[2]

Proposition 1. *$R(t)$ cannot be dominated by any other letter.*

Proof. Assume by contradiction that $R(t)$ is dominated by some letter α, i.e., $f(\alpha) \geq f(R(t))$ and $b(\alpha) \leq b(R(t))$. Consider $\beta = R(t) \setminus \alpha$ and $\gamma = \alpha \setminus R(t)$. Clearly, $f(\beta) \leq f(\gamma)$ and $b(\beta) \geq b(\gamma)$. On the other hand, $\forall (a_i, a_j) \in \beta$, $f_{ij}/b_i b_j > t$, and $\forall (a_i, a_j) \in \gamma$, $f_{ij}/b_i b_j \leq t$. This implies that $f(\beta) = \sum_{(a_i,a_j)\in\beta} f_{ij} > t \sum_{(a_i,a_j)\in\beta} b_i b_j = t b(\beta)$ and similarly $f(\gamma) \leq t b(\gamma)$. We then have $f(\beta) > t b(\beta) \geq t b(\gamma) \geq f(\gamma)$, which contradicts $f(\beta) \leq f(\gamma)$. □

Proposition 1 suggests that letters $R(t)$ are good candidates to be included in the seed alphabet.

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 (W) down to 3.69 for valine (V).

Among nonidentical pairs, only 25 have a score greater than 1 (Fig. 1). A quick analysis shows that these do not form a transitive relation, and therefore, $R(1)$ does not verify the transitivity requirement. This is also the case for other threshold values.

We analyzed a family of threshold letters $R(t)$ for t ranging from 0 to 3 with step 0.05. At the extremities of this interval, $R(0)$ is the "joker" letter admitting all amino acid pairs and $R(3)$ is the letter corresponding to the exact match relation. Among all these letters, there are only 34 different ones. This alphabet of 34 letters (data not shown), denoted by `Nontransitive`, will be used in the experimental part of the paper (Section 5) in order to study how restrictive the requirement of transitive letters is, i.e., how much better are general seeds than those obtained with the restriction of transitivity.

2. It is interesting to point out the relationship with the Neyman-Pearson lemma which is a more general formulation of this statement.

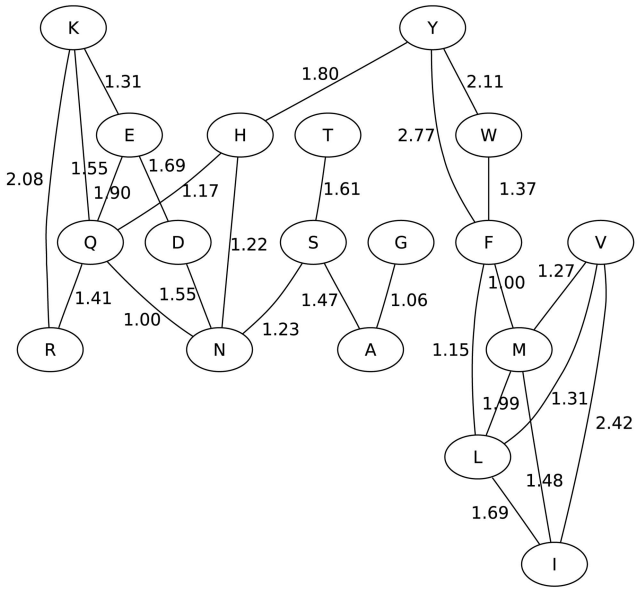

Fig. 1. Amino acid pairs forming letter $R(1)$ of alphabet `Nontransitive`.

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 nonintersecting subsets of keys.

In Sections 4.1 and 4.2, we explore transitive seed alphabets satisfying an additional "hierarchy 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,

$$\text{for any } \alpha_1, \alpha_2 \in \mathcal{B}, \text{either } \alpha_1 \prec \alpha_2 \text{ or } \alpha_2 \prec \alpha_1, \qquad (2)$$

where $\alpha \prec \beta$ means that every set of P_β is a subset of some set of P_α.

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 predefined 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

