Pp. 747–760

A Novel Approach to Remote Homology Detection: Jumping Alignments

RAINER SPANG, MARC REHMSMEIER, and JENS STOYE

ABSTRACT

We describe a new algorithm for protein classification and the detection of remote homologs. 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 such as profiles and profile hidden Markov models which focus on vertical information as they model the columns of the alignment independently and to family pairwise search which focuses on horizontal information as it treats given sequences separately. 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 is at each position 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 compare it to profiles, profile hidden Markov models, and family pairwise search on a subset of the SCOP database of protein domains. The discriminative quality is assessed by median false positive counts (med-FP-counts). For moderate med-FP-counts, the number of successful searches with our method is considerably higher than with the competing methods.

Key words: sequence analysis, protein classification, homology detection, jumping alignments.

1. INTRODUCTION

CLASSIFICATION OF PROTEINS on the basis of common conservation patterns is a standard approach in computational biology. One collects sequences of a family of proteins with known function and then, given an uncharacterized candidate sequence from a sequence database, decides whether this sequence fits into the query family. In this setting, several methods have been developed, including templates (Taylor, 1986), profiles (Gribskov *et al.*, 1987; Luthy *et al.*, 1994), profile hidden Markov models (HMMs) (Krogh *et al.*, 1994; Baldi *et al.*, 1994; Eddy *et al.*, 1995), Bayesian models calculating posterior distributions on

Deutsches Krebsforschungszentrum (DKFZ), Theoretische Bioinformatik, Heidelberg, Germany.

possible motifs in a family (Liu *et al.*, 1995), discriminative approaches such as the combination of support vector machines and the Fisher kernel method (Jaakkola *et al.*, 2000), and family pairwise search (FPS) (Grundy, 1998). Templates, profiles, HMMs, and Bayesian models extract the information common to all sequences of a family and test an unknown sequence for existence of these family-specific features. This requires a summary of the protein family which can be derived from a multiple alignment. FPS uses the given sequence data without further processing. Discriminative approaches focus on differences between family members and nonmembers.

Figure 1 exemplifies a typical problem of the summary approach. The left part shows some columns of a multiple alignment whose seven sequences subdivide into three subfamilies which is indicated by the shaded bars to 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 profile-based classification methods. Profiles consist of column-specific scores representing the residue distributions in these columns. Essentially the same is true for the emission distributions of profile HMMs and the product multinomial distributions of block motifs in the Bayesian alignment approach.

However, there is more information contained in the alignment of Fig. 1: In column four, we can frequently observe a C, but if there is a C at this position, it is part of the conserved pattern CLTK. 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 the right part of the figure. The alignments on both sides are the same. 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 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. 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 profile HMMs are based on scoring a query sequence versus the columns of an alignment. Since these approaches model columns independently from each other, they do not keep track of 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.

Although we are talking about correlations between alignment positions, our approach is not to model these correlations statistically, but to reduce the negative effect of hopping by means of a new algorithm.

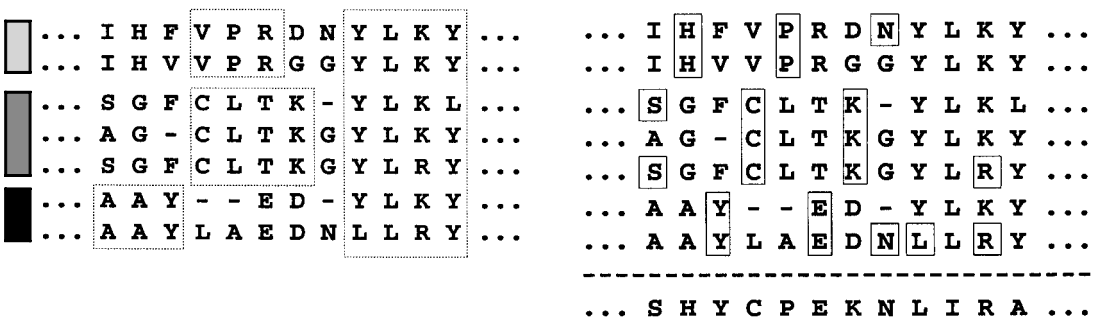

FIG. 1. Left: Part of an (artificial) multiple alignment of a family consisting of seven sequences which subdivides into three subfamilies. The bars on the left indicate the subfamilies; the dotted boxes highlight conservation patterns. **Right:** The same alignment and a candidate sequence aligned to it (below the dashed line). Residues that are identical in the family alignment and the candidate sequence are highlighted.

We 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.

