Sequence Database Search Using Jumping Alignments

Rainer Spang* **Marc Rehmsmeier** **Jens Stoye†**

German Cancer Research Center (DKFZ)
Theoretical Bioinformatics (H0300)
Heidelberg, Germany

Abstract

We describe a new algorithm for amino acid sequence classification and the detection of remote homologues. The rationale is to exploit both vertical and horizontal information of a multiple alignment in a well balanced manner. This is in contrast to established methods like profiles and hidden Markov models which focus on vertical information as they model the columns of the alignment independently.

In our setting, we want to select from a given database of "candidate sequences" those proteins that belong to a given superfamily. In order to do so, each candidate sequence is separately tested against a multiple alignment of the known members of the superfamily by means of a new jumping alignment algorithm. This algorithm is an extension of the Smith–Waterman algorithm and computes a local alignment of a single sequence and a multiple alignment. In contrast to traditional methods, however, this alignment is not based on a summary of the individual columns of the multiple alignment. Rather, the candidate sequence at each position is aligned to one sequence of the multiple alignment, called the "reference sequence". In addition, the reference sequence may change within the alignment, while each such jump is penalized.

To evaluate the discriminative quality of the jumping alignment algorithm, we compared it to hidden Markov models on a subset of the SCOP database of protein domains. The discriminative quality was assessed by counting the number of false positives that ranked higher than the first true positive (FP-count). For moderate FP-counts above five, the number of successful searches with our method was considerably higher than with hidden Markov models.

*Current address: Institute of statistics and decision sciences, Duke University, Durham, North Carolina, USA

†Corresponding author. Address: German Cancer Research Center (DKFZ), Theoretical Bioinformatics (H0300), Im Neuenheimer Feld 280, D–69120 Heidelberg, Germany. Phone: +49 6221 422719, Fax: +49 6221 422849. E-mail: j.stoye@dkfz.de

Introduction

The determination of the function of proteins is among the most fundamental problems in molecular biology. In principle, this task requires elaborate experimental studies for each protein in question. The genome era confronts us with large numbers of protein sequences of unknown function. Scientists now strive for a large scale understanding of protein function. In view of the new sequence data produced by the genome projects, it is crucial to ask for fast and easy methods of functional analysis. Roughly speaking, proteins with similar function descend from common ancestors. They have been randomly altered by mutation events; however, natural selection has extinguished those mutant molecules that do not preserve the protein's function. At the level of sequences, common function is often reflected in terms of conserved regions of the sequences. The standard approach is to compare unknown sequences to proteins with known function. Bioinformatitians have contributed to this project in that they have developed large assortments of computer based methods for extracting information from molecular sequences via sequence comparison, characterization, and classification.

Related Work

Classification of proteins on the basis of common conservation patterns is a standard approach in computational biology. One collects families of proteins with known function and then, given an uncharacterized *candidate* protein sequence, the question is, whether this sequence fits into one of the known families. In this setting, several methods have been developed, including templates (Taylor, 1986), profiles (Gribskov, McLachlan, & Eisenberg, 1987; Luthy, Xenarios, & Bucher, 1994), hidden Markov models (Krogh *et al.*, 1994; Baldi *et al.*, 1994; Eddy, Mitchison, & Durbin, 1995), Bayesian models calculating posterior distributions on possible motifs in a family (Liu, Neuwald, & Lawrence, 1995), and discriminative approaches like the combination of support vector machines and the Fisher kernel method (Jaakkola, Diekhans, & Haussler, 1999).

The first four methods have a common rationale: First one extracts the information common to all sequences of a family, and then one tests the unknown

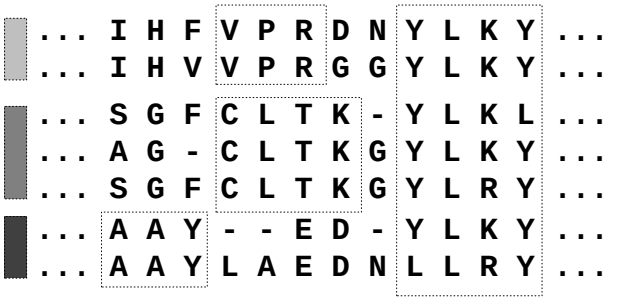

Figure 1: Part of a an (artificial) multiple alignment of a family consisting of 7 sequences, which subdivide into 3 subfamilies. The bars on the left indicate the subfamilies, the dotted boxes highlight conservation patterns.

sequence for existence of these family-specific features. This requires a "summary" of the protein family which can be derived from a multiple alignment. The last method in contrast is a discriminative approach which focuses on differences between family members and non family members.

The Problem

Figure 1 shows some columns of a multiple alignment that exhibits a typical problem of the "summary" approach. One can clearly observe that the 7 sequences subdivide into 3 subfamilies, which is indicated by the shaded bars on the left. A first approach for summarizing the information in the alignment is to proceed column by column: In the first column, we observe either an 'I', an 'S' or an 'A', in the second column it is 'H','G' or 'A', etc. If one wants to decide whether a candidate sequence fits into this family, one can align it to the family and then look site by site whether the residue of the candidate sequence is identical or similar to one of the residues in the family alignment at this position. This *vertical view* on a multiple alignment is the basis of most sequence classification methods. Profiles consist of column specific scores, representing the residue distribution in these columns. Essentially the same is true for the emission distributions of hidden Markov models and the product multinomial distributions of block motifs in the Bayesian alignment approach. However, there is more information contained in the alignment: In column 4 we can frequently observe a 'C', but if there is a 'C' at this position, it is part of the conserved pattern 'C L T K'. This information is obscured in a column based summary of the alignment. A *horizontal view* on the alignment reveals this. The importance of this information becomes obvious in Figure 2.