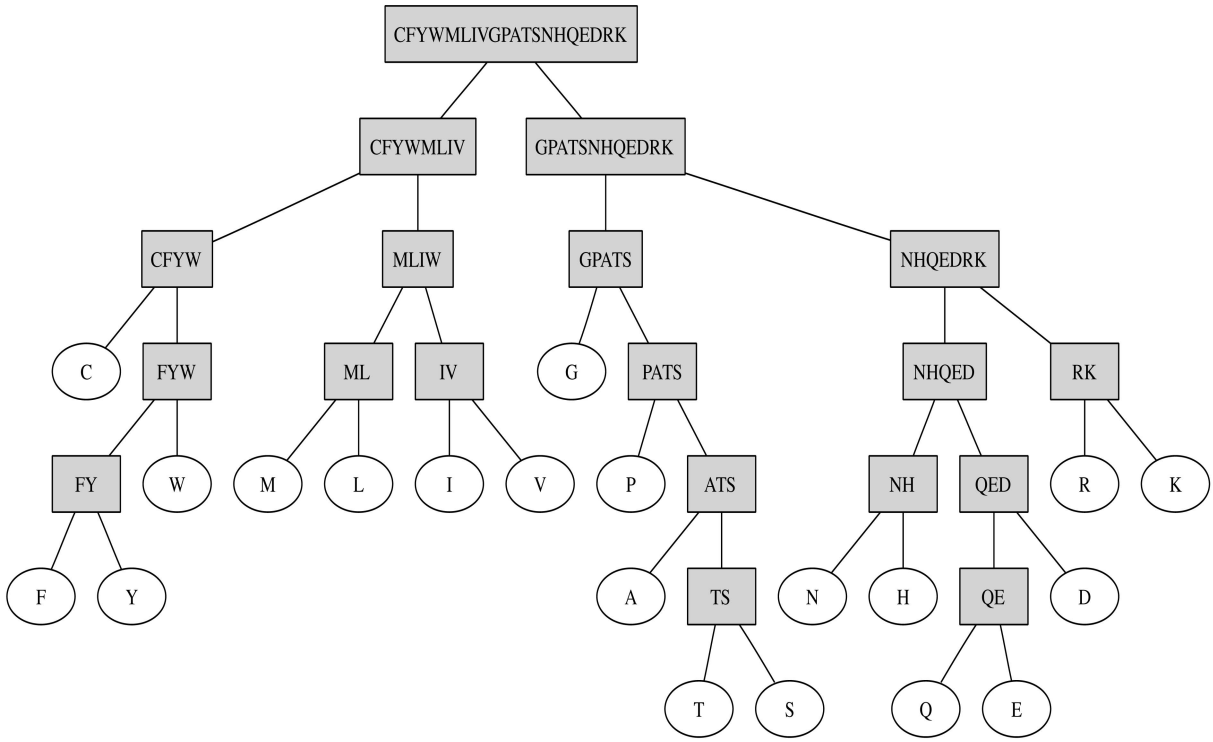

Fig. 2. Hierarchical tree derived from [23].

Section 4.3, we lift condition (2) and study "nonhierarchical" seed alphabets.

4.1 Transitive Alphabets Based on a Predefined Clustering

Assume that we have a biologically significant hierarchical clustering tree which is a rooted binary tree T with 20 leaves labeled by amino acids. Such trees have been proposed in [23], [24] based on different similarity relations. The hierarchical tree derived from [23] is shown in Fig. 2. The tree, obtained with a purely bioinformatics 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`, amino acids with an alcohol group `S,T`, or charged/polar amino acids `E,D,N,Q`. A similar grouping has been obtained in [24].

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

1. α contains all leaves and
2. 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. Clearly, each letter induces a partition on the set of leaves (amino acids). For example, for the tree in Fig. 2, a letter defined by the cut through nodes C, FYW, MLIV, G, P, ATS, NHQEDRK corresponds to the partition `{{C},{FYW},{MLIV},{G},{P},{ATS},{NHQEDRK}}`.

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

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

Hence, in mathematical terms, a seed alphabet is a chain in the inclusion ordering of seed letters. Each hierarchical alphabet can be obtained by a series of refinements (set splittings) of its least refined letter.

Let us analyze what are the maximal seed alphabets w.r.t. inclusion. Clearly, each maximal seed alphabet $\mathcal{B}$ always contains the smallest and the largest seed letters # and _. Interestingly, each maximal alphabet $\mathcal{B}$ contains also @, as @ is comparable (by inclusion) to any other seed letter.

It can be shown that under the above definitions, 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.

Seed alphabets and constrained independence systems. It is interesting to observe that the set of seed alphabets forms a *constrained independence system* [25]. An independence system is a collection of subsets $\mathcal{I} \subseteq 2^E$ over a ground set E, called *independent sets*, such that 1) $\emptyset \in \mathcal{I}$ and 2) if

$X \in \mathcal{I}$ and $Y \subseteq X$, then $Y \in \mathcal{I}$. A maximal (w.r.t. inclusion) independent set is called a *base*.

Let E be the set of all possible seed letters as defined earlier. Then, alphabets verifying (3) form an independence system, where bases correspond to maximal seed alphabets. Moreover, seed alphabets verify two additional conditions of *constrained independence system* [25]: 3) if $X, Y \in \mathcal{I}$ with $|Y| < |X|$, then there is an element $e \in E \setminus Y$ such that $Y \cup \{e\} \in \mathcal{I}$ and 4) the cardinality of every minimal (w.r.t inclusion) set of $2^E \setminus \mathcal{I}$ is 2.

The interest in this observation follows from results of [25] showing that some optimization problems on constrained independence systems can be solved efficiently by greedy algorithms. Assume that we have a score function $s : E \to R$ that we extend additively to independent sets by $s(X) = \sum_{e \in X} s(e)$. For an independence system $\mathcal{I}$, we want to find a base $X \in \mathcal{I}$ with optimal (maximal or minimal) $s(X)$. For constrained independence systems, it was proved [25] that the greedy algorithm yields a base which is *locally optimal*, i.e., better than any neighbor base $Y = (X \setminus \{\alpha_1\}) \cup \{\alpha_2\}$ for some $\alpha_1 \in X$, $\alpha_2 \in E \setminus X$. Here, the greedy algorithm starts with the empty set and iteratively adds most optimal elements of E as long as the current set remains independent. The absolute optimum is hard to compute in general, and the greedy solution is an approximation of it.

Assigning letter score. The above setting requires that each letter α is assigned a score that, intuitively, should measure the "usefulness" of α in a potential alphabet. Defining such a measure is a difficult question as there are too many potential alphabets and we cannot check them all exhaustively. Therefore, we chose to consider only small alphabets $\mathcal{B}_\alpha$, containing α together with a few other letters that are always present in a good seed alphabet. These letters are $\{\_, @, \#\}$. The experiments reported in Section 5 use the alphabet $\mathcal{B}_\alpha = \{\_, \alpha\}$.