In spirit, the jumping alignment method is similar to fragment-based alignment methods such as DIALIGN (Morgenstern *et al.*, 1998) where longer alignments are "chained" together from several strong local similarities (fragments). We are not aware, though, that fragment-based alignment methods have been applied in a database search context. Also, most fragment-based methods implement a heuristical two-step procedure, while jumping alignments are based on the rigorous optimization of the jumping alignment score using dynamic programming. Our method is also algorithmically related to a method used to detect chimeric sequences (Komatsoulis and Waterman, 1997) and to another method used to find alternative splicings (Gelfand *et al.*, 1996).

A preliminary version of this paper has been presented at the Eighth International Conference on Intelligent Systems for Molecular Biology (Spang *et al.*, 2000).

2. THE JUMPING ALIGNMENT ALGORITHM

The algorithm presented in this paper is based on the dynamic programming paradigm. In fact, it can be viewed as an extension of the Smith–Waterman algorithm (Smith and Waterman, 1981) to the case where one wants to add a single new sequence to a set of already-aligned sequences. This is in spirit similar to profile alignments, but rather than condensing the alignment to a position-specific scoring matrix, we keep it in its original form.

The scenario is the following. We are given a multiple alignment, called the *query alignment*, consisting of sequences that 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 the query 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.

The jumping alignment problem is formally described as follows. Let $S = s_1, \ldots, s_n$ denote the candidate sequence, and let A be the query alignment with K rows and m columns, $A = (a_{k,j})_{1 \le k \le K, 1 \le j \le m}$. 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 query alignment the candidate sequence is aligned to. The *jumping alignment score* of such an alignment is the standard alignment score of the candidate sequence and the selected query alignment sequence minus a penalty *jumpcost* for each jump, i.e., for each position where the annotation changes its value. Sequence similarity is measured by means of 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, we need to score gap positions. In order to simplify the exposition, we describe the algorithm stepwise, starting with linear gap costs. Later we extend it to a more general scoring scheme for gaps.

In the linear case, the scoring scheme 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*). 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.

2.1. Basic algorithm

We solve the optimization problem by an extension of the Smith–Waterman algorithm. We use K *edit matrices* $D_1, \ldots, D_K$, one for each sequence in the query alignment. For $1 \le k \le K$, $D_k(i, j)$ holds the maximal score of all jumping alignments that end at positions i in s and (k, j) in A. For calculating $D_k(i, j)$, one needs to know the values of $3K$ predecessor cells: three predecessor cells in D_k as in the

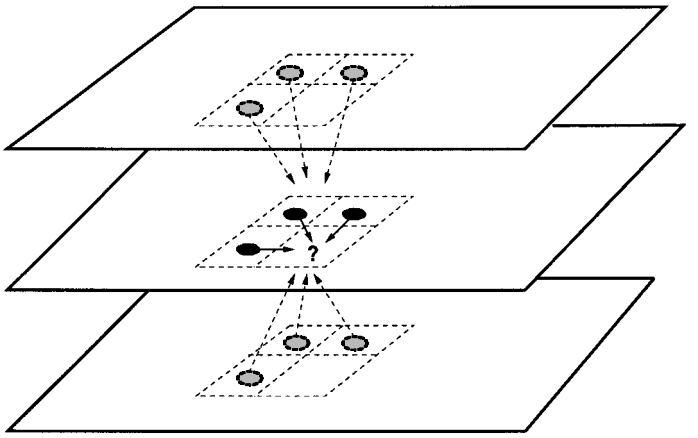

FIG. 2. The edit matrices used to compute the best jumping alignment.

case of pairwise alignment, and three predecessor cells in each of the $K-1$ other edit matrices $D_{k'}$, $k' \neq k$ (see Fig. 2):

$$D_k(i, j) = \max \begin{cases} 0 \\ D_k(i-1, j-1) + w(s_i, a_{k,j}) \\ D_k(i-1, j) - gapcost \\ D_k(i, j-1) - G_k(j) \\ D_{k'}(i-1, j-1) + w(s_i, a_{k,j}) - jumpcost \quad \text{for all } k' \neq k \\ D_{k'}(i-1, j) - gapcost - jumpcost \quad \text{for all } k' \neq k \\ D_{k'}(i, j-1) - G_k(j) - jumpcost \quad \text{for all } k' \neq k \end{cases}$$

where

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

The reason for the special scoring of gaps that are introduced in the sequence s is that we do not want to penalize gaps that are running parallel to already existing gaps in the query alignment A. The function $G_k(j)$ distinguishes between the case where s_i is aligned with an already existing gap in A (*gapchar*), which is not penalized, and the case where s_i is aligned with an amino acid in A, and hence the penalty *gapcost* is imposed.

The maximal entry in the set of all edit matrices gives the optimal jumping alignment score for the candidate sequence and the query alignment. This algorithm runs in $O(nmK^2)$ time and uses $O(nmK)$ space.