The alignment in Figure 2 is the same as in Figure 1. In addition, we have aligned a candidate sequence shown below the dashed line. Taking the vertical point of view, one would clearly say that the candidate sequence fits very well into the family since for almost all residues of the candidate sequence the same residue can be found several times among the family sequences

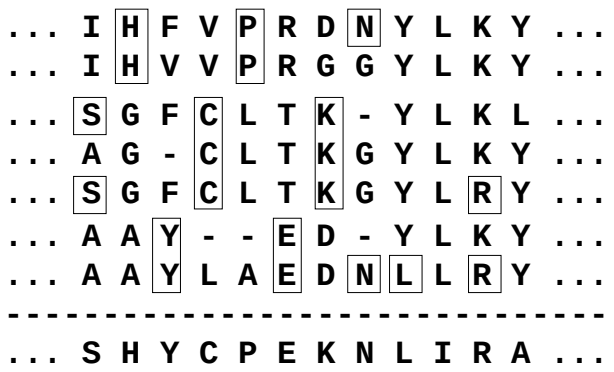

Figure 2: The alignment of Figure 1 and a candidate sequence aligned to it (below the dashed line). Residues that are identical in the family alignment and the candidate sequence are highlighted.

at the same site. However, none of the individual sequences in the family is very similar to the candidate sequence which speaks against a membership in the family. The vertical view indicates a good fit of the candidate sequence to the family; however, this indication is based on a "wild hopping" through the alignment, as is shown by the highlighted residues in Figure 2. This problem is aggravated if we consider an alignment consisting of a large number of sequences from a divergent family. In such a case one has many alternative residues for most sites and hence one has a very high probability of chance hits, due to the "hopping phenomenon".

Both profiles and hidden Markov models are based on scoring a query sequence versus the columns of an alignment. Since both approaches model columns independently from each other, they do not keep track which sequences in the alignment contributed to a high score for a certain residue in a certain column. Consequently, both approaches are subject to hopping, and hopping causes noise in sequence classification.

Our Approach

Clearly, we are talking about correlations between alignment positions. Our approach, however, is not to model these correlations statistically, but to reduce the negative effect of hopping by means of a new algorithm. We will describe a dynamic programming algorithm that locally aligns the candidate sequence to one reference sequence in the multiple alignment, and in addition allows that the reference sequence may change within the alignment. This enables us to make use of the many alternative residues for a certain column in the alignment. But in order to avoid "wild hopping", we penalize each jump. In this way, we reduce the total number of jumps and hence reduce noise. The jumping alignment method is algorithmically related to a method used to detect chimeric sequences (Komatsoulis & Waterman, 1997) and to another method used to find alternative splicings (Gelfand, Mironov, & Pevzner, 1996).

Organization of the Paper

In the next section we briefly review the concept of dynamic programming for calculating local pairwise alignments. The following section contains a detailed description of the jumping alignment algorithm. In the Results section, we derive the optimal jump cost parameter. Based on this parameter, we compare the performance of our algorithm to the well established HMMER package of hidden Markov models. The Discussion addresses zero and infinite jump costs as well as the statistics of the jumping alignment score.

Local Alignment

It is well known that the concept of dynamic programming is well suited for many sequence alignment problems. For example, the Smith-Waterman algorithm (Smith & Waterman, 1981) is one of the most widely used procedures for aligning pairs of molecular sequences.

The algorithm presented in this paper is also based on the dynamic programming paradigm. In fact it can be viewed as an extension of the Smith-Waterman algorithm to the case where one wants to add a single new sequence to a set of already aligned sequences. Before we describe this algorithm, we briefly review the concept of dynamic programming in the simpler case of local pairwise alignments.

Here we are interested in finding and aligning similar segments in a pair of sequences. This requires a formal definition of sequence similarity, which is usually done by introducing a score matrix w that provides a score for every pair of amino acids. In general, similar or identical amino acids are assigned positive scores, whereas dissimilar amino acids obtain negative scores. The expectation value of the score must be negative in order to ensure that the alignments are local. In addition, one needs to score the gap positions in an alignment. For the moment, we assume this is done by penalizing each position in a gap by the same constant value, called *gapcost*.

Given a local alignment, one obtains an alignment score by summing up all similarity scores of positions where two amino acids are aligned to each other, and then subtracting all gap costs. The problem is to construct a local alignment with an optimal similarity score among all possible local alignments of the two sequences.

