Learning models for aligning protein sequences with predicted secondary structure

Eagu Kim*, Travis Wheeler, and John Kececioglu

{egkim,twheeler,kece}@cs.arizona.edu
Department of Computer Science
The University of Arizona, Tucson AZ 85721, USA

Abstract Accurately aligning distant protein sequences is notoriously difficult. A recent approach to improving alignment accuracy is to use additional information such as predicted *secondary structure*. We introduce several new models for scoring alignments of protein sequences with predicted secondary structure, which use the predictions and their confidences to modify both the substitution and gap cost functions. We present efficient algorithms for computing optimal pairwise alignments under these models, all of which run in near-quadratic time. We also review an approach to learning the values of the parameters in these models called *inverse alignment*. We then evaluate the accuracy of these models by studying how well an optimal alignment under the model recovers known benchmark reference alignments. Our experiments show that using parameters learned by inverse alignment, these new secondary-structure-based models provide a significant improvement in alignment accuracy for distant sequences. The best model improves upon the accuracy of the standard sequence alignment model for pairwise alignment by as much as 15% for sequences with less than 25% identity, and improves the accuracy of multiple alignment by 20% for difficult benchmarks whose average accuracy under standard tools is less than 40%.

Keywords Sequence alignment, protein secondary structure, inverse parametric alignment, substitution score matrices, affine gap penalties.

1 Introduction

While sequence alignment is one of the most basic and well-studied tasks in computational biology, accurate alignment of distantly-related protein sequences remains notoriously difficult. Accurately aligning such sequences usually requires multiple sequence alignment, and a succession of ideas have been employed by modern multiple alignment tools to improve their accuracy, including: (a) *hydrophobic gap penalties*, which modify the alignment score to avoid gaps in regions that may be in the structural core, and are employed by `CLUSTAL W` [34], `T-Coffee` [28], and `MUSCLE` [10]; (b) *polishing*, which refines the alignment by

* Corresponding author. Current address: egkim@biostat.wisc.edu, Department of Biostatistics and Medical Informatics, University of Wisconsin, Madison WI 53706.

realigning subgroups, and is used by MAFFT [18], MUSCLE, ProbCons [7], and Opal [36]; and (c) *consistency*, both in its combinatorial [28] and probabilistic [9, 7] settings, which favors matches in the alignment that have high support, and is employed by T-Coffee, MAFFT, ProbCons, SPEM [39], PROMALS [29], and ISPAlign [23]. These techniques all operate on the input sequences alone, and do not require additional sources of information.

Recent techniques that recruit additional information beyond the input sequences, and that have afforded significant boosts in accuracy, include: (d) *sequence profiles*, which augments the input residues with profiles of amino acid exchanges from closely related sequences found through database searches, and is used by SPEM, PRALINE [32], PROMALS, and ISPAlign; (e) *intermediate sequences*, which adds new sequences to the alignment that link distant input sequences through chains of similar sequences, and is employed by MAFFT and ISPAlign; and (f) predicted *secondary structure*, which annotates the input residues with their predicted structural type and modifies the scoring of alignments to take these types into account, used by SPEM, PRALINE, PROMALS, and ISPAlign. These latter techniques, including secondary structure prediction, all involve database searching to find sequences that are closely related to the input sequences, which adds considerable overhead to the running time for alignment.

Of these latter techniques, incorporating secondary structure seems to have the greatest scope for further improvement. The scoring models based on secondary structure employed by current tools [39, 32, 29, 23] do not make full use of predicted structural information when modifying the scores of substitutions and gaps. Furthermore, recent advances in single-sequence secondary structure prediction [1] suggest it may eventually be possible to make sufficiently accurate predictions based on the input sequences alone, which would yield improved alignment accuracy without the slowdown caused by sequence database searches.

In this paper, we introduce several new models for protein sequence alignment based on predicted secondary structure, and show they yield a substantial improvement in accuracy for distant sequences. These models improve on those used by current tools in several ways. They explicitly take into account the confidences with which structural types are predicted at a residue, and use these confidences in a rigorous way to modify both the scores for substitutions and the penalties for gaps that disrupt the structure. Furthermore, optimal alignments under these new models can be efficiently computed. Prior models tend to either have a limited number of ad hoc parameters that are set by hand [39, 32, 23], or have a large number of parameters that are estimated by counting frequencies of observed events in comparatively small sets of reference alignments [24, 29, 33]. Our new models have multiple parameters whose values must be set, and we show that recently developed techniques for parameter learning [19, 21, 22] can be used to find values for these parameters that are optimal in a well-defined sense. Finally, experimental results using our models show that our best model improves on the accuracy of the standard model by 15% for pairwise alignment of sequences with less than 25% identity, and by 20% for multiple alignment of difficult benchmarks whose accuracy under standard tools is less than 40%.

Related work Structure is often conserved among related proteins even when sequence similarity is lost [31], so incorporating structure has the potential to improve the accuracy of aligning distant sequences. While deducing full three-dimensional structure from protein sequences remains challenging, accurate tools are available for predicting secondary structure from protein sequences [17]. Predicted secondary structure can be incorporated into the alignment model by encouraging substitutions between residues of the same secondary structure type, and discouraging gaps that disrupt regions of secondary structure.

The first work on incorporating secondary structure into the alignment scoring model appears to be by Lüthy, McLachlan and Eisenberg [24], who applied the log-odds scoring methodology of Dayhoff et al. [6] to derive substitution score matrices that take both the amino acids and the secondary structure types of the aligned residues into account when scoring a substitution. For the three secondary structure types of α-helix, β-strand, and other, Lüthy et al. derive three log-odds substitution score matrices, where a given matrix applies when both of the aligned residues have the *same* structural type. In particular, their work does not provide matrices that apply to substitutions between residues with differing structural types. Moreover, the log-odds methodology for substitution scores does not provide appropriate gap penalties for these matrices.