2.2. *Speedup of the basic algorithm*

The time complexity of the algorithm 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 precompute 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. More precisely, in the calculation of cells $D_k(i, j)$ for all values of k, let k_d, k_v, k_h be the sequences with the best scores in the $3K$ predecessor cells of $D_k(i, j)$:

$$k_d = \operatorname*{argmax}_{k' \in \{1,\ldots,K\}} D_{k'}(i-1, j-1),$$

$$k_v = \operatorname*{argmax}_{k' \in \{1,\ldots,K\}} D_{k'}(i-1, j),$$

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

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:

$$D_k(i, j) = \max \begin{cases} 0 \\ D_k(i-1, j-1) + w(s_i, a_{k,j}) \\ D_k(i-1, j) - gapcost \\ D_k(i, j-1) - G_k(j) \\ D_{k_d}(i-1, j-1) + w(s_i, a_{k,j}) - jumpcost \\ D_{k_v}(i-1, j) - gapcost - jumpcost \\ D_{k_h}(i, j-1) - G_k(j) - jumpcost \end{cases}$$

This reduces the time complexity to $O(nmK)$.

2.3. *Affine gap costs*

It is generally accepted that a good compromise between computation time and biologically appropriate gap penalties is provided by affine gap costs where a gap of length ℓ is penalized by an affine function $gapcost(\ell) = gapinit + (\ell - 1)gapext$ for nonnegative constants *gapinit* and *gapext*. In this section, we show how the jumping alignment algorithm can be extended to handle affine gap costs with essentially no overhead in computational complexity. As a guideline, we use the algorithm described by Huang *et al.* (1990).

The complicating factor is that one has to be careful how exactly to score affine gaps in the setting of jumping alignments. For linear gap costs, we distinguished the two possibilities that a gap newly introduced into S either is aligned with an amino acid or with an already existing gap in A. These two cases could easily be tracked by the function G_k. With affine gap costs, though, many more cases are possible where a newly introduced gap in S starts or ends before, with, or after an already existing gap in A. Fig. 3 lists all 15 such possibilities.

As for linear gap costs, we apply the rule that newly introduced gaps in S that run parallel to already existing gaps in A are not penalized. As above, this is taken care of by special functions that distinguish between the two cases. While in the linear case one function $G_k(j)$ suffices, here we need two such functions, denoted $G_k^{init}(j)$ for gap initiations and $G_k^{ext}(j)$ for gap extensions:

$$G_k^{init}(j) = \begin{cases} 0 & \text{if } a_{k,j} = gapchar \\ gapinit & \text{otherwise} \end{cases}$$

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

Using G_k^{init} and G_k^{ext} instead of *gapinit* and *gapext* takes care of all alignment columns in Fig. 3 that have a dot in the top row aligned with a dash in the bottom row, so that cases (2), (3), and (4) reduce to case (1); cases (6), (8), and (10) reduce to case (11); and cases (7) and (9) dissolve into two separate gaps, one of type (1) and one of type (11). Case (5) is also considered; the penalty is 0.

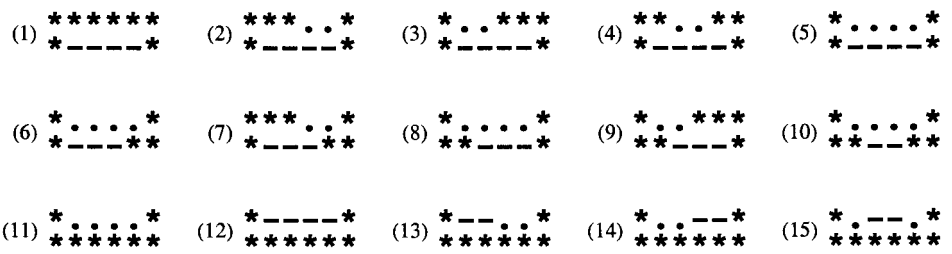

FIG. 3. The 15 different cases for combinations of already existing and newly introduced gaps when aligning a sequence (bottom row) and a multiple alignment (top row; only one sequence of the alignment is shown). Asterisks denote amino acids, dots denote already existing gaps in the alignment, dashes denote gaps that are introduced by the jumping alignment algorithm.

Next, we make a design decision that regards cases (11), (13), (14), and (15) of Fig. 3, where amino acids from the candidate sequence S are aligned with already existing gaps in the query alignment A. In our algorithm, such an alignment is penalized only by the gap extension cost. This accounts for the fact that there are other sequences in the alignment that do contain characters in this region, and hence this region of the alignment might be of dubious value anyway. Algorithmically, we can treat the existing gap character in A as an additional character of the alphabet and generalize the score matrix w such that $w(a, gapchar) = gapext$ for every amino acid a. By this, cases (13), (14), and (15) reduce to case (12), and case (11) is no longer treated as an explicit gap. Hence, what remains are the two simple cases (1) and (12), where the full affine gap penalty is applied.