There is a simple dynamic programming algorithm for solving this optimization problem rigorously (Smith & Waterman, 1981). Assume we are given two sequences $S = s_1, \ldots, s_n$ and $T = t_1, \ldots, t_m$ of length n and m, respectively. The algorithm runs on an $(n+1) \times (m+1)$ matrix D, called the *edit matrix*. For $0 \leq i \leq n$, $0 \leq j \leq m$, cell $D(i,j)$ holds the score of the best local alignment that ends with the positions s_i and t_j. The leftmost column and the top row of D are initialized by $D(i,0) = D(0,j) = 0$. Figure 3 illustrates

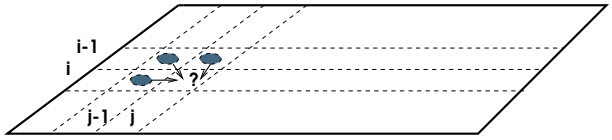

Figure 3: The computation of entry (i,j) of matrix D from its three "predecessor" cells $D(i-1,j-1)$, $D(i-1,j)$, and $D(i,j-1)$.

how the remainder of D is computed recursively by

$$D(i,j) \quad = \quad \max \begin{cases} 0 \\ D(i-1,j-1) + w(s_i,t_j) \\ D(i-1,j) - gapcost \\ D(i,j-1) - gapcost. \end{cases}$$

The optimal local alignment score is the largest entry in the edit matrix, $D(i_{max}, j_{max})$, say.

This algorithm runs in time $O(nm)$ and uses $O(nm)$ space to store the matrix D. The alignment itself is obtained by tracing back through the matrix from $D(i_{max}, j_{max})$ following the choices that were made in the maximization, until a cell with entry 0 is reached. It is a standard technique, however, to reduce the memory usage to $O(n+m)$ by a method introduced by Hirschberg (1975), and extended to the context of local alignment by Huang, Hardison, & Miller (1990), at the expense of essentially only doubling the computation time. If we are only interested in the score of the best local alignment, however, it is much easier to reduce space. Assuming a row-wise computation order, to compute entries in row i, say, we only need values from rows i and $i-1$. Hence, it suffices to store only two neighboring rows at any time of the algorithm which reduces space requirements to $O(m)$.

We have described the algorithm in the case of linear gap costs where the penalty for a gap is proportional to the length of the gap. It is widely acknowledged, however, that a long gap should be penalized less than two shorter gaps of the same length in total. Since computationally cheap, so-called *affine gap costs* are in common use. Here, the initiation of a gap is penalized by a high value *gapinit*, and each additional character of the gap is penalized by another, usually smaller value *gapext*. Based on ideas first published by Gotoh (1982), Huang, Hardison, & Miller (1990) show how to incorporate affine gap costs in local alignments without an overhead in computational complexity.

The Jumping Alignment Algorithm

We now extend the dynamic programming approach to jumping alignments where the scenario is the following. We are given a multiple alignment, called the *seed alignment*, consisting of a few sequences which represent a protein family, and a database of other sequences. The question is, which of the database sequences, called *candidate sequences*, fit into this family. Our method

to test the fit of a candidate sequence to a seed alignment is to compute their optimal local *jumping alignment score*, where, in addition to the standard local alignment score, hopping from one sequence to another sequence in the alignment is penalized.

Formally, the jumping alignment problem is described as follows. Let $S = s_1, \ldots, s_n$ denote the candidate sequence and let A be the seed alignment with K rows and m columns, $A = (A_{k,j})_{k \leq K; j \leq m}$. A *jumping alignment* of S and A is a local alignment of S and the columns of A with an additional annotation that tells for each column of the jumping alignment, which of the rows of the seed alignment the candidate sequence is aligned to. The *jumping alignment cost* of such an alignment is the standard alignment cost of the candidate sequence and the selected alignment sequence, as described above, minus a penalty *jumpcost* for each jump, i.e., for each position where the row-annotation vector changes its value. The problem we want to solve is to find among all jumping alignments of S and A an alignment that maximizes the jumping alignment score. An algorithm to solve this problem is described next.

In order to simplify the exposition, we describe the algorithm stepwise, starting with a simple variant, and then adding more advanced features which improve its performance. We start with linear gap costs. The scoring scheme thus consists of the score matrix (w), the penalty for a gap position (*gapcost*), and the penalty for jumping from one reference sequence to another (*jumpcost*).

Basic Algorithm

We solve the optimization problem by an extension of the Smith-Waterman algorithm. Instead of the one edit matrix $D(i,j)$, we now employ K edit matrices $D_1(i,j), \ldots, D_K(i,j)$, one for each sequence in the seed alignment. For $1 \leq k \leq K$, $D_k(i,j)$ holds the maximal score of all jumping alignments which end with positions s_i and $A_{k,j}$, see Figure 4. For calculating $D_k(i,j)$ one needs to know the value of $3K$ predecessors cells: 3 predecessor cells in D_k as in the case of pairwise alignment, and 3 additional predecessor cells in each of the $K-1$ other edit matrices $D_{k'}$, $k' \neq k$. The maximal entry in the set of all edit matrices gives the optimal jumping alignment score for the candidate sequence and the seed. Obviously, this algorithm runs in $O(nmK^2)$ time and uses $O(nmK)$ space.