Modern multiple alignment tools that take predicted secondary structure into account include `PRALINE` [32], `SPEM` [39], `PROMALS` [30], and `ISPAlign` [23]. `PRALINE` uses the three substitution matrices of Lüthy et al. [24] when aligned residues have the same predicted structure type, plus the `BLOSUM62` matrix [16] when they have differing types. `PRALINE` also employs four pairs of hand-chosen affine gap penalties (a gap open and extension penalty), one pair per matrix. `SPEM` modifies a standard substitution matrix by adding a bonus of x to the substitution score when aligned residues have the same type, and a penalty of $-x$ when they have differing types, where x is a single hand-chosen constant. No gaps are permitted following a residue predicted to be in an α-helix or β-strand; otherwise, a fixed gap open and extension penalty is used. `PROMALS` employs a hidden Markov model approach where match, insert, and delete states also emit secondary structure types. Emission and transition probabilities for each of these states are set by counting frequencies of events in a collection of reference alignments. When scoring an alignment by the logarithm of its emission and transition probabilities, the relative contribution to the alignment score of secondary structure emission probabilities versus amino acid emission probabilities is controlled by a hand-chosen weighting constant. `ISPAlign` uses the hidden Markov model of `ProbCons` [7], where match states are modified to emit secondary structure types. The probability of emitting a pair of the same type is set to x for all three types, while the probability for a pair of differing types is $1-x$, where x is a single hand-chosen constant. The transition probabilities of `ProbCons` that correspond to gap open and extension penalties are used, shifted by factor y to compensate for the effect of the structure type emission probabilities on the substitution score, where y is a second hand-chosen constant. In increasing accuracy, these tools are ranked: `PRALINE`, `SPEM`, `PROMALS`, and `ISPAlign`.

In contrast to the above approaches, we develop general scoring models with no ad hoc parameters that modify substitution scores on the basis of the pair of secondary structure states of the aligned residues, and that use a suite of affine gap penalties whose values depend on the degree of secondary structure in the region disrupted by the gap. A unique feature is that at each residue we also take into account the *confidence* of the prediction for the three structure types. (Confidences are output by tools such as `PSIPRED` [17].) While our models are more complex, especially in how gap costs are determined, we show that optimal pairwise alignments under the models can still be computed efficiently. And though these models have many parameters, we rigorously learn their values using inverse alignment [19, 21, 22] applied to training sets of reference alignments.

Overview In Section 2, we present several new models for scoring alignments of protein sequences based on their predicted secondary structure. Section 3 develops efficient algorithms for computing optimal alignments of two sequences under these models. Section 4 reviews an approach called inverse alignment that uses these optimal alignment algorithms to learn parameter values for the models from examples of biological reference alignments. Section 5 then presents experimental results that compare the accuracy of these models when used for both pairwise and multiple alignment on standard benchmark reference alignments.

2 Scoring alignments with predicted secondary structure

We now introduce several models for scoring an alignment of two protein sequences A and B that make use of predicted secondary structures for A and B. These models score an alignment $\mathcal{A}$ by specifying a cost function $f(\mathcal{A})$, where an optimal alignment minimizes f. The features of an alignment that f scores are: (i) substitutions of pairs of residues, (ii) internal gaps, and (iii) external gaps. A *gap* is a maximal run of either insertions or deletions. A gap is *external* if it inserts or deletes a prefix or a suffix of A or B; otherwise it is *internal*.

Substitutions and gaps are scored in a position-dependent manner that takes into account the predicted secondary structure. A *substitution* in alignment $\mathcal{A}$ of residues $A[i]$ and $B[j]$ is denoted by a tuple (A, i, B, j). An *internal gap* is denoted by a tuple (S, i, j, T, k), where substring $S[i:j]$ is deleted from sequence S and inserted between positions k and $k+1$ of sequence T. An *external gap* is denoted by a pair (i, j), where prefix or suffix $S[i:j]$ is deleted from A or B.

In general, the alignment cost function $f(\mathcal{A})$ uses two other functions: function s for the cost of substitutions, and function g for the cost of internal gaps. External gaps use standard *affine* gap costs [12]. The general form of f is then

$$f(\mathcal{A}) \;:=\; \sum_{\substack{\text{substitutions} \\ (A,i,B,j)\,\in\,\mathcal{A}}} s(A,i,B,j) \;+\; \sum_{\substack{\text{internal gaps} \\ (S,i,j,T,k)\,\in\,\mathcal{A}}} g(S,i,j,T,k) \;+\; \sum_{\substack{\text{external gaps} \\ (i,j)\,\in\,\mathcal{A}}} \Big(\widetilde{\gamma} \;+\; \big(j-i+1\big)\,\widetilde{\lambda}\Big), \tag{1}$$

where $\widetilde{\gamma}, \widetilde{\lambda}$ are the respective gap *open* and *extension* penalties for external gaps. We next describe substitution cost function s, and internal gap cost function g.

2.1 Scoring substitutions

Consider a substitution of two residues in an alignment, where these residues have amino acids a and b, and are involved in secondary structures of types c and d. For secondary structures, we consider the standard three types of alpha-helix, beta-strand, and loop, which we represent by the symbols α, β, and ϕ, respectively. In the following, $\Gamma = \{\alpha, \beta, \phi\}$ denotes the alphabet of secondary structure types, and Σ denotes the alphabet of amino acids.

Function s scores a substitution of amino acids $a, b \in \Sigma$ with secondary structure types $c, d \in \Gamma$ using two costs: $\sigma(a, b)$, the cost for *substituting* amino acids a and b, and $\mu(c, d)$, an additive *modifier* for the residues having secondary structure types c and d. Values $\sigma_{ab} = \sigma(a, b)$ and $\mu_{cd} = \mu(c, d)$ are parameters to our substitution model. In the model, both a, b and c, d are unordered pairs. This results in 210 substitution costs σ_{ab}, plus 6 secondary structure modifiers μ_{cd}, for a total of 216 parameters that must be specified for the substitution model.

We consider two forms of prediction, which we call *lumped* or *distributed*.

Lumped prediction In lumped prediction, which is a special case of distributed prediction, the prediction at each residue is a *single* secondary structure type. The predicted secondary structure for protein sequence A can then be represented by a string S_A, where residue i of A has predicted type $S_A[i] \in \Gamma$.