As in the nonaffine case, jumps are penalized independently of gaps. For example, if a gap is opened in one alignment sequence and after a jump it is continued in another alignment sequence (see Fig. 4), then this is treated like a single gap, and consequently the gap initiation penalty is imposed only once.

In order to compute affine gap costs efficiently, similar to Gotoh (1982) and later Huang *et al.* (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. Similarly as with k_d, k_v, and k_h above, for efficiency reasons, we precompute the two sequences with the best vertical and horizontal predecessor in the matrices V and H, respectively:

$$k_V = \operatorname*{argmax}_{k' \in \{1,\ldots,K\}} V_{k'}(i-1, j),$$

$$k_H = \operatorname*{argmax}_{k' \in \{1,\ldots,K\}} H_{k'}(i, j-1).$$

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

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

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

$$D_k(i, j) = \max \begin{cases} 0 \\ D_k(i-1, j-1) + w(s_i, a_{k,j}) \\ D_{k_d}(i-1, j-1) + w(s_i, a_{k,j}) - jumpcost \\ V_k(i, j) \\ H_k(i, j) \end{cases}$$

Finally, it is possible to apply the space-saving technique of Huang *et al.* (1990) to our jumping alignment algorithm. This would reduce the space complexity to $O((n+m)K)$. The time complexity is not increased by the introduction of affine gap costs, nor would it be increased by the reduction of the space complexity, and thus remains $O(nmK)$.

FIG. 4. A jump inside a gap. Asterisks denote amino acids, dots denote existing gaps in the alignment, and dashes denote gaps that are introduced by the jumping alignment procedure.

We have implemented the above algorithm including affine gap costs in a program called JALI (short for *Jumping ALIgnments*). Given an alignment and a candidate sequence, the program provides the optimal jumping alignment score as well as one alignment of 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. The programs and further information can be obtained from *bibiserv.techfak.uni-bielefeld.de/jali*.

3. RESULTS

The jumping alignment method is designed for the purpose of searching sequence databases for remote homologs of a given protein superfamily. Superfamilies are maximal sets of homologous sequences. In general, protein superfamilies subdivide into families which are less divergent than the entire superfamily. The challenge is to detect new families of a given superfamily.

3.1. Choice of evaluation data

For evaluation purposes, we need a data set consisting of superfamilies with annotated family structure. However, there is a dependence of annotation and search methods which creates a "chicken and egg" problem (Brenner *et al.*, 1998): Assume, we used a test set where superfamily membership was assessed on the basis of sequence similarity that was reported as being significant by some search method. 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; Lo Conte *et al.*, 2000) is such a database. It has been used several times for the comparison of database search methods (Brenner *et al.*, 1998; Park *et al.*, 1997; Jaakkola *et al.*, 2000). The SCOP database classifies protein domains according to the categories *class*, *fold*, *superfamily*, and *family*. Members of a family have close evolutionary relationships. A 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. Note that the SCOP classification into superfamily and family categories includes structural and functional knowledge. This is an important feature to circumvent the chicken and egg problem mentioned above.

From the SCOP release 53, we used the nonredundant subset *pdb90d* which comprises sequences that do not share more than 90% sequence similarity and which contains 4,861 sequences. We split this set into two halves, one consisting of every second superfamily starting with the first one (the *odd set*), the other consisting of the rest (the *even set*). The whole evaluation, including construction of test and training data, execution of methods, performance assessment, and figure and table construction, was accomplished with the Phase4 evaluation system (Rehmsmeier, 2002).

3.2. Construction of calibration and evaluation data

For the evaluation of our method, we selected from the odd set all superfamilies that comprised at least two families. From these superfamilies, families were chosen to be test families if both the family itself and the remainder of the superfamily contained at least five sequences each. The remainder of the superfamily (the *training set*) was further restricted to have not more than 50 sequences for reasons of efficiency. The procedure resulted in a *calibration set* of 42 test families with associated training sets. The construction was repeated on the even set, resulting in an *evaluation set* of 56 test families with associated training sets. The selection of test and training sequences is the same as in Rehmsmeier and Vingron (2001), though there release 37 of the SCOP database was used and no split was done. It matches the problem of discovering new families of an already known superfamily.

For each test family, the remaining sequences of the surrounding superfamily were multiply aligned with CLUSTAL W (version 1.74) (Thompson *et al.*, 1994). From the alignments, domain HMMs were built with hmmbuild from the HMMER package (version 2.1.1) (*www.hmmer.wustl.edu/*) and profiles with pfmake

from the ISREC generalized profile package (version 2.1) (Bucher *et al.*, 1996) using the BLOSUM62 score matrix with a gap initiation cost of 9 and a gap extension cost of 2.