Speedup of the Basic Algorithm

The time complexity can be improved. Since a jump from one sequence to another sequence has the same cost for all alignment sequences, we do not have to compute the best jump individually for each alignment sequence. Instead, we pre-compute the optimal values for the three predecessor cells *diagonal* (d), *vertical* (v), and *horizontal* (h) over all alignment sequences, which can be done in $O(K)$ time. To be more precise, assume the computation of cell $D_k(i,j)$ for all values of k. Let

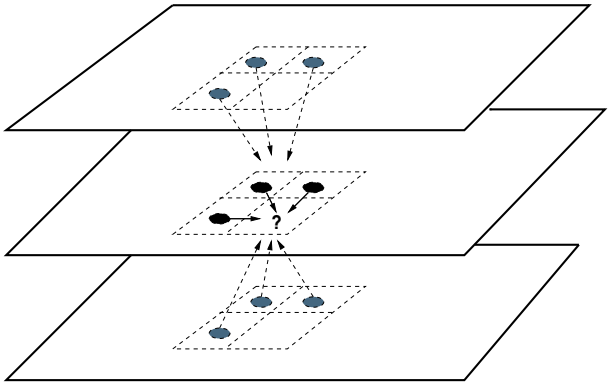

Figure 4: The edit matrices used to compute the best jumping alignment.

k_d, k_v, k_h be the sequences with the best scores in the three predecessor cells of the current cell:

$$
\begin{aligned}
k_d &= \operatorname*{argmax}_{k' \in \{1,\ldots,K\}} \{D_{k'}(i-1, j-1)\}, \\
k_v &= \operatorname*{argmax}_{k' \in \{1,\ldots,K\}} \{D_{k'}(i-1, j)\}, \\
k_h &= \operatorname*{argmax}_{k' \in \{1,\ldots,K\}} \{D_{k'}(i, j-1)\}.
\end{aligned}
$$

Then, when computing $D_k(i,j)$, we need not maximize over all other sequences, but only consider sequences k, k_d, k_v, and k_h. This reduces the time complexity to $O(nmK)$.

Affine Gap Costs

Affine gap costs pose the next problem. The algorithm described by Huang, Hardison, & Miller (1990) can be used as a guideline for calculating optimal jumping alignments with affine gap penalties.

However, one has to be careful how exactly to score affine gaps in the setting of jumping alignments. One has to distinguish between gaps that are introduced by the jumping alignment algorithm and gaps that already exist in the seed alignment. And in particular one has to answer the question how to score gaps in the candidate sequence that run parallel to gaps in the seed alignment. Table 1 summarizes the different types of gaps in the jumping alignment setting, and how they are scored in our algorithm. The full gap penalty only applies if the gap is inserted into the seed alignment (1) or if it is inserted into the candidate sequence and does not run parallel to gaps in the seed alignment (2). A gap in the candidate sequence that runs parallel to a gap in the seed alignment shows that the candidate sequence is "similar" to at least one of the seed sequences and is thus not penalized (3). Next, one needs to consider the rare case where letters in the candidate sequence run parallel to gap characters in the seed alignment (4). These gaps are penalized only by the gap extension penalty to account for the fact that there are other sequences in the alignment that do contain characters

	(1)	(2)	(3)	(4)
candidate seed	***** *---*	*---* *****	*---* *...*	***** *...*
gap penalty	$gapinit + 2 \cdot gapext$	$gapinit + 2 \cdot gapext$	0	$3 \cdot gapext$

Table 1: Affine gaps in jumping alignment. Asterisks denote amino acids, dots denote gaps in the seed alignment, dashes denote gaps that are introduced by the jumping alignment algorithm.

in this region. Finally, there is the possibility for a jump from a gap in one alignment sequence into a gap in another alignment sequence, as is shown in Figure 5. In this case we apply the jump cost, but the gap is treated like a single gap, consequently the gap initiation penalty is imposed only once.

In order to include affine gap costs, similar to Huang, Hardison, & Miller (1990) we use auxiliary matrices V and H that hold the score of the best alignments ending with a gap in either sequence: $V_k(i,j)$ contains the maximal score when s_i is aligned with a gap, and $H_k(i,j)$ contains the maximal score when $A_{k,j}$ is aligned with a gap. We precompute the two sequences with the best vertical and horizontal predecessor in the matrices V and H, respectively:

$$\begin{aligned} k_V &= \underset{k' \in \{1,\dots,K\}}{\operatorname{argmax}} \{V_{k'}(i-1,j)\}, \\ k_H &= \underset{k' \in \{1,\dots,K\}}{\operatorname{argmax}} \{H_{k'}(i,j-1)\}. \end{aligned}$$

Then the following recurrences compute the optimal local jumping alignment cost with affine gap costs as defined above:

$$V_k(i,j) = \max \begin{cases} D_k(i-1,j) - gapinit \\ V_k(i-1,j) - gapext \\ D_{k_v}(i-1,j) - gapinit - jumpcost \\ V_{k_V}(i-1,j) - gapext - jumpcost \end{cases}$$

$$H_k(i,j) = \max \begin{cases} D_k(i,j-1) - gapinit \\ H_k(i,j-1) - G_k(j) \\ D_{k_h}(i,j-1) - gapinit - jumpcost \\ H_{k_H}(i,j-1) - G_k(j) - jumpcost \end{cases}$$

$$D_k(i,j) = \max \begin{cases} 0 \\ D_k(i-1,j-1) + w(s_i, A_{k,j}) \\ D_{k_d}(i-1,j-1) + w(s_i, A_{k,j}) \\ \qquad\qquad - jumpcost \\ V_k(i,j) \\ H_k(i,j) \\ \left.\begin{array}{l} D_k(i,j-i) \\ D_{k_h}(i,j-1) \\ \quad - jumpcost \end{array}\right\} \text{ if } A_{k,j} = gapchar \end{cases}$$

where

$$G_k(j) = \begin{cases} 0 & \text{if } A_{k,j} = gapchar \\ gapext & \text{otherwise.} \end{cases}$$