Given $\mathcal{B}_\alpha$, we define the score of α as follows: We enumerate all seeds of a given span (typically, 5 or 6) over $\mathcal{B}_\alpha$, and compute the sensitivity and selectivity of each seed according to the protocol described in Section 5.2. Each seed is then associated with a point on a unit square with coordinates corresponding to sensitivity and selectivity (see plots in Fig. 6). The distance of this point to point $(1, 1)$, denoted by $\rho(\alpha)$, measures how good the sensitivity and selectivity jointly are. Besides, the number of occurrences of α in the seed should be taken into account. Overall, we chose to compute the score of a letter by the following formula:

$$w(\alpha) = \sum_{\pi} occ_\pi(\alpha) \cdot \Big(\sqrt{2} - \rho(\alpha)\Big),$$

where the sum is taken over all seeds π of a given span and $occ_\pi(\alpha)$ is the number of occurrences of α in π.

Greedy algorithm. Once every seed letter has been assigned a score, we compute the greedy solution as follows: We compute the maximal subset L of *locally good letters*, i.e., letters α that score better than any letter α' such that $\{\alpha, \alpha'\} \notin \mathcal{I}$. It can be shown that this subset is independent and is included in the solution computed by the greedy algorithm. Then we redefine E and $\mathcal{I}$ by $E' = E \setminus L$ and $\mathcal{I}' = \{Z \subseteq E' \mid Z \cup L \in \mathcal{I}\}$, and apply the algorithm recursively to the independence system $(E', \mathcal{I}')$. The union of all sets L of locally good letters computed along this procedure forms the solution of the greedy algorithm.

Resulting alphabet. Fig. 3 shows alphabet `Transitive-predefined` designed through the approach of this section. The alphabet has been designed from the tree of Fig. 2 and using the alphabet $\mathcal{B}_\alpha = \{\_, \alpha\}$ for assigning the score of a letter α. Each line in Fig. 2 corresponds to a letter (amino acid partition). Among alphabets obtained by varying different parameters in scoring individual letters (such as the alphabet and seed spans used in the scoring procedure), alphabet `Transitive-predefined` produced best seeds and will be used in the experimental part of this work (Section 5).

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 approach that constructs ab initio a hierarchical clustering of amino acids, using a likelihood measure. The approach can be seen as constructing a hierarchy of connected components of a graph based on the likelihood relation considered in Section 3 (see Fig. 1) trying to build components with high likelihood values.

As in Section 4.1, our goal here is to construct a family of seed letters verifying the hierarchy condition (2). This family will be obtained with a simple greedy neighbor-joining clustering algorithm. 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,
   a. consider the set $Bridge(C_k, C_\ell) = \{(a_i, a_j) | a_i \in C_k,\ a_j \in C_\ell\}$.
   b. compute $ForeBridgeProb(k, \ell) = \sum\{f_{ij} | a_i \in C_k,\ a_j \in C_\ell\}$ and $BackBridgeProb(k, \ell) = \sum\{b_i b_j | a_i \in C_k,\ a_j \in C_\ell\}$.
   c. compute $L(k, \ell) = ForeBridgeProb(k, \ell) / BackBridgeProb(k, \ell)$.
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 acids are merged together which produce the maximal increment in the likelihood. 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 Fig. 4. It will be used in experiments presented later in Section 5.

4.3 Nonhierarchical Alphabets

Previous approaches (Sections 4.1 and 4.2) were based on requirement (2), specifying that letters of the seed alphabet should be embedded one into another to form a "nested"

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

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

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 Section 1). Therefore, we also designed nonhierarchical alphabets in order to compare them to hierarchical ones.

To design nonhierarchical alphabets, we used a heuristic that generalizes the one of Section 4.2. The heuristic consists of two stages: first, generate a big number (several thousands) of "reasonable" candidate letters, and then select from them an alphabet containing ~ 20 transitive letters (not necessarily forming a hierarchy).

The algorithm of the first stage exploits the standard paradigm of genetic algorithms: it consequently creates "generations" of transitive letters. The initial population consists of a single "identity" seed letter. At the kth iteration ($k = 1, \ldots, 19$), each letter generates p descendants, each having $(20 - k)$ sets.

To generate descendants of a letter from the kth generation, we use the algorithm given in Section 4.2 but maintain p (instead of just one) best partitions according to the likelihood of the "bridge." The $(k+1)$th generation is selected among all descendants of the kth generation by selecting those q letters α which have the highest likelihood ratio $f(\alpha)/b(\alpha)$. With $p = 100$ and $q = 500$, the procedure gives about 8,000 candidate letters.

To select a small number of these letters to form an alphabet, we tried different heuristics based on the following two ideas: 1) letters with high likelihood ratio are preferred and 2) alphabet letters should have a range of different weights. The second option produced a better alphabet.

Resulting alphabet. We selected 20 letters out of about 8,000 candidates by partitioning the candidates into 20 groups according to their weight ranging from 0 to 1 with increment 0.05, and by picking in each group the letter with maximal likelihood. An alphabet obtained with the above heuristic, called `Nontree-transitive`, is shown in Fig. 5. This alphabet will be used in the experiments reported in Section 5.