3.3. Calibration of the jump cost

To determine the optimal jump cost for JSEARCH, we used the 42 alignments from the calibration data set as queries against the odd set of the pdb90d database, using the BLOSUM62 score matrix with a gap initiation cost of 9 and a gap extension cost of 2. For the jump cost parameter, several values were tested, ranging from 0 to 50. For each jump cost, ROC_{50} measures were calculated and averaged over all test cases. ROC_{50} is the sum of all true positive counts up to TPC_{50}, with TPC_n being the number of positives that score equally well or better than the first n negatives (cf. Gribskov and Robinson, 1996). The results are shown in Fig. 5.

The curve has a distinct convex shape with its maximum performance at a jump cost of 18. The two extremes, zero jump costs (purely vertical point of view; data not shown) and infinite jump costs (purely horizontal point of view, data not shown but identical to jump costs of 50) 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 is best.

3.4. Performance on evaluation data

We used the evaluation data for comparing JSEARCH to several other methods. Finding an excluded family with the rest of the surrounding superfamily as in our evaluation setting is a relatively hard problem 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. False positive counts (FP-counts) are well suited for this purpose. The FP-count for a test sequence is the number of equally well or higher scoring unrelated (nonsuperfamily) sequences. We used the median of FP-counts for all test sequences (med-FP-count). From the med-FP-counts, we constructed performance figures (Jaakkola *et al.*, 2000) which for cut-offs of med-FP-counts on the x-axis show the percentage of test sets with an equally well or better med-FP-count on the y-axis.

We compared JSEARCH to the family pairwise search (FPS) method, both in its original form of using BLAST (Altschul *et al.*, 1990) and in a variant that performs full Smith–Waterman searches, to profiles as implemented in the ISREC generalized profile package, and to profile hidden Markov models (HMMs) as implemented in the HMMER package. Using the Smith–Waterman variant of FPS makes FPS more comparable to JSEARCH.

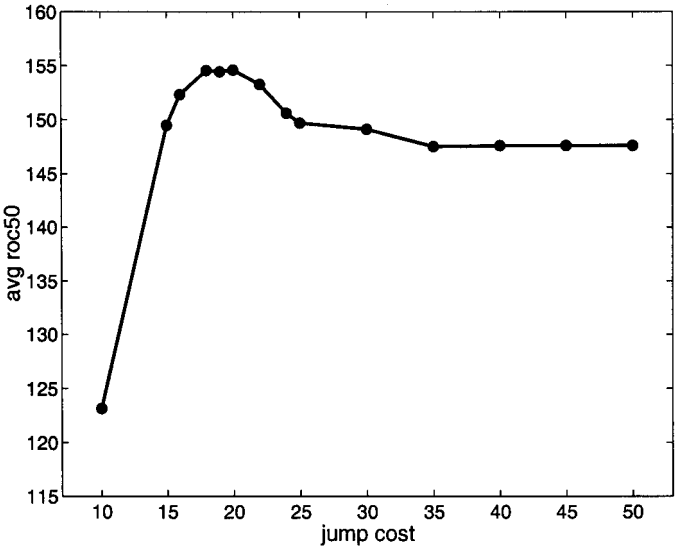

FIG. 5. Average ROC_{50} for various choices of jump costs.

In FPS, pairwise alignment scores are calculated for each sequence of the query against a database sequence (candidate sequence). The maximum of these scores is taken as the score of the candidate. Thus, FPS exclusively uses horizontal information. Profiles and profile HMMs, on the other hand, exclusively use vertical information in that they condense each (conserved) column of a multiple alignment to a score distribution.

Figure 6 shows performance figures of JSEARCH and the four other methods tested. Solid lines represent the performance of JSEARCH and dashed and dotted lines represent the other methods. In the top left, we compare JSEARCH with jumps to JSEARCH with infinite jump costs. One can observe how the vertical information improves the average performance over the 56 test sets. In the top right, we show the performance of the FPS approach in the Smith–Waterman variant (dashed line) and in the BLAST variant (dotted line). The two plots demonstrate that the gap information contributes to the performance of JSEARCH. In the bottom left, we compare JSEARCH to pfsearch and in the bottom right to hmmsearch.

Our results indicate that both hidden Markov models and profiles perform on average worse on our evaluation data than JSEARCH and FPS. The poor performance comes a little bit as a surprise since both are very popular methods for homology detection. However, other evaluation studies (Grundy, 1998) came to similar conclusions. These results need to be interpreted with care. It is important to note that Fig. 6 shows only the average performance of the methods. Table 1 shows the 56 test sets from the evaluation data set with their superfamily and excluded test family names. One observes several cases where the hidden Markov models give better results than the jumping alignments and others where the opposite holds.