Finally, it is straight forward to apply the space saving technique by Huang, Hardison, & Miller (1990) to our jumping alignment algorithm. This reduces the space complexity to $O((n+m)K)$, while the time complexity is not increased by the introduction of affine gap costs nor by the reduction of the space complexity. It is $O(nmK)$.

```
                     *************************
seed alignment       *****.........***********
                     ***********..........*****
                     *************************
                     -------------------------
candiate sequence    *****---------------*****
```

Figure 5: A jump inside a gap. Asterisks denote amino acids, dots denote existing gaps in the alignment, dashes denote gaps that are introduced by the jumping alignment procedure.

Implementation

We have implemented the above algorithm including affine gap costs in a program called `JALI` (short for *Jumping ALIgnments*). Given a seed alignment and a candidate sequence, the program provides the optimal jumping alignment score as well as one alignment obtaining this score. A second program called `JSEARCH` is available for using the jumping alignment algorithm in a database search context. The programs are written in standard C and have been compiled on several UNIX platforms. In order to obtain the programs and for further information, see `http://www.dkfz-heidelberg.de/tbi/services/jali/jali.html`.

Results

The jumping alignment method is designed for the purpose of searching molecular databases for remote homologues of a given protein family. In general, protein superfamilies subdivide into families which are less divergent than the entire superfamily. By a superfamily we denote a maximal set of homologous sequences. By a subfamiliy we denote a subset of a superfamily with the property that, if one knows one of its members, one can easily find all other subfamily members by standard database search methods such as BLAST (Altschul *et al.*, 1990) or FASTA (Pearson, 1990). The challenge is to detect new subfamilies of a given superfamily.

Choice of Evaluation Data

For evaluation purposes we need a data set consisting of superfamilies with annotated subfamily structure. However, membership to a superfamily should not only be based on sequence similarity. Such a dependence of annotation and methods creates a "chicken and

egg" problem (Brenner, Chothia, & Hubbard, 1998): Assume, we used a test set where superfamily membership is assessed on the basis of sequence similarity that was reported as being significant by some search method. Clearly, any evaluation procedure based on this data would test for the capability of our method to reproduce the method that was used for annotation. The only way to circumvent this problem is to use a database of known homologies that is not based on sequence comparison only.

The SCOP database (Murzin *et al.*, 1995; Hubbard *et al.*, 1999) is such a database. It has been used several times for the comparison of database search methods (Brenner, Chothia, & Hubbard, 1998; Park *et al.*, 1997; Jaakkola, Diekhans, & Haussler, 1999). The SCOP database classifies protein domains according to the categories *class*, *fold*, *superfamily*, and *subfamily*[1]. A SCOP superfamily comprises sequences that might have low sequence similarity, but whose structure and function suggest a common evolutionary origin. Folds and classes are more abstract and contain sequences that have structural similarities but are not related. SCOP superfamilies are subdivided into subfamilies which are less divergent with respect to sequence, structure, and function. Note that the SCOP classification into superfamily and subfamily includes structural and functional knowledge. This is an important feature to circumvent the "chicken and egg" problem mentioned above. On the other hand, while being based on different concepts, the SCOP data set meets our definition of the superfamily-subfamily relation. SCOP superfamilies are divergent sets of homologous sequences, and a SCOP subfamily consists of sequences with close evolutionary relationships; in general close enough such that if one has detected one member, one can easily find the rest by using this sequence as query in a standard database search.

The SCOP database of release 37 contains 11,822 protein domains. This database is called *pdb100d*. Furthermore, there is a subset of 2,670 sequences that do not share more than 95% sequence similarity called *pdb95d*, one of 2,466 sequences that do not share more than 90% sequence similarity (*pdb90d*), and finally *pdb40d*, consisting of 1,434 sequences.

Evaluation Setting

For evaluating our method we used a setting first described by Jaakkola, Diekhans, & Haussler (1999). In order to emulate the discovery of unknown subfamilies, we split off one subfamily from a SCOP superfamily, thus dividing this SCOP superfamily into two parts, the subfamily which we call the *excluded subfamily* and its complement which we call the *seed*. We used sequences from the seed to construct a seed alignment, and then we searched a database for members of the excluded subfamily.

This setup matches the problem of discovering new subfamilies of a known superfamily. Excluding subfamilies is essential to obtain a fair assessment of the method. Since we are looking for unknown subfamilies, including all available data in the training set would not match the real problem. Note that this setup gives us relatively hard problems of homology detection. A large number of false positives is a common observation. Hence we need a sensible measure to evaluate the performance of the search method. Counts of false positives (Park *et al.* (1998); Jaakkola, Diekhans, & Haussler (1999)) are well suited for this purpose. We used a slight modification of this measure which is motivated by the possibility of iterated searches.