5 Experiments

This section describes the experiments we conducted 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 and 4 with respect to probabilistic models. Then in Section 5.5, we benchmark the performance of seeds built over different alphabets from Section 4 against BLASTP on several reference protein databases. For Sections 5.4 and 5.5, all relative experimental data, including scripts, designed alphabets, seeds and seed families, and resulting sensitivity and selectivity measures, have been collected in a supplementary Web page available at http://bioinfo.lifl.fr/yass/iedera_proteins/.

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

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

5.1 Probability Assignment and Alphabet Generation

First of all, we derived probabilistic models in accordance with the BLOSUM62 data from the original paper [26]. We obtained the BLOCKS database (version 5) [27] and the software of [26] 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 Sections 3 (alphabet `Nontransitive`), 4.1 (alphabet `Transitive-predefined`), 4.2 (alphabet `Transitive-ab-initio`), and 4.3 (alphabet `Nontree-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 two-dimensional plot. Best seeds are then defined to be those which belong to the *Pareto* set among all seeds, i.e., those that 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, individual seeds have been chosen to have approximately the same weight, within 5 percent 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 estimated from below by summing the background probabilities of individual seeds.

Seed family design has been done using a hill climbing heuristic (see [28], [29]), 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, and 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 cumulative contribution of several neighboring positions to define a hit (see Section 1). Therefore, they use a more powerful (and also more costly to implement) formalism of seeding.

To estimate the sensitivity and selectivity of these 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

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

Fig. 5. Nonhierarchical alphabet `Nontree-transitive` designed with the algorithm of Section 4.3.

pairs. After that, sensitivity and selectivity is computed similarly to the previous case.

5.4 Results on Theoretical Models

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 Fig. 6 show the results for alignment lengths 16 and 32, respectively. Red and green 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 and 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 (but not exclusively), points on the plots correspond to seed families with four to six seeds. Typically, the seed span ranges between three and five (respectively, three and six) for alignment length 16 (respectively, 32). Seeds with larger span (more than four) tend to occur in seed families with larger number of seeds (more than three).

We observe that seeds over the alphabet of Section 3 (dark blue curve) are comparable in performance with the vector seed family from [4] and clearly outperform seeds over other alphabets. This result is interesting in itself, although in many cases, this alphabet is not practical due to its incompatibility with the transitivity condition.

As for the other alphabets, they roughly show a comparable performance among them. 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. Note that the nonhierarchical alphabet from Section 4.3 does not bring much of improvement, which might indicate that lifting condition (3) does not bring much of additional power. This point, however, requires further investigation.

5.5 Results on Real Data

We made large-scale tests of our seeds on real data by applying them to several main databases of protein alignments. These databases are BALIBASE (version 3) [30], HOMSTRAD [31], IRMBASE (version 1) [32], OXBENCH (version 1.3) [33], PFAM (release 22) [34], PREFAB (version 4) [35], and SMART (version 4) [36].

First, since all above databases except for OXBENCH contain *multiple* alignments, we extracted from each of them a data set of *pairwise* alignments. For this, pairs of aligned sequences have been randomly extracted from multiple alignments and matching gaps removed. To avoid a bias induced by big (in terms of the number of sequences)