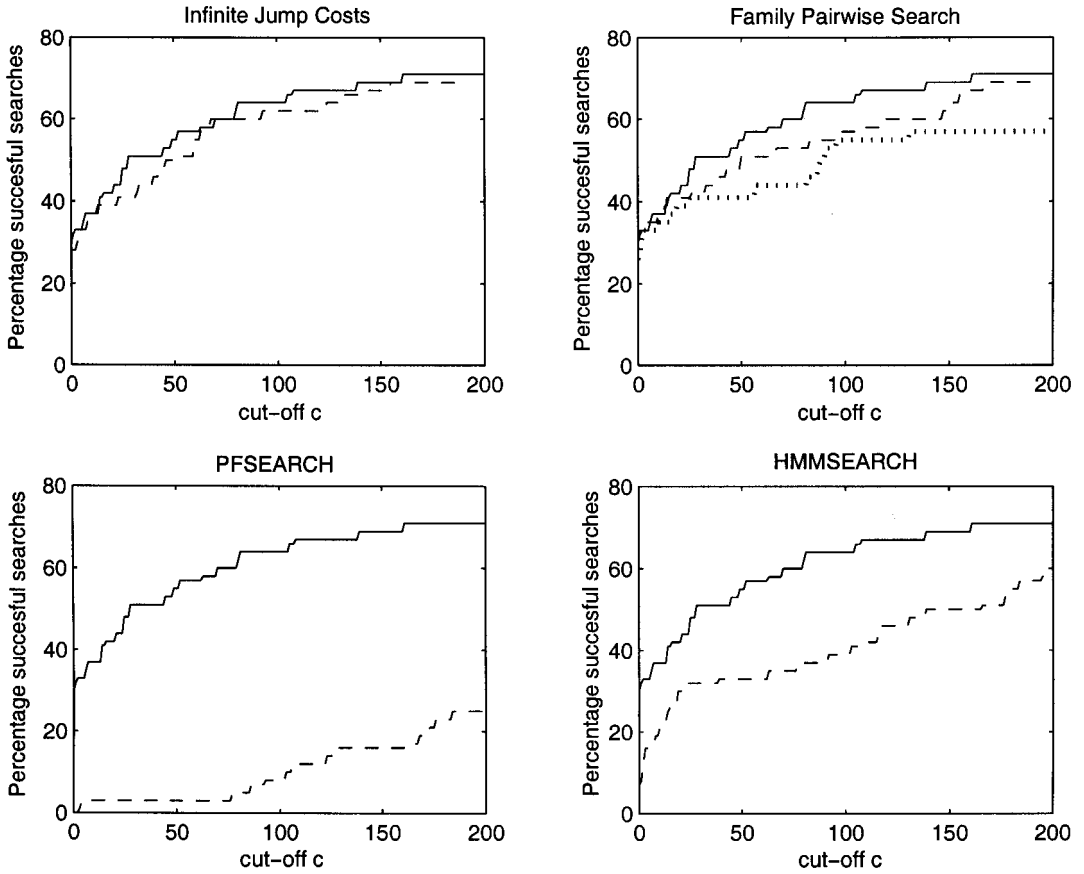

FIG. 6. Comparison of the overall performance of the jumping alignment algorithm and four other methods. The cut-off c is plotted versus the percentage of test sets with a median FP-count smaller or equal to c. The solid line in all plots shows the performance of JSEARCH with optimized jump costs. The dashed line represents JSEARCH with infinite jump costs (**top left**), family pairwise search using the Smith-Waterman algorithm (**top right**), PFSEARCH (**bottom left**), and HMMSEARCH (**bottom right**). The dotted line in the top right figure shows the performance of family pairwise search using BLAST.

TABLE 1. THE 56 TEST SETS FROM OUR EVALUATION DATA AND THE RESPECTIVE MED-FP-COUNTS FOR FPS IN THE SMITH-WATERMAN VARIANT (A), FPS IN THE BLAST VARIANT (B), PFSEARCH (C), HMMSERACH (D), JSEARCH (E), AND JSEARCH WITH INFINITE JUMP COST (F)[a]