For lumped prediction, the substitution cost function s is

$$s(A, i, B, j) \;:=\; \sigma\big(A[i], B[j]\big) \;+\; \mu\big(S_A[i], S_B[j]\big). \tag{2}$$

The modifer $\mu(c, d)$ may be positive or negative. When the residues have the same secondary structure type, $\mu(c, c) \leq 0$, which makes it more favorable to align the residues. When the residues have different types, $\mu(c, d) \geq 0$, making it less favorable to align them. These constraints on the modifiers can be enforced during parameter learning, as described in Section 4.

Distributed prediction The most accurate tools for secondary structure prediction, such as `PSIPRED` [17], output a *confidence* that the residue is in each possible type. For residue i of sequence A, we denote the predictor's confidence that the residue is in secondary structure type c by $P_A(i, c) \geq 0$. In practice, we normalize the confidences output by the predictor at each residue i to obtain a *distribution* with $\sum_{c \in \Gamma} P_A(i, c) = 1$.

For distributed prediction, the substitution cost function s is

$$s(A, i, B, j) \;:=\; \sigma\big(A[i], B[j]\big) \;+\; \sum_{c, d \,\in\, \Gamma} P_A(i, c)\; P_B(j, d)\; \mu(c, d). \tag{3}$$

When the predictor puts all its confidence on one structure type at each residue, this reduces to the lumped prediction substitution function.

2.2 Scoring gaps

With standard *affine* gap costs [12], the cost of inserting or deleting a substring of length k is $\gamma + \lambda k$, where γ and λ are respectively the gap open and extension costs. The new gap scoring models generalize this to a suite of gap open and extension costs whose values depend on the secondary structure around the gap. The basic idea is that the gap open cost γ depends on a *global* measure of how disruptive the entire gap is to the secondary structure of the proteins, while the gap extension cost λ charged per residue depends on a *local* measure of disruption at that residue's position. We define these notions more precisely below.

For an internal gap that deletes the substring $S[i:j]$, and inserts it after the residue $T[k]$, the gap cost function g has the general form,

$$g(S,i,j,T,k) \;:=\; \gamma\Big(H(S,i,j,T,k)\Big) \;+\; \sum_{i \le p \le j} \lambda\Big(h(S,p,T,k)\Big).$$

The first term is a per-gap cost, and the second term is a sum of per-residue costs. Functions H and h are respectively the global and local measures of secondary structure disruption. Both H and h return integer values in the range $L = \{1, 2, \ldots, \ell\}$ that give the discrete *level* of disruption. The corresponding values for the gap open and extension costs, $\gamma_i = \gamma(i)$ and $\lambda_i = \lambda(i)$ for $i \in L$, are parameters to our model. For ℓ levels, the internal gap cost model has 2ℓ parameters γ_i and λ_i that must be specified.

The gap costs at these levels satisfy $0 \le \gamma_1 \le \ldots \le \gamma_\ell$ and $0 \le \lambda_1 \le \ldots \le \lambda_\ell$. In other words, a higher level of disruption incurs a greater gap cost. These constraints are enforced during parameter learning, as described in Section 4.

Functions H and h, which give the level of secondary structure disruption, depend on two aspects: (i) how strongly a position is involved in secondary structure, and (ii) which positions are considered when determining the level. We call the first aspect the *degree* of secondary structure, and the second aspect the *context* of the gap.

Measuring the degree of secondary structure As described in Section 2.1, the predicted secondary structure for the residues of a protein sequence A may be represented by string S_A of structure types in the case of lumped prediction, or vector P_A of confidences for each type in the case of distributed prediction. We consider three ways of using such predictions to determine the degree of secondary structure at residue position i in sequence A. This *degree* $\Psi_A(i)$ is in the range $[0,1]$, where 0 corresponds to no involvement in secondary structure, and 1 corresponds to full involvement.

(1) The *lumped-binary* approach assumes a lumped prediction, and produces a binary value for the degree: $\Psi_A(i)$ is 1 if $S_A[i] \in \{\alpha, \beta\}$, and 0 otherwise.

(2) The *distributed-binary* approach assumes a distributed prediction, and produces a binary degree: $\Psi_A(i) = 1$ iff $P_A(i,\alpha) + P_A(i,\beta) > P_A(i,\phi)$.

(3) The *distributed-continuous* approach assumes a distributed prediction, and produces the real value $\Psi_A(i) = P_A(i,\alpha) + P_A(i,\beta)$.

Specifying the gap context The gap context is specified by the positions that functions H and h consider when measuring the global and local secondary structure level. Both functions use the above secondary structure degree $\Psi(i)$.

To measure the local level h at position i in a sequence S of length n, we consider a small window $W(i,n)$ of consecutive positions centered around i:

$$h(S,i) \;\;:=\;\; \left\langle \sum_{p \,\in\, W(i,n)} \Psi_S(p) \Big/ \Big|W(i,n)\Big| \right\rangle.$$

Here the notation $\langle x \rangle$ maps real value $x \in [0,1]$ to the discrete levels $1, 2, \ldots, \ell$ by $\langle x \rangle := \lfloor (\ell - 1)x \rfloor + 1$. In words, the local level h at position i is the average secondary structure degree Ψ for the residues in a window around i. Generally all windows have the same width $\big|W(i,n)\big| = w$, except if i is too close to 1 or n to be centered in a window of width w, then $W(i,n)$ shrinks on one side of i.

In our experiments in Section 5, we consider three ways of specifying the gap context, depending on whether we take an insertion view, a deletion view, or a mixed view of a gap. (The same context applies to all gaps in an alignment.)