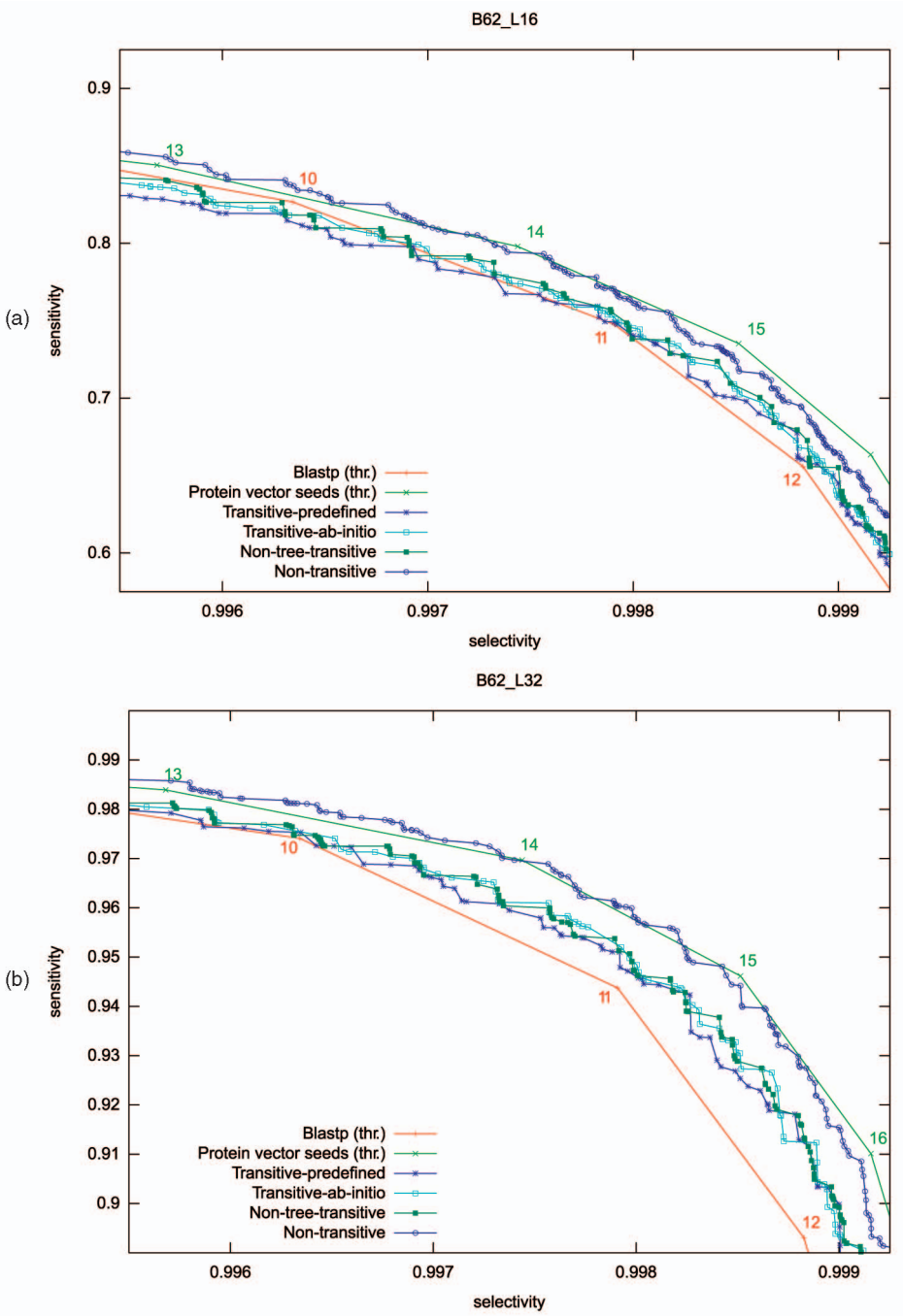

Fig. 6. ROC curves of seed performance measured on probabilistic models for alignment lengths (a) 16 and (b) 32. Blue, cyan, dark green, and dark blue curves represent Pareto-optimal seed families constructed, respectively, over alphabets `Transitive-predefined`, `Transitive-ab-initio`, `Nontree-transitive`, and `Nontransitive`. Each point of these curves corresponds to a seed family, typically three to five seeds (respectively, three to six seeds) for alignment length 16 (respectively, 32). Red and green polylines show the performance of BLASTP with word size 3 and the vector seed family from [4] for different score thresholds.

multiple alignments, we selected a smaller fraction of pairwise alignments from big multiple alignments than from small ones: the number of selected alignments varied from order of n^2 for small alignments to $\sqrt{n}$ for big ones. The total number of alignment processed in our experiments varied from 640 (IRMBASE) to more than 250,000 (PFAM).

For all these data sets, we identified alignments detected by the BLASTP seed for different score thresholds (word length 3, BLOSUM62 matrix, score threshold 10-13). On the other hand, for each BLASTP score threshold, we identified the closest seed family in the Pareto set (cf., Section 5.2) with equivalent or greater selectivity. This has been done for each of the three transitive alphabets proposed in Section 4. Selected seeds can be found at the supplementary material Web page http://bioinfo.lifl.fr/yass/iedera_proteins/.

Results are shown in Fig. 7. Both methods detect a very high fraction of alignments of IRMBASE (all of them for thresholds 10 and 11). The poorest sensitivity is observed on SMART where alignments represent small sequences of protein domains of the same family. A relatively weak