		Size	*A*	*B*	*C*	*D*	*E*	*F*
	PH domain-like							
1	Phosphotyrosine-binding domain (PTB)	5	0	1561	970	993	1495	1395
2	Pleckstrin-homology domain (PH domain)	8	83	87	1227	834	14	8
	gamma-Crystallin-like							
3	Crystallins/Ca-binding development proteins	11	190	131	1767	131	45	43
	alpha/beta-Hydrolases							
4	Serine carboxypeptidase	5	10	2	168	63	25	60
5	Pancreatic lipase, N-terminal domain	6	13	214	348	318	70	4
6	Fungal lipases	8	0	0	292	19	16	3
	Scorpion toxin-like							
7	Plant defensins	5	39	57	176	14	0	0
8	Long-chain scorpion toxins	10	0	0	3	0	0	0
9	Short-chain scorpion toxins	23	0	0	706	13	28	40
	Ribonuclease H-like							
10	DnaQ-like 3'-5' exonuclease	5	67	1522	227	343	139	68
	Actin-like ATPase domain							
11	Hexokinase	7	147	592	707	499	694	514
12	Succinyl-CoA synthetase domains	11	50	301	659	395	850	766
	FAD/NAD(P)-binding domain							
13	FAD-linked reductases, N-terminal domain	9	0	19	173	3	25	33
14	FAD/NAD-linked reductases, N-terminal and central domains	26	15	82	916	860	105	32
15	Succinate dehydrogenase/fumarate reductase N-terminal domain	5	1	0	103	3	0	0
	Concanavalin A-like lectins/glucanases							
16	Xylanase/endoglucanase 12	8	155	57	703	166	251	399
17	Legume lectins	22	479	1172	583	476	221	349
18	Galectin (animal S-lectin)	8	2039	2065	1226	1154	1135	1378
	Cytochrome c							
19	monodomain cytochrome c	32	26	10	1305	340	7	9
	2Fe-2S ferredoxin-like							
20	2Fe-2S ferredoxin-related	14	0	0	1528	2	0	0
21	2Fe-2S ferredoxin domains from multidomain proteins	7	0	807	735	15	0	0
	Glutathione synthetase ATP-binding domain-like							
22	Biotin carboxylase/Carbamoyl phosphate synthetase	5	0	0	539	260	2	23
	Multiheme cytochromes							
23	Di-heme elbow motif	6	3	2	407	1	0	0
24	Cytochrome c3-like	8	0	1	1324	1180	0	0
	Cyclin-like							
25	Cyclin	6	855	636	1160	77	205	187
	Cystine-knot cytokines							
26	Neurotrophin	5	152	91	1145	271	161	133
27	Transforming growth factor (TGF)-beta	6	1165	671	1603	1354	108	93

TABLE 1. (*Continued*)

		Size	*A*	*B*	*C*	*D*	*E*	*F*
	DNA/RNA polymerases							
28	Reverse transcriptase	5	613	1484	123	117	508	842
29	DNA polymerase I	7	289	696	106	116	208	209
	(Trans)glycosidases							
30	Alanine racemase-like, N-terminal domain	7	382	300	184	22	21	218
31	alpha-Amylases, N-terminal domain	21	43	16	165	184	28	60
32	beta-glycanases	18	235	226	229	179	80	124
	Ribulose-phoshate binding barrel							
33	Tryptophan biosynthesis enzymes	5	33	90	495	306	52	52
	Porins							
34	Porin	5	117	25	215	195	14	64
	4Fe-4S ferredoxins							
35	Short-chain ferredoxins	5	0	0	1447	2	0	0
	Glyceraldehyde-3-phosphate dehydrogenase-like, C-terminal domain							
36	GAPDH-like	11	1281	278	940	715	623	931
	Histone-fold							
37	Nucleosome core histones	7	0	0	4	0	0	0
	4-helical cytokines							
38	Short-chain cytokines	8	792	956	1097	550	315	344
39	Interferons/interleukin-10 (IL-10)	9	108	96	548	177	460	516
40	Long-chain cytokines	10	793	839	425	109	480	714
	Viral coat and capsid proteins							
41	Animal virus proteins	52	98	88	475	289	63	143
	Glucocorticoid receptor-like (DNA-binding domain)							
42	LIM domain	11	0	0	2614	1919	0	0
43	Nuclear receptor	9	50	435	996	103	293	611
	lambda repressor-like DNA-binding domains							
44	Phage repressors	6	980	0	575	92	49	46
	Trypsin-like serine proteases							
45	Eukaryotic proteases	61	0	0	129	0	0	0
	Thioredoxin-like							
46	Thioltransferase	11	0	0	1741	139	0	0
47	Glutathione S-transferases, N-terminal domain	22	1877	1862	2164	2142	1589	1744
48	Glutathione peroxidase-like	5	1469	2202	526	375	81	155
	Metalloproteases ("zincins"), catalytic domain							
49	Matrix metalloproteases, catalytic domain	10	0	0	879	703	0	0
	EF-hand							
50	S100 proteins	6	0	0	1482	303	0	0
51	EF-hand modules in multidomain proteins	5	167	688	930	6	410	271
52	Parvalbumin	8	0	0	1138	19	0	0
53	Calmodulin-like	26	1	3	646	0	0	0
	PLP-dependent transferases							
54	AAT-like	9	159	205	86	39	0	4
55	Cystathionine synthase-like	6	403	793	93	10	6	63
56	omega-Amino acid:pyruvate aminotransferase-like	12	450	677	77	8	1	13

[a] Above a block of test families, the name of the corresponding superfamily is given.