More precisely, we looked for the excluded sequence with the highest score and counted the number of non-superfamily members that ranked above this sequence. Let c be the maximal number of false positives that one is willing to accept before one would consider the search to have failed. If the number of false positives fp is below c, then the *FP-count* is this number, otherwise it is c: $\mathit{FP\text{-}count} = \min(fp, c)$. We speak of an FP-count with cutoff c. Instead of the FP-count which only depends on the highest score of an excluded sequence, Jaakkola, Diekhans, & Haussler (1999) used the median and the maximum of the false positive rates of all excluded sequences. We think that our modification is justified since in view of iterative search methods, detecting a single member of the excluded subfamily is equivalent to detecting the entire subfamily.

We call a SCOP superfamily divided into an excluded subfamily and a seed a *test set*. In total we used 80 test sets, including test sets from the same superfamily but with different excluded subfamilies. These 80 test sets were divided into two parts of 40 test sets each. The first part was used to choose appropriate parameters for the jumping alignment algorithm. We call it the *calibration data*. The second part was used to compare the performance of our method with the HMMER implementation of the hidden Markov model approach. We call these test sets the *evaluation data*.

From each seed we collected all sequences that can also be found in the *pdb90d* database. From these sequences we automatically generated multiple alignments using the CLUSTALW program (Thompson, Higgins, & Gibson, 1994) with default parameters. From these aligned sequences we then fetched all sequences that are contained in the *pdb40d* database. This yielded a multiple alignment consisting of a set of divergent members of the seed. Note that there is no sequence of the excluded subfamily in this alignment and there was no such sequence in the initial larger alignment.

We used these 40 alignments as seed alignments for the jumping alignment algorithm and searched them against the *pdb95d* database. In all experiments shown, we used the VT160 score matrix (Müller & Vingron, to

[1] In the original SCOP notation, what we call a subfamily is called a family. To emphasize the difference to the superfamily category, however, we decided to use the term subfamily.

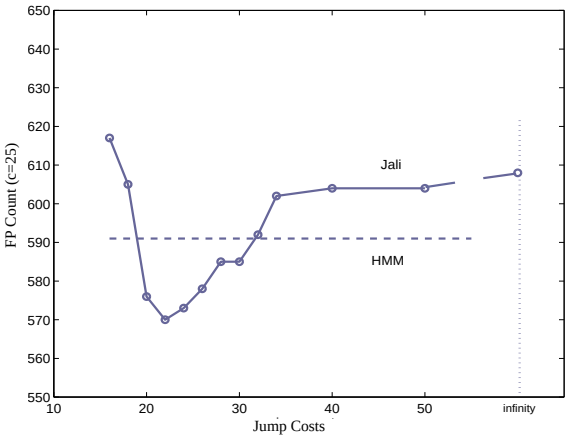

Figure 6: Plot of the jump costs versus the sum of FP-counts of all 40 test sets. For each of the individual 40 FP-counts a cutoff of $c = 25$ was applied.

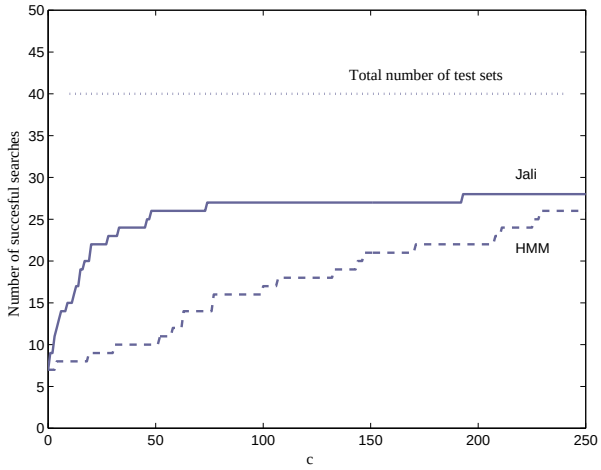

Figure 7: Comparison of the overall performance of the jumping alignment algorithm and the HMMER package of hidden Markov models. The cutoff c is plotted versus the total number of test sets where not more than c false positives were found with a higher score than the first sequence from the excluded subfamily. The dotted line indicates the total number of test sets in the evaluation data.

appear) with gap initiation cost 8 and gap extension cost 4.

Calibration of the Jump Cost

In the described setting we tested several values for the jump cost parameter. For several values between $jumpcost = 0$ and $jumpcost = 50$, and for $jumpcost = \infty$, we added the FP-counts of all test sets from the calibration data. Figure 6 shows the jump costs plotted versus the sum of the FP-counts of all 40 test sets. For each of the individual FP-counts we used a cutoff of $c = 25$.

One can clearly observe the convex shape of the curve which has a minimum for a jump cost of 22. The two extreme cases, zero jump costs (purely vertical point of view; data not shown) and infinite jump costs (purely horizontal point of view) perform worse than intermediate jump costs. This indicates that neither the vertical nor the horizontal point of view is optimal, but a well balanced combination of both. We have also tested various other combinations of the gap costs and different score matrices (data not shown). The effect of these parameters is smaller than that of the jump cost. Our choice, however, is optimal for the 40 test sets of the calibration data.