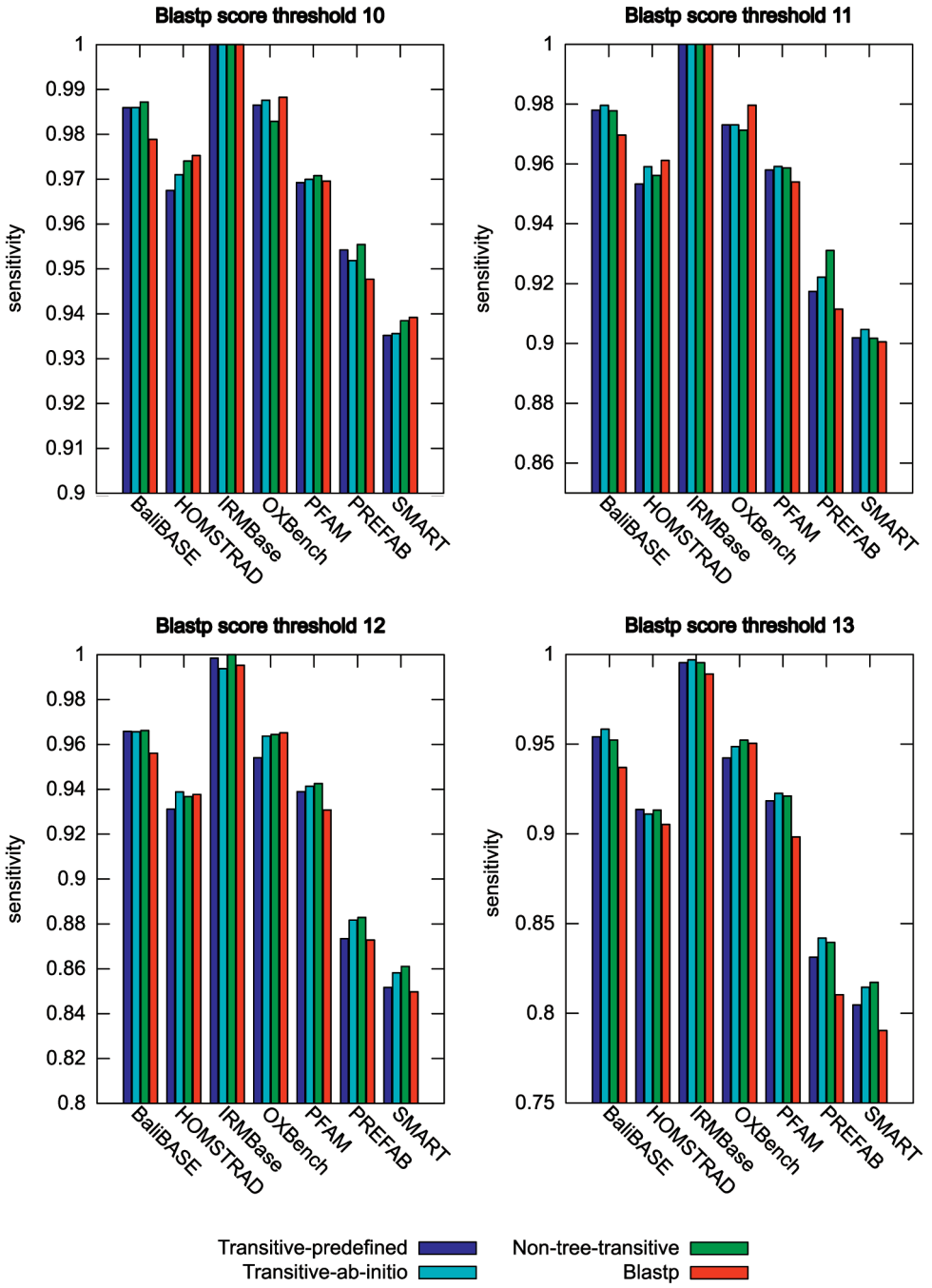

Fig. 7. Sensitivity of subset seeds versus BLASTP measured on benchmark alignment databases.

sensitivity on PREFAB is due to its method of obtaining alignments which is based on structural information and, at the first step, "does not incorporate sequence similarity." Finally, HOMSTRAD combines both structural information (using FUGUE [37]) and sequence information (using PSI-BLAST [3]) which explains a better performance of seed-based search in this case.

Comparing the performance of subset seeds versus BLASTP, the former show clearly a better performance on BALIBASE, PREFAB, and PFAM. On OXBENCH, HOMSTRAD, and SMART, the obtained sensitivity is very close to that of BLASTP. Globally, subset seeds show a better performance for higher selectivity levels (greater thresholds).

6 Conclusion

In this paper, we studied the design of *subset seeds* for protein alignments, which is a very attractive seeding principle that does not use scores at the hitting stage of the alignment procedure. The design of efficient subset seeds subsumes a design of appropriate *seed alphabets*, i.e., sets of *seed letters* that seeds can be built from. In this paper, we studied several approaches to designing alphabets. In Section 3, we considered the most general case when seed letters are only required to induce a symmetric binary relation on amino acids. In Section 4, we focused on *transitive seed alphabets*, where seed letters are required to induce an equivalence

relation. In Section 4.1, we proposed an alphabet construction based on *predefined* hierarchical clusterings of amino acids, while in Section 4.2, we considered a construction based on an ad hoc clustering of amino acids based on the likelihood ratio measure. Finally, in Section 4.3, we lifted the requirement of hierarchical clustering and considered alphabets with possibly "incompatible" letters (in the sense of embedding of equivalence classes).

The main conclusion of our work is that although the subset seed model is less expressive than the method of cumulative 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 cumulative 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], [18]).

Interestingly, the BLAST team reported recently in [38] that they used a reduced amino acid alphabet in order to allow for longer seeds while still keeping the hash table of acceptable size. (Note also that this idea has recently been independently applied in [39] in a slightly different context.) This is done, however, by translating one of the sequences into a compressed alphabet and still using neighborhoods and a cumulative hit criterion. In this work, we demonstrated that instead of this, one can apply carefully designed subset seeds to avoid using neighborhoods and scoring systems at the seeding stage, without sacrificing the performance.

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 nonoptimal seeds, thus explaining some "drop-offs" in high-selectivity parts of plots of Fig. 6.

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

Finally, it would be very interesting to further study the relationship between optimal seeds and seed letters contained in these seeds. In particular, it often appeared in our experiments that optimal seeds contained "nonoptimal" seed letters. Understanding this phenomenon is an interesting theoretical question for further study.

ACKNOWLEDGMENTS

Parts of this work have been done during visits to LIFL of Ewa Szczurek (June-August 2006), Anna Gambin and Slawomir Lasota (August 2006), and Mikhail Roytberg (October-December 2006). These visits were supported by the ECO-NET and Polonium programs of the French Ministry of Foreign Affairs. Laurent Noé was supported by the ANR project CoCoGen (BLAN07-1_185484). Mikhail Roytberg and Eugenia Furletova were supported by grants RFBR 06-04-49249 and 08-01-92496, and INTAS 05-10000008-8028. The authors thank Ivan Tsitovich for fruitful discussions of statistical questions related to this work, and Mathieu Giraud and Marta Girdea for commenting on the manuscript. A preliminary version of this paper appeared in the Proceedings of the ALBIO '08 Workshop, Vienna, Austria, 7-9 July 2008.

REFERENCES

[1] G. Kucherov, L. Noé, and M. Roytberg, "A Unifying Framework for Seed Sensitivity and Its Application to Subset Seeds," *J. Bioinformatics and Computational Biology,* vol. 4, no. 2, pp. 553-570, Apr. 2006 (preliminary version in WABI '05).

[2] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic Local Alignment Search Tool," *J. Molecular Biology,* vol. 215, pp. 403-410, 1990.