(1) The *deletion context* views the disruption caused by the gap in terms of the secondary structure lost by deleting substring $S[i\!:\!j]$. For the global measure of disruption, this context takes the maximum local level of secondary structure over the positions in the deleted substring, which gives the gap cost function,

$$g(S,i,j,T,k) \;\;:=\;\; \gamma\Big(\max_{i \le p \le j} h(S,p)\Big) \;+\; \sum_{i \le p \le j} \lambda\Big(h(S,p)\Big). \tag{4}$$

(2) The *insertion context* views the disruption in terms of the secondary structure displaced at residues $T[k]$ and $T[k+1]$ where the insertion occurs. For both the global and local measures of disruption this context uses

$$H(T,k) \;\;:=\;\; \Big\langle \tfrac{1}{2}\, h(T,k) \;+\; \tfrac{1}{2}\, h(T,k{+}1) \Big\rangle,$$

which gives the gap cost function,

$$g(S,i,j,T,k) \;\;:=\;\; \gamma\Big(H(T,k)\Big) \;+\; (j{-}i{+}1)\,\lambda\Big(H(T,k)\Big). \tag{5}$$

(3) The *mixed context* combines the above global measure H of the insertion context with the local measure h of the deletion context, which gives

$$g(S,i,j,T,k) \;\;:=\;\; \gamma\Big(H(T,k)\Big) \;+\; \sum_{i \le p \le j} \lambda\Big(h(S,p)\Big). \tag{6}$$

To summarize, the *parameters* of the scoring model for protein sequence alignment are the 210 substitution costs σ_{ab}, the 6 substitution modifiers $\mu_{\underset{\sim}{cd}}$, the 2ℓ gap costs γ_i and λ_i for internal gaps, and the two gap costs $\widetilde{\gamma}$ and $\widetilde{\lambda}$ for external gaps. This is a total of $218 + 2\ell$ parameters. In general, the model depends on the window width w, the number of levels ℓ, whether the secondary structure prediction is lumped or distributed, the choice of measure Ψ for the degree of secondary structure, and the choice of gap context.

3 Computing optimal alignments efficiently

We can efficiently compute an optimal alignment of sequences A and B under scoring function f given by equation (1) using dynamic programming. Let $C(i,j)$ be the cost of an optimal alignment of prefixes $A[1\!:\!i]$ and $B[1\!:\!j]$. This alignment ends with either a *substitution* of residues $A[i]$ and $B[j]$, or a *gap* involving substring $A[k\!:\!i]$ or $B[k\!:\!j]$ for some k. In each case, the alignment must be preceded by an optimal solution over shorter prefixes. This leads to the recurrence,

$$C(i,j) \;:=\; \min \begin{cases} C(i-1,j-1) \;+\; s(A,i,B,j), \\ \min\limits_{1\le k\le i} \Big\{ C(k-1,j) \;+\; g(A,k,i,B,j) \Big\}, \\ \min\limits_{1\le k\le j} \Big\{ C(i,k-1) \;+\; g(B,k,j,A,i) \Big\}. \end{cases} \tag{7}$$

(To simplify the presentation, this ignores boundary conditions and the special case of external gaps.) For two sequences of length n, the straightforward algorithm that directly evaluates this recurrence in a table, and recovers an optimal alignment using the table, takes $\Theta(n^3)$ time. (This assumes evaluating g takes $O(1)$ time, which can be achieved through preprocessing, as discussed later.)

By studying gap cost function g, the time to compute an optimal alignment for all three gap contexts can be reduced to nearly $O(n^2)$, as discussed next.

Insertion and mixed contexts For the *insertion context*, function g given by equation (5) is very close to a standard affine gap cost function. The same technique developed by Gotoh [12] for standard affine gap costs, namely (1) keeping track of three separate quantities at each entry (i,j) of the dynamic programming table, depending on whether the alignment ends by a substitution, insertion, or deletion, and (2) considering the last two columns of an alignment to decide whether or not the gap open cost should be charged, can be used to reduce the total time to compute an optimal alignment to $O(n^2)$. For the *mixed context*, given by equation (6), this same approach also leads to an $O(n^2)$ time algorithm.

Deletion context For the *deletion* context, function g given by equation (4) involves a maximization to determine the gap open cost, which complicates matters. For this context, the total time can be sped up significantly using the *candidate list technique* originally developed for alignment with convex gap costs by Miller and Myers [26] and Galil and Giancarlo [11]. While our gap cost function g is not convex in their original sense, their technique still applies. We briefly review its ideas below.

The candidate list technique speeds up the evaluation of the two inner minimizations in equation (7) for $C(i,j)$. The minimization involving $g(B,k,j,A,i)$ can be viewed as computing the function

$$F_i(j) \;:=\; \min_{1\le k\le j} \Big\{ G_i(k,j) \Big\},$$

where $G_i(k,j) := C(i,k-1) + g(B,k,j,A,i)$. Similarly, the minimization involving $g(A,k,i,B,j)$ can be viewed as computing a function $\widetilde{F}_j(i)$ that is the minimum of another function $\widetilde{G}_j(k,i)$. At a high level, when filling in row i of the table for $C(i,j)$, the candidate list approach maintains a data structure for row i that enables fast computation of $F_i(j)$ across the row for increasing j. Similarly, it maintains a separate data structure for each column j that enables fast computation of $\widetilde{F}_j(i)$ down the column. When processing entry (i,j) of the dynamic programming table, the data structures for row i and column j permit evaluation of $F_i(j)$ and $\widetilde{F}_j(i)$ in $O(\log n)$ amortized time. Evaluating the recurrence at the $O(n^2)$ entries of the table then takes a total of $O(n^2 \log n)$ time. A very readable exposition is given by Gusfield [15, pp. 293–302].

More specifically, for a fixed row i the candidate list technique computes $F(j)$ as follows. (When i is fixed, we drop subscript i on F_i and G_i.) Each index k in the minimization of $G(k,j)$ over $1 \le k \le j$ is viewed as a *candidate* for determining the value of $F(j)$. Candidate k contributes the *curve* $G(k,j)$, which is viewed as a function of j for $j \ge k$. Geometrically, the set of values of $F(j)$ for $1 \le j \le n$, which is the minimum of these curves, is known as their *lower envelope* [4].

When computing $F(j)$ across the row for each successive j, a representation of this lower envelope is maintained at all $j' \ge j$, only considering curves for candidates k with $k \le j$. This representation for $j' \ge j$ is a partition of the interval $[j,n]$ into maximal subintervals such that across each subinterval the minimum is given by exactly one curve. The process of adding a candidate's curve to the lower envelope exploits the following property of the gap cost function.