Comparison to HMMs

The dashed horizontal line in Figure 6 indicates the performance of the hidden Markov model implementation HMMER on the same data. A general advantage of hidden Markov models is that the computation time of a database search is independent of the number of sequences used to train the model. Therefore we used the entire *pdb90d* alignments to train the hidden Markov models, whereas we only used the much smaller *pdb40d* alignments as seeds for the jumping alignments. Hence, the hidden Markov models had much more sequences at their disposal than the jumping alignments. However, this seems to be fair since the sequences are available. It is the time complexity of the jumping alignment algorithm that makes it impractical to explore this information. The HMMs were constructed by the program hmmbuild from the HMMER-package (version 2.1.1) (`http:://hmmer.wustl.edu`). We built both semi-global and local HMMs. Since the local versions performed considerably worse (data not shown) and we again consider it to be fair to use the HMMs in the best possible way, we restricted our comparison to the semi-global versions. The search was accomplished by hmmsearch from the same suite of programs.

Performance on Evaluation Data

For an optimal choice of jump costs, the jumping alignment algorithm seems to be superior. However, we have chosen the parameters such that they perform well on these 40 test sets which leads to a biased view on the comparison of jumping alignments and hidden Markov models. Therefore, we used the evaluation data for comparing these methods. We searched the *pdb95d* database for the excluded families of the 40 test sets, using both HMMER and our algorithm. Figure 7 summarizes the results.

The horizontal axis shows the cutoff c. On the vertical axis the total number of test sets is plotted where we found not more than c false positives before the first excluded sequence is found. The dotted line indicates the total number of test sets in the evaluation setting. First, note that 12 of the 40 test sets show more than

250 false positives for both methods. This clearly indicates the general limitations of sequence based homology detection. For small values of c, that is, if one is very reluctant to tolerate any false positives, there is no significant difference in the performance of both methods. However, if one increases the cutoff to values higher than 5, the jumping alignment procedure finds more sequences from excluded subfamilies than the HMMs. For very high cutoff values this difference in performance shrinks again. Obviously, for difficult problems one has to deal with a certain amount of false positives, and it is in this setup of hard problems, where the jumping alignments outperform the hidden Markov models. These results need to be interpreted with care. We have observed several test sets where the hidden Markov models provide better results than the jumping alignments and others where the opposite holds. There are even a few test sets where pairwise comparison (infinite jump cost) seems to be the optimal method. In the overall evaluation of all 40 test sets, Figure 7 shows that jumping alignments compare favorably to hidden Markov models.

Discussion

We have developed a novel dynamic programming algorithm for detecting remote subfamilies of a given protein superfamily. The general idea is to exploit both horizontal and vertical information of a multiple alignment in a well balanced manner. We call this algorithm the jumping alignment algorithm. While the work on jumping alignments is in a very early stage, our performance evaluation shows that it can compete with such mature, elaborated and established methods as hidden Markov models.

The idea of jumping alignments suggests that remote homologues frequently are chimera of the other members of a protein family. Only few of the data that we have examined, however, seems to support this conjecture. Therefore, we would like to stress that this view is not our main emphasis. Our arguments are more methodical. They are based on the hopping effect and how it causes noise in database searches. We use the jumping alignment score as a measure of fit to a protein family. Of course we could traceback the jumping alignment path, and our program actually allows to do so. That would provide us with a new multiple alignment with the candidate sequence as an additional sequence aligned to the seed alignment. However, this is only of interest if the candidate sequence belongs to the family. Our intention, in contrast, is to decide *whether* it belongs to the family.

Jumping alignments balance the horizontal and the vertical information of a multiple sequence alignment. However, this is done locally. When enlarging the alignment, the jumping alignment algorithm takes both a horizontal look at the next residues in the reference sequence as well as a vertical look on alternative residues in the current position in other sequences. The method can not cope with long range correlations of residues that are in spatial vicinity in the folded protein.

It is instructive to have a look at the two extreme cases where one chooses either zero or infinite jump costs. Clearly, zero jump costs refer to a purely vertical point of view as implemented in profile search methods, and infinite jump costs refer to a purely horizontal point of view, equivalent to a standard database search which returns the database sequence most closely related to the candidate sequence. A combination is only given for intermediate jump costs. In our evaluations, zero jump costs performed significantly bad (data not shown). There are two explanations for the failure of the algorithm for this choice of parameters. First, the method is a very crude version of a profile search. The established profile search methods are much more elaborate using sequence weighting and Bayesian estimators for the amino acid distribution at each position. Furthermore, the profile approach is mostly restricted to conserved blocks of a protein family. In contrast to zero jump costs, infinite jump costs perform surprisingly well. In this case the jumping alignment algorithm is reduced to a very simple method, which is only based on pairwise comparisons. A similar approach has been described by Grundy (1998), and also there the pairwise comparisons performed quite well.

The jumping alignment score is a local alignment score. As in the case of pairwise local alignments, we face the problem that when searching a database, long database entries have a higher chance of obtaining a high score than short ones, even if they are not related at all, see for example Karlin & Altschul (1990); Waterman & Vingron (1994) or Spang & Vingron (1998). In addition we expect that phase transition laws exist for the jump costs as well as for the gap costs, compare Arratia & Waterman (1994). However, the setting seems to be much more complicated than in the case of pairwise sequence alignments. We think that simulations for every individual seed alignment will be appropriate to derive the parameters for length correction, the distribution of scores, and the phase transition lines. In addition to the length normalization, this kind of statistical analysis would yield practical p-values which indicate the statistical significance of a jumping alignment score. However, this work remains to be done.