[3] S. Altschul, T. Madden, A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs," *Nucleic Acids Research,* vol. 25, no. 17, pp. 3389-3402, 1997.

[4] D. Brown, "Optimizing Multiple Seed for Protein Homology Search," *IEEE/ACM Trans. Computational Biology and Bioinformatics,* vol. 2, no. 1, pp. 29-38, Jan. 2005 (early version appeared in WABI '04).

[5] B. Ma, J. Tromp, and M. Li, "PatternHunter: Faster and More Sensitive Homology Search," *Bioinformatics,* vol. 18, no. 3, pp. 440-445, 2002.

[6] W.J. Kent, "BLAT—The BLAST-Like Alignment Tool," *Genome Research,* vol. 12, pp. 656-664, 2002.

[7] M. Li, B. Ma, D. Kisman, and J. Tromp, "PatternHunter II: Highly Sensitive and Fast Homology Search," *J. Bioinformatics and Computational Biology,* vol. 2, no. 3, pp. 417-439, 2004 (earlier version in GIW '03).

[8] L. Noé and G. Kucherov, "YASS: Enhancing the Sensitivity of DNA Similarity Search," *Nucleic Acid Research,* vol. 33, pp. W540-W543, 2005.

[9] D. Mak, Y. Gelfand, and G. Benson, "Indel Seeds for Homology Search," *Bioinformatics,* vol. 22, no. 14, pp. e341-e349, 2006.

[10] M. Csürös and B. Ma, "Rapid Homology Search with Neighbor Seeds," *Algorithmica,* vol. 48, no. 2, pp. 187-202, June 2007.

[11] B. Brejova, D. Brown, and T. Vinar, "Vector Seeds: An Extension to Spaced Seeds," *J. Computer and System Sciences,* vol. 70, no. 3, pp. 364-380, 2005.

[12] Y. Sun and J. Buhler, "Designing Multiple Simultaneous Seeds for DNA Similarity Search," *Proc. Eighth Ann. Int'l Conf. Computational Molecular Biology (RECOMB '04),* Mar. 2004.

[13] G. Kucherov, L. Noé, and M. Roytberg, "Multi-Seed Lossless Filtration," *Proc. 15th Ann. Combinatorial Pattern Matching Symp. (CPM '04),* pp. 297-310, July 2004.

[14] I.-H. Yang, S.-H. Wang, Y.-H. Chen, P.-H. Huang, L. Ye, X. Huang, and K.-M. Chao, "Efficient Methods for Generating Optimal Single and Multiple Spaced Seeds," *Proc. IEEE Fourth Symp. Bioinformatics and Bioeng. (BIBE '04),* pp. 411-416, 2004.

[15] J. Xu, D. Brown, M. Li, and B. Ma, "Optimizing Multiple Spaced Seeds for Homology Search," *Proc. 15th Symp. Combinatorial Pattern Matching,* pp. 47-58, July 2004.

[16] D. Kisman, M. Li, B. Ma, and L. Wang, "tPatternHunter: Gapped, Fast and Sensitive Translated Homology Search," *Bioinformatics,* vol. 21, no. 4, pp. 542-544, 2005.

[17] P. Peterlongo, L. Noé, D. Lavenier, G. Georges, J. Jacques, G. Kucherov, and M. Giraud, "Protein Similarity Search with Subset Seeds on a Dedicated Reconfigurable Hardware," *Proc. Second Workshop Parallel Computational Biology,* 2007.

[18] V.H. Nguyen and D. Lavenier, "Speeding Up Subset Seed Algorithm for Intensive Protein Sequence Comparison," *Proc. Sixth IEEE Int'l Conf. Research, Innovation & Vision for the Future (RIVF '08),* pp. 57-63, 2008.

[19] L. Noé and G. Kucherov, "Improved Hit Criteria for DNA Local Alignment," *BMC Bioinformatics,* vol. 5, no. 149, Oct. 2004.