Lemma 1 (Dominance property) *Consider candidates a and b with $a < b$. Suppose $G(a,c) < G(b,c)$ at some $c \ge b$. Then $G(a,d) < G(b,d)$ at all $d \ge c$.*

Proof The key is to show that the difference

$$\Big(G(b,d) - G(b,c)\Big) - \Big(G(a,d) - G(a,c)\Big) \tag{8}$$

is nonnegative, as adding $G(b,c) - G(a,c)$ implies the lemma. For the deletion context, quantity (8) above equals

$$\Big(\gamma\big(H(S,b,d)\big) - \gamma\big(H(S,b,c)\big)\Big) - \Big(\gamma\big(H(S,a,d)\big) - \gamma\big(H(S,a,c)\big)\Big), \tag{9}$$

where function $H(S,x,y)$ is the maximum of $h(S,p)$ over the interval $p \in [x,y]$. Considering where $h(S,p)$ attains its maximum on the intervals $[b,d]$, $[b,c]$, $[a,d]$, $[a,c]$, and noting that $\gamma\big(\langle x \rangle\big)$ is nondecreasing in x, shows quantity (9) above is nonnegative, which proves the lemma. $\square$

A key consequence of Lemma 1 is that the curves for any pair of candidates cross *at most once.* More precisely, for a pair $a < b$ of candidates and an interval x, either one candidate dominates the other across x, or x can be split into two pieces y, z such that the later candidate b dominates on the left piece y, and the earlier candidate a dominates on the right piece z. Using this property, we update the lower envelope as follows.

Given a new candidate j, we compare it on intervals of the current partition $p_1, \ldots, p_t$, starting with the leftmost. When comparing against the ith interval $x = (p_{i-1}, p_i]$, we first examine its right endpoint $c = p_i$. If $G(j, c) \leq G(k_i, c)$, then candidate j dominates across x, so we *delete* interval i from the partition (effectively merging it with the next interval), and continue comparing j against interval $i+1$. If $G(j, c) > G(k_i, c)$, then by Lemma 1, j is dominated on intervals $i+1, \ldots, t$ by their corresponding candidates, so those intervals do not change in the partition. Interval i may change, though if it does, it at worst splits into two pieces with j dominating in the left piece. In the case of a split, we *insert* a new leftmost interval $[j, p]$ into the partition with j as the corresponding candidate. To find the split point p (if it exists), we can use binary search to identify the rightmost position such that $G(j, p) < G(k, p)$, where k is the candidate corresponding to the current interval. (The proof of Lemma 1 implies that the difference between the curves for candidates k and j is nonincreasing.) During the binary search, gap cost function g is evaluated $O(\log n)$ times.

In general, updating the partition when considering a new candidate involves a series of deletes, followed by an insert. Assuming g can be evaluated in $O(1)$ time, a delete takes $O(1)$ time while an insert takes $O(\log n)$ time. While a given update can involve several deletes, each delete removes an earlier candidate, which is never reinserted. Charging the delete to the removed candidate, the total time for deletes is then $O(n^2)$. The total time for all inserts is $O(n^2 \log n)$.

The final issue is the time to evaluate $g(S, i, j, T, k)$ for the deletion context. With $O(n)$ time preprocessing, we can evaluate the sum of gap extension penalties λ in the definition of g in $O(1)$ time, by taking the difference of two precomputed prefix sums. With $O(n^2)$ time and space preprocessing, we can look up the gap open penalty γ in $O(1)$ time, by precomputing $H(S, i, j)$ for all i, j. (Alternately, we can use a *range tree* [4] to find the maximum for $H(S, i, j)$ online when evaluating the cost of a gap; this only requires $O(n)$ time and space preprocessing, but it evaluates g in $O(\log n)$ time, which is slightly slower.) In short, using $O(n^2)$ time and space preprocessing, we can evaluate g in $O(1)$ time.

We summarize the total time for the alignment algorithm below.

Theorem 1 (Deletion context running time) *For the deletion gap context, optimally aligning two sequences of lengths $m \leq n$ takes $O(n^2 + mn \log n)$ time.*

This provides a significant speedup in practice as well as in theory.

4 Learning model parameters by inverse alignment

We now show how to learn values for the parameters of the alignment scoring models using the *inverse alignment* approach developed in Kim and Kececioglu [21], modified to incorporate a loss function as introduced by Yu, Joachims, Elber and Pillardy [38]. The goal of parameter learning is to find values for which the optimal scoring alignment is the biologically correct alignment. Inverse alignment takes as input a collection of *examples* of correct reference alignments, and outputs an assignment of values for the parameters that makes the reference alignments score as close as possible to optimal scoring alignments of their sequences. The approach of Kim and Kececioglu [21] finds a parameter assignment that minimizes the average absolute error in score across the examples, while Yu et al. [38] also incorporate the error in recovery between an optimal scoring alignment and the reference alignment. We briefly review these approaches below.

The reference alignments of protein sequences that are widely available for parameter learning, which are generally based on aligning the known three-dimensional structures of families of related proteins, are actually only partial alignments. In a *partial alignment*, columns are labeled as reliable or unreliable. Reliable columns typically correspond to core blocks, which are gapless regions of the alignment where common three-dimensional structure is shared across the family. At unreliable columns, the alignment is effectively left unspecified. In a *complete alignment*, all columns are reliable. We follow the presentation of Kim and Kececioglu [22], which first develops an algorithm for inverse alignment from examples that are complete alignments, and then extends it to partial examples.

Complete examples In the following, we define inverse alignment under a new optimization criterion that we call *discrepancy.* The discrepancy criterion generalizes the score error approach of Kim and Kececioglu [21] and the recovery error approach of Yu et al. [38]. In our experiments, it is superior to both.