Acknowledgments

We are very grateful to Martin Vingron who had some of the initial ideas for this work. The distinction between the horizontal and the vertical aspect of a multiple alignment is due to him. He encouraged us to exploit horizontal information for sequence classification. We would also like to thank Tobias Müller for many helpful discussions.

References

Altschul, S.; Gish, W.; Miller, W.; Myers, E.; and Lipman, D. 1990. Basic local alignment search tool. *J.*

Mol. Biol. 215:403–410.

Arratia, R., and Waterman, M. 1994. A phase transition for the score in matching random sequences allowing deletions. *Ann. Appl. Probab.* 4:200–225.

Baldi, P.; Chauvin, Y.; Hunkapiller, T.; and McClure, M. 1994. Hidden Markov models of biological primary sequence information. *Proc. Natl. Acad. Sci. USA* 1;91(3):1059–1063.

Brenner, S.; Chothia, C.; and Hubbard, T. 1998. Assessing sequence comparison methods with reliable structurally identified distant evolutionary relationships. *Proc. Natl. Acad. Sci. USA* 95(11):6073–6078.

Eddy, S.; Mitchison, G.; and Durbin, R. 1995. Maximum discrimination hidden Markov models of sequence consensus. *J. Comp. Biol.* 2(1):9–23.

Gelfand, M.; Mironov, A.; and Pevzner, P. 1996. Gene recognition via spliced sequence alignments. *Proc. Natl. Acad. Sci. USA* 93:9061–9066.

Gotoh, O. 1982. An improved algorithm for matching biological sequences. *J. Mol. Biol.* 162:705–708.

Gribskov, M.; McLachlan, A.; and Eisenberg, D. 1987. Profile analysis: detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA* 84(13):4355–4358.

Grundy, W. 1998. Family-based homology detection via pairwise sequence comparison. In *Proc. of the Second Annual International Conference on Computational Molecular Biology, RECOMB 98*, 94–100. ACM Press.

Hirschberg, D. 1975. A linear space algorithm for computing maximal common subsequences. *Commun. ACM* 18(6):341–343.

Huang, X.; Hardison, R.; and Miller, W. 1990. A space-efficient algorithm for local similarities. *CABIOS* 6(4):373–381.

Hubbard, T.; Ailey, B.; Brenner, S.; Murzin, A.; and Chothia, C. 1999. SCOP: A Structural Classification of Proteins database. *Nucl. Acids. Res.* 27(1):254–256.

Jaakkola, T.; Diekhans, M.; and Haussler, D. 1999. Using the Fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology, ISMB 99*, 149–158. AAAI Press.

Karlin, S., and Altschul, S. 1990. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc. Natl. Acad. Sci. USA* 87:2264–2268.

Komatsoulis, G., and Waterman, M. 1997. Chimeric alignment by dynamic programming: Algorithm and biological uses. In *Proc. of the First Annual International Conference on Computational Molecular Biology, RECOMB 97*, 174–180. ACM Press.

Krogh, A.; Brown, M.; Mian, I.; Sjølander, K.; and Haussler, D. 1994. Hidden Markov models in computational biology. Applications to protein modeling. *J. Mol. Biol.* 235(5):1501–1531.

Liu, J.; Neuwald, A.; and Lawrence, C. 1995. Bayesian models for multiple local sequence alignments and Gibbs sampling strategies. *J. Am. Stat. Assoc.* 90:1156–1170.

Luthy, R.; Xenarios, I.; and Bucher, P. 1994. Improving the sensitivity of the sequence profile method. *Protein Sci.* 3(1):139–146.

Müller, T., and Vingron, M. to appear. Modeling amino acid replacement frequencies. *J. Comp. Biol.*

Murzin, A.; Brenner, S.; Hubbard, T.; and Chothia, C. 1995. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.* 247(4):536–540.

Park, J.; Teichmann, S.; Hubbard, T.; and Chothia, C. 1997. Intermediate sequences increase the detection of homology between sequences. *J. Mol. Biol.* 273(1):349–354.

Park, J.; Karplus, K.; Barrett, C.; Hughey, R.; Haussler, D.; Hubbard, T.; and Chothia, C. 1998. Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods. *J. Mol. Biol.* 284(4):1201–1210.

Pearson, W. 1990. Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods in Enzymology* 183:63–98.

Smith, T., and Waterman, M. 1981. Identification of common molecular subsequences. *J. Mol. Biol.* 147:195–197.

Spang, R., and Vingron, M. 1998. Statistics of large scale sequence searching. *Bioinfomatics* 14:279–284.

Taylor, W. 1986. Identification of protein sequence homology by consensus template alignment. *J. Mol. Biol.* 188:233–258.

Thompson, J.; Higgins, D.; and Gibson, T. 1994. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucl. Acids Res.* 22(22):4673–4680.

Waterman, M., and Vingron, M. 1994. Sequence comparison significance and Poisson approximation. *Stat. Sci.* 9:367.