[20] L. Zhou, J. Stanton, and L. Florea, "Universal Seeds for cDNA-to-Genome Comparison," *BMC Bioinformatics,* vol. 9, no. 36, 2008.
[21] B. Ma and H. Yao, "Seed Optimization is No Easier Than Optimal Golomb Ruler Design," *Proc. Sixth Asia Pacific Bioinformatics Conf. (APBC '08),* pp. 133-144, Jan. 2008.
[22] U. Keich, M. Li, B. Ma, and J. Tromp, "On Spaced Seeds for Similarity Search," *Discrete Applied Math.,* vol. 138, no. 3, pp. 253-263, 2004 (preliminary version in 2002).
[23] T. Li, K. Fan, J. Wang, and W. Wang, "Reduction of Protein Sequence Complexity by Residue Grouping," *J. Protein Eng.,* vol. 16, pp. 323-330, 2003.
[24] L. Murphy, A. Wallqvist, and R. Levy, "Simplified Amino Acid Alphabets for Protein Fold Recognition and Implications for Folding," *J. Protein Eng.,* vol. 13, pp. 149-152, 2000.
[25] S. Cheng and Y.-F. Xu, "Constrained Independence System and Triangulations of Planar Point Sets," *Proc. Computing and Combinatorics,* pp. 41-50, 1995.
[26] S. Henikoff and J. Henikoff, "Amino Acid Substitution Matrices from Protein Blocks," *Proc. Nat'l Academy of Sciences USA,* vol. 89, pp. 10915-10919, 1992.
[27] S. Henikoff and J. Henikoff, "Automated Assembly of Protein Blocks for Database Searching," *Nucleic Acids Research,* vol. 19, no. 23, pp. 6565-6572, 1991.
[28] J. Buhler, U. Keich, and Y. Sun, "Designing Seeds for Similarity Search in Genomic DNA," *Proc. Seventh Ann. Int'l Conf. Computational Molecular Biology (RECOMB '03),* pp. 67-75, Apr. 2003.
[29] L. Ilie and S. Ilie, "Long Spaced Seeds for Finding Similarities Between Biological Sequences," *Proc. Second Int'l Conf. Bioinformatics & Computational Biology (BIOCOMP '07),* pp. 3-8, 2007.
[30] A. Bahr, J. Thompson, J. Thierry, and O. Poch, "BAliBASE (Benchmark Alignment dataBASE): Enhancements for Repeats, Transmembrane Sequences and Circular Permutations," *Nucleic Acids Research,* vol. 29, no. 1, pp. 323-326, 2001.
[31] A. Stebbings and K. Mizuguchi, "HOMSTRAD: Recent Developments of the Homologous Protein Structure Alignment Database," *Nucleic Acids Research,* vol. 32, pp. D203-D207, 2004.
[32] A.R. Subramanian, J. Weyer-Menkhoff, M. Kaufmann, and B. Morgenstern, "DIALIGN-T: An Improved Algorithm for Segment-Based Multiple Sequence Alignment," *BMC Bioinformatics,* vol. 6, no. 66, 2005.
[33] G. Raghava, S. Searle, P. Audley, J. Barber, and G. Barton, "OXBench: A Benchmark for Evaluation of Protein Multiple Sequence Alignment Accuracy," *BMC Bioinformatics,* vol. 4, no. 47, 2003.
[34] R. Finn, J. Mistry, B. Schuster-Bckler, S. Griffiths-Jones, V. Hollich, T. Lassmann, S. Moxon, M. Marshall, A. Khanna, R. Durbin, S. Eddy, E. Sonnhammer, and A. Bateman, "PFAM: Clans, Web Tools and Services," *Nucleic Acids Research,* vol. 34, pp. D247-D251, 2006.
[35] R.C. Edgar, "MUSCLE: Multiple Sequence Alignment with High Accuracy and High Throughput," *Nucleic Acids Research,* vol. 32, no. 5, pp. 1792-1797, 2004.
[36] I. Letunic, R. Copley, B. Pils, S. Pinkert, J. Schultz, and P. Bork, "SMART 5: Domains in the Context of Genomes and Networks," *Nucleic Acids Research,* vol. 34, no. 1, pp. D257-D260, 2006.
[37] R. Nunez Miguel, J. Shi, and K. Mizuguchi, "Protein Fold Recognition and Comparative Modeling using HOMSTRAD, JOY and FUGUE," *Protein Structure Prediction: Bioinformatic Approach,* pp. 143-169, Int'l University Line Publishers, 2001.
[38] S. Shiryev, J. Papadopoulos, A. Schäffer, and R. Agarwala, "Improved BLAST Searches Using Longer Words for Protein Seeding," *Bioinformatics,* vol. 23, no. 21, pp. 2949-2951, 2007.
[39] P. Peterlongo, L. Noé, D. Lavenier, N.V.H. , G. Kucherov, and M. Giraud, "Optimal Neighborhood Indexing for Protein Similarity Search," *BMC Bioinformatics,* vol. 9, no. 534, 2008.

Mikhail Roytberg received the PhD degree in computer science from Moscow State University in 1983. He is the head of the Applied Math Lab in the Institute of Mathematical Problems in Biology at the Russian Academy of Sciences, Pushchino, Russia. For the past few years, his main research field has been the development of algorithms for comparative analysis of biological sequences.

Anna Gambin received the PhD degree in computer science in 2000 and the habilitation degree in 2008. She is an assistant professor at the Institute of Informatics, Warsaw University, Poland. For the last ten years, she has been doing research on mathematical modeling and algorithms for bioinformatics and computational biology.

Laurent Noé received the PhD degree in computer science from Henri Poincaré University of Nancy, France, in 2005, and is now a lecturer at the University of Lille 1. He is interested in comparative genomics and sequence analysis, and, more specifically, in filtering methods devoted to those tasks.

Slawomir Lasota received the PhD degree in computer science in 2000 and the habilitation degree in 2008. He is an assistant professor in the Institute of Informatics, Warsaw University, Poland. Recently, he has been doing research on concurrency theory as well as on algorithms for bioinformatics and computational biology.

Eugenia Furletova received the master's degree from Puschino State University in 2008, and is now working on her PhD thesis. She is a juniour researcher at the Applied Mathematics Lab in the Institute of Mathematical Problems in Biology of the Russian Academy Sciences, Pushchino, Russia.

Ewa Szczurek received the master's degree in computer science from the University of Warsaw, Poland, in 2006, and the University of Uppsala, Sweden, in 2005. Currenty, she is a PhD student at the International Max Planck Research School for Computational Biology and Scientific Computing.

Gregory Kucherov received the PhD degree in computer science from the USSR Academy of Sciences in 1988 and the habilitation degree from Henri Poincaré University in Nancy in 2000. He is a CNRS research director in the Laboratory for Computer Science (LIFL) in Lille, France. He is currently on leave at the French-Russian J.-V. Poncelet Lab in Moscow, Russia. Since the beginning of the 1990s, he has been doing research on word combinatorics, text algorithms, and combinatorial algorithms for bioinformatics and computational biology.