For the alignment scoring function f, let $p_1, p_2, \ldots, p_t$ be its parameters. We view the entire set of parameters as a vector $p = (p_1, \ldots, p_t)$ drawn from domain $\mathcal{D}$. When we want to emphasize the dependence of f on its parameters p, we write f_p. The input to inverse alignment consists of many example alignments $\mathcal{A}_i$, where each example aligns a corresponding pair of sequences $\mathcal{S}_i$. We denote the average length of the sequences in $\mathcal{S}_i$ by $||\mathcal{S}_i||$.

For an alignment $\mathcal{B}_i$ of sequences $\mathcal{S}_i$ for example $\mathcal{A}_i$, let function $d(\mathcal{A}_i, \mathcal{B}_i)$ be the fraction of reliable columns of example $\mathcal{A}_i$ that are *not* present in alignment $\mathcal{B}_i$. In other words, function d measures the error in recovering example $\mathcal{A}_i$ by alignment $\mathcal{B}_i$. Yu et al. [38] call d a loss function.

Definition 1 (Inverse alignment with complete examples) *Inverse Alignment* from complete examples under the discrepancy criterion is the following problem. The input is a collection of complete alignments $\mathcal{A}_i$ of sequences $\mathcal{S}_i$ for $1 \le i \le k$. The output is parameter vector $x^* := \mathrm{argmin}_{x \in \mathcal{D}} D(x)$, where

$$D(x) \;:=\; \frac{1}{k} \sum_{1 \le i \le k} \max_{\mathcal{B}_i} \left\{ (1-\alpha)\, \frac{f_x(\mathcal{A}_i) \;-\; f_x(\mathcal{B}_i)}{||\mathcal{S}_i||} \;+\; \alpha\, d(\mathcal{A}_i, \mathcal{B}_i) \right\}. \tag{10}$$

In the above, the max is over all alignments $\mathcal{B}_i$ of $\mathcal{S}_i$, and $\alpha \in [0,1]$ is a constant that controls the relative weight on score error versus recovery error. Function $D(x)$ is the *average discrepancy* of the examples under parameters x. □

The discrepancy criterion reduces to the approach of Kim and Kececioglu [21] when $\alpha = 0$, and to the approach of Yu et al. [38] when $\alpha = \frac{1}{2}$, upon removing the length normalization of score error by setting $||\mathcal{S}_i|| \equiv 1$. Intuitively, when minimizing discrepancy $D(x)$, the recovery error term $d(\mathcal{A}_i, \mathcal{B}_i)$ drives the alignments $\mathcal{B}_i$ that score better than example $\mathcal{A}_i$ (which have a positive score error term under f_x) toward also having low recovery error d. In other words, the recovery error term helps make the best scoring alignments agree with the example on its columns. On the other hand, the scale of recovery error $d \in [0,1]$ is small, while the score error $f_x(\mathcal{A}_i) - f_x(\mathcal{B}_i)$ can grow arbitrarily big, especially for long sequences. So correctly tuning the relative contribution of score error and recovery error is impossible for examples $\mathcal{A}_i$ of varying lengths, unless score error is length normalized, which leads to the above discrepancy formulation.

For the alignment scoring functions f presented in Section 2, inverse alignment from complete examples can be reduced to linear programming. The parameters of scoring function f_x are the variables x of the linear program. The domain $\mathcal{D}$ of the parameters is described by a set of inequalities that includes the bounds $(0,-1,0,0) \leq (\sigma_{ab}, \mu_{cd}, \gamma_i, \lambda_i) \leq (1,1,1,1)$. The description of $\mathcal{D}$ also contains the inequalities $\sigma_{aa} \leq \sigma_{ab}$, $\mu_{cc} \leq 0$, $\mu_{cd} \geq 0$, $\gamma_i \leq \gamma_{i+1}$, and $\lambda_i \leq \lambda_{i+1}$.

The remaining inequalities measure discrepancy $D(x)$. Associated with each example $\mathcal{A}_i$ is an error variable δ_i. For example $\mathcal{A}_i$, and *every alignment* $\mathcal{B}_i$ of sequences $\mathcal{S}_i$, the linear program has an inequality

$$\big(1-\alpha\big)\,\frac{f_x(\mathcal{A}_i) \;-\; f_x(\mathcal{B}_i)}{||\mathcal{S}_i||} \;\;+\;\; \alpha\; d\big(\mathcal{A}_i, \mathcal{B}_i\big) \;\;\leq\;\; \delta_i. \tag{11}$$

Note this is a linear inequality in the variables x, since function f_x is linear in x.

The objective function for the linear program is to minimize $\frac{1}{k}\sum_{1 \leq i \leq k} \delta_i$. Minimizing this objective forces each δ_i to equal the term with index i in the summation for $D(x)$ in equation (10). Thus, an optimal solution x^* to the linear program minimizes the average discrepancy of the examples.

This linear program has an exponential number of inequalities of form (11), since for an example $\mathcal{A}_i$, the number of alignments $\mathcal{B}_i$ of $\mathcal{S}_i$ is exponential in the lengths of the sequences [13]. Nevertheless, this program can be solved in *polynomial time* by a far-reaching result from linear programming theory known as the equivalence of separation and optimization [14]. This equivalence result is that a linear program can be solved in polynomial time iff the *separation problem* for the linear program can be solved in polynomial time. The separation problem is, given a possibly infeasible vector $\widetilde{x}$ of values for the variables, to report an inequality from the linear program that is violated by $\widetilde{x}$, or to report that $\widetilde{x}$ satisfies the linear program if there is no violated inequality.

We can solve the separation problem in polynomial time for the above linear program, given a concrete parameter choice $\widetilde{x}$, by the following. We consider each example $\mathcal{A}_i$, and find an alignment $\mathcal{B}_i^*$ that maximizes the left-hand side of inequality (11), where for scoring function f we use parameters $\widetilde{x}$. If for this alignment $\mathcal{B}_i^*$ the left-hand side is at most δ_i, then all inequalities of form (11) involving $\mathcal{A}_i$ are satisfied by $\widetilde{x}$; if the left-hand side exceeds δ_i, then this $\mathcal{B}_i^*$ gives the requisite violated inequality. Finding this extreme alignment $\mathcal{B}_i^*$ can be reduced to computing an optimal scoring alignment where scoring function f is modified slightly to take into account the contribution of substitutions that coincide with the columns of $\mathcal{A}_i$ to the recovery error $d(\mathcal{A}_i, \mathcal{B}_i)$. (More details are provided in [38] and [22].) For an instance of inverse alignment with k examples, solving the separation problem for the linear program involves computing at most k optimal scoring alignments, which can be done in polynomial time.