There are also test sets where FPS methods seem to be the optimal method. In the overall evaluation of all 56 test sets, Figure 6 shows that jumping alignments compare favorably to hidden Markov models.

4. DISCUSSION

We have developed a new dynamic programming algorithm for detecting remote families 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. Our performance evaluation shows that it can compete with such established methods as profiles, profile hidden Markov models, and family pairwise search.

The idea of jumping alignments might suggest that remote homologues frequently are chimera of the other members of a protein family. Only a few of the data that we have examined, however, seem 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.

In our setting of searching protein databases for remote homologs of a given protein superfamily, we use the jumping alignment *score* as a measure of fit. We could also trace back the jumping alignment path, and our program actually allows one to do so. That would provide us with a new multiple alignment with the candidate sequence as an additional sequence aligned to the query alignment. However, this is only of interest if the candidate sequence belongs to the query superfamily. Our intention, in contrast, is to decide *whether* it belongs to the query.

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 at alternative residues in the current position in other sequences. The method cannot cope with long range correlations of residues that are in spatial vicinity in the folded protein.

It is instructive to look at the two extreme cases where one chooses either zero or infinite jump costs. 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 like in family pairwise searches. A combination is only given for intermediate jump costs. In our evaluations, zero jump costs performed significantly badly (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 restricted to conserved blocks of a protein family. In contrast to zero jump costs, infinite jump costs perform comparatively well. In this case, the jumping alignment algorithm still exploits information on expected gap positions, while vertical information is completely ignored.

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 do short ones, even if they are not related at all; see, for example, Karlin and Altschul (1990), Waterman and Vingron (1994), or Spang and Vingron (1998). In addition, we expect that phase transition laws exist for the jump costs as well as for the gap costs; compare Arratia and Waterman (1994). However, the setting seems to be much more complicated than in the case of pairwise sequence alignments. Simulations for every individual query alignment should 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 that 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 and Constantin Bannert for many helpful discussions. Two anonymous referees provided several helpful comments including how to simplify the dynamic programming recurrences in the case of affine gap costs.

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* 91, 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, 6073–6078.

Bucher, P., Karplus, K., Moeri, N., and Hofmann, K. 1996. A flexible search technique based on generalized profiles. *Comput. Chem.* 20, 3–24.

Eddy, S., Mitchison, G., and Durbin, R. 1995. Maximum discrimination hidden Markov models of sequence consensus. *J. Comp. Biol.* 2, 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, 4355–4358.

Gribskov, M., and Robinson, N. L. 1996. The use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. *Computers and Chemistry* 20(1), 25–34.

Grundy, W. 1998. Homology detection via family pairwise search. *J. Comp. Biol.* 5, 479–492.

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

Jaakkola, T., Diekhans, M., and Haussler, D. 2000. A discriminative framework for detecting remote protein homologies. *J. Comp. Biol.* 7, 95–114.

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. *Proc. 1st Ann. Int. Conf. Computational Molecular Biology (RECOMB 97)* 174–180.

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, 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.

Lo Conte, L., Ailey, B., Hubbard, T., Brenner, S., Murzin, A., and Chothia, C. 2000. SCOP: A structural classification of proteins database. *Nucl. Acids. Res.* 28, 257–259.

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

Morgenstern, B., Frech, K., Dress, A., and Werner, T. 1998. DIALIGN: Finding local similarities by multiple sequence alignment. *Bioinformatics* 14, 290–294.

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, 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, 349–354.

Rehmsmeier, M., and Vingron, M. 2001. Phylogenetic Information Improves Homology Detection. *Prot. Struct. Func. Genet.* 45(4), 360–371.

Rehmsmeier, M. 2002. Phase4—automatic evaluation of database search methods. *Briefings in Bioinformatics*, to appear.

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

Spang, R., Rehmsmeier, M., and Stoye, J. 2000. Sequence database search using jumping alignments. *Proc. 8th Int. Conf. Intelligent Systems for Molecular Biology (ISMB 00)* 367–375.

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, 4673–4680.

Waterman, M., and Vingron, M. 1994. Sequence comparison significance and Poisson approximation. *Stat. Sci.* 9, 367–381.

Address correspondence to:
Jens Stoye
Universität Bielefeld
Technische Fakultät
AG Genominformatik
D33594 Bielefeld, Germany

E-mail: stoye@techfak.uni-bielefeld.de