In practice, we solve the linear program using a *cutting plane algorithm* [5]. This approach starts with a small subset $\mathcal{P}$ of all the inequalities $\mathcal{L}$. An optimal solution $\widetilde{x}$ of the linear program restricted to $\mathcal{P}$ is found, and the separation algorithm for the full linear program $\mathcal{L}$ is called on $\widetilde{x}$. If the separation algorithm reports that $\widetilde{x}$ satisfies $\mathcal{L}$, then $\widetilde{x}$ is an optimal solution to $\mathcal{L}$. Otherwise, the violated inequality that is returned is added to $\mathcal{P}$, and the process is repeated. (See [22] for more details.) While such cutting plane algorithms are not guaranteed to terminate in polynomial time, they can be fast in practice. We start with $\mathcal{P}$ containing just the trivial inequalities that specify parameter domain $\mathcal{D}$.

Partial examples As mentioned earlier, the examples that are currently available are partial alignments. Given such a partial example $\mathcal{A}$ for sequences $\mathcal{S}$, a *completion* $\overline{\mathcal{A}}$ of $\mathcal{A}$ is a complete alignment of $\mathcal{S}$ that agrees with the reliable columns of $\mathcal{A}$. In other words, a completion $\overline{\mathcal{A}}$ can change $\mathcal{A}$ on the substrings that are in unreliable columns, but must not alter $\mathcal{A}$ in reliable columns.

When learning parameters by inverse alignment from partial examples, we treat the unreliable columns as missing information.

Definition 2 (Inverse alignment with partial examples) *Inverse Alignment* from partial examples is the following problem. The input is a collection of partial alignments $\mathcal{A}_i$ for $1 \le i \le k$. The output is parameter vector

$$x^* \;:=\; \operatorname*{argmin}_{x \in \mathcal{D}} \; \min_{\overline{\mathcal{A}_1}, \ldots, \overline{\mathcal{A}_k}} \; D(x),$$

where the min is over all possible completions $\overline{\mathcal{A}_i}$ of the partial examples. $\square$

Kim and Kececioglu [21] present the following iterative approach to partial examples. Start with an initial completion $(\overline{\mathcal{A}_i})_0$ for each partial example $\mathcal{A}_i$, which may be formed by computing alignments of the unreliable regions that are optimal with respect to a default parameter choice $(x)_0$. Then iterate the following for $j = 0, 1, 2, \ldots$. Compute an optimal parameter choice $(x)_{j+1}$ by solving inverse alignment on the complete examples $(\overline{\mathcal{A}_i})_j$. Given $(x)_{j+1}$, form a new completion $(\overline{\mathcal{A}_i})_{j+1}$ of each example $\mathcal{A}_i$ by computing an alignment of each unreliable region that is optimal with respect to parameters $(x)_{j+1}$, and concatenating these alignments of the unreliable regions alternating with the alignments given by the reliable regions. Such a completion optimally stitches together the reliable regions, using the current parameter estimate.

The discrepancy for successive parameter estimates forms a monotonically decreasing sequence, and hence converges [21]. In practice, we iterate until the improvement is too small, or a limit on the number of iterations is reached [22].

5 Experimental results

To evaluate these scoring models, we studied how well an optimal alignment under the model recovers a biological benchmark alignment, when using the models for both *pairwise alignment* and *multiple alignment.*

Pairwise alignment For the experiments on pairwise alignment, we collected examples from the following suites of reference alignments: `BAliBASE` [2], `HOMSTRAD` [27], `PALI` [3], and `SABMARK` [35]. From each suite, we selected a subset of 100 pairwise alignments as examples. The corresponding subsets of `BAliBASE` are denoted by B; similarly, H for `HOMSTRAD`, P for `PALI`, and S for `SABMARK`. We also define the union $U = B \cup H \cup P \cup S$, and for a set $X \in \{B, H, P, S\}$ we denote the complement with respect to U by $\overline{X} = U - X$. Gap level $\ell = 8$, window width $w = 7$, and discrepancy weight $\alpha = 0.1$ are used throughout. To predict secondary structure, we used `PSIPRED` [17]; to solve linear programs, we used `GLPK` [25]. Recovery is measured in terms of the *SPS score* [2]: the percentage of residue pairs in the core blocks of all induced pairwise alignments of the reference alignment that are correctly aligned in the computed alignment.

The first set of experiments, shown in Table 1, study the effect of the substitution models alone (without the new gap models). We compared the recovery rates of the lumped and distributed prediction models for secondary structure modifiers; gap costs use the standard affine model of a gap open and extension penalty for internal gaps, and a separate gap open and extension penalty for terminal gaps. The new substitution models are also compared with the standard model that does not use modifiers, called "none." These experiments use hold-out cross validation, where parameters learned on training set $\overline{B}$ are applied to test set B, and so on for each suite of examples. As the table shows, the highest recovery is achieved using the *distributed prediction* model.

Table 2 studies the effect of the gap models alone (without substitution modifiers). The table compares each of the three models of gap context, combined with each of the three measures of the degree of secondary structure. Due to page limits, only results where the test and training sets are both U are shown. As the table shows, the highest recovery is achieved using the *mixed context* combined with the *distributed-continuous degree* measure of secondary structure.

Table 3 shows the improvement in recovery over the standard model (that does not use the new substitution or gap models), upon first adding the best new substitution model (distributed prediction), and then adding the best new gap model (distributed-continuous degree with mixed context). As can be seen, adding the new substitution model gives a large improvement in recovery; adding the new gap model gives a further, but smaller, improvement.

Figure 1 plots the improvement in recovery for pairwise alignment for examples that are ranked according to the dissimilarity of their sequences. We sorted the examples from set U in order of decreasing normalized alignment cost [36], where their cost is scored according to the standard alignment model using the default parameter values of the multiple alignment tool `Opal` [37]. Each value

Table 1 Recovery rates comparing only the new substitution models

		Modifier model		
Training set	Test set	none	lumped	distributed
$\overline{B}$	B	73.52	80.99	81.57
$\overline{H}$	H	79.92	82.44	83.58
$\overline{P}$	P	67.87	74.67	76.60
$\overline{S}$	S	68.76	72.58	72.55

Table 2 Recovery rates comparing only the new gap models

Secondary structure	Gap context model		
degree model	deletion	insertion	mixed
lumped–binary	75.63	75.59	76.51
distributed–binary	75.86	75.90	76.43
distributed–continuous	75.96	76.22	76.63

Table 3 Recovery rates comparing both the substitution and gap models

		Substitution and gap model		
Training set	Test set	std sub std gap	*new* sub std gap	*new* sub *new* gap
$\overline{B}$	B	73.51	81.57	82.88
$\overline{H}$	H	79.92	83.58	84.62
$\overline{P}$	P	67.87	76.60	77.32
$\overline{S}$	S	68.76	72.55	75.52

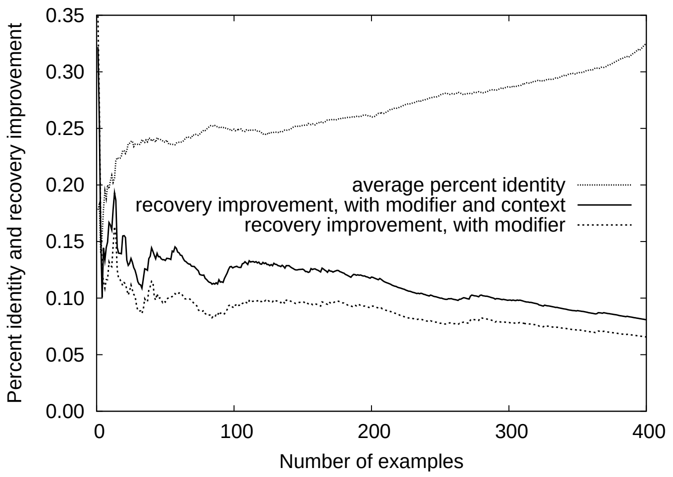

Figure 1 Improvement in recovery rate in relation to percent identity.

of n along the horizontal axis corresponds to using as a test set the first n examples in this ranking. The uppermost curve plots the average percent identity for these corresponding test sets, which generally increases as the average normalized alignment cost decreases along the horizontal axis. The lower two curves plot the improvement in recovery as measured on these test sets for parameter values learned on training set U. Improvement is measured with respect to the standard model with no substitution modifiers and no gap context. The lowest curve adds only substitution modifiers to the standard model, and the next higher curve adds both substitution modifers and gap context. Note that the greatest improvement is generally achieved for examples with the lowest percent identity. For examples with less than 25% identity, the combined improvement in recovery using both substitution modifiers and gap context is as much as 15%.

To give an idea of the running time, solving an inverse alignment instance with 300 examples took at most 55 minutes on a Pentium IV running at 3 GHz with 1 Gb of RAM, and involved around 18,600 cutting planes.

Multiple alignment To evaluate these models when used for multiple sequence alignment, we incorporated them into the alignment tool `Opal` [37]. `Opal` is unique in that it constructs multiple sequence alignments using an algorithm for optimally aligning two multiple alignments under the sum-of-pairs scoring function [20, 36]. Since its subalignments have optimal score in this sense, it is a good testbed for comparing scoring models, as effects due to suboptimality with respect to the scoring function are reduced. The baseline version of `Opal` uses the `BLOSUM62` substitution matrix [16] with affine gap penalties (specifically, a gap open and extension penalty for internal gaps, and another open and extension penalty for terminal gaps). We modified `Opal` to incorporate predicted secondary structure, still using the `BLOSUM62` substitution matrix, but now combined with parameters learned using our best scoring model (namely, the distributed prediction model for substitution modifiers, and the distributed-continuous measure with mixed context for gap costs). In these experiments, we rank benchmarks according their *hardness*: their average recovery rate under `MAFFT`, `ProbCons`, and baseline `Opal` (which all have comparable recovery rates [36]).

Figure 2 shows a scatter plot of the improvement in recovery of `Opal` when using secondary structure, over all benchmarks in the `BAliBASE` [2], `PALI` [3], and `HOMSTRAD` [27] suites. Note that for the more difficult benchmarks (with lower baseline recovery), the boost in recovery using secondary structure tends to be greater. Notice also that using the secondary structure scoring model occasionally worsens the recovery. (This is to be expected, as structure prediction is imperfect, and no universal parameter choice will improve all benchmarks).

Figure 3 shows the recovery rates on these same benchmarks for three variants of `Opal`: (1) the baseline version with no secondary structure model, (2) using the new substitution model with secondary structure but the baseline gap model, and (3) using the new substitution model and the new gap model with secondary structure. Benchmarks are binned according to their baseline recovery in `Opal`; at the top of each bar in parentheses is the number of benchmarks in its bin.

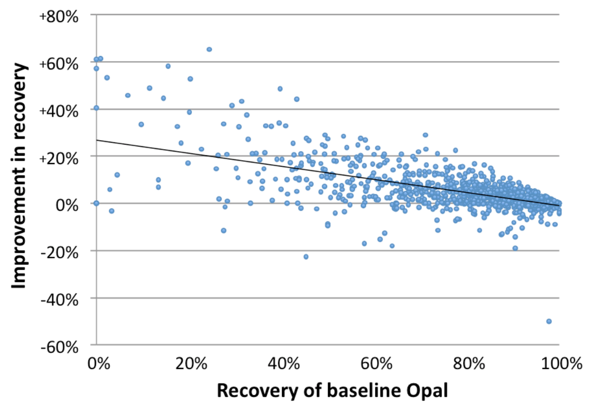

Figure 2 Improvement in recovery using secondary structure within `Opal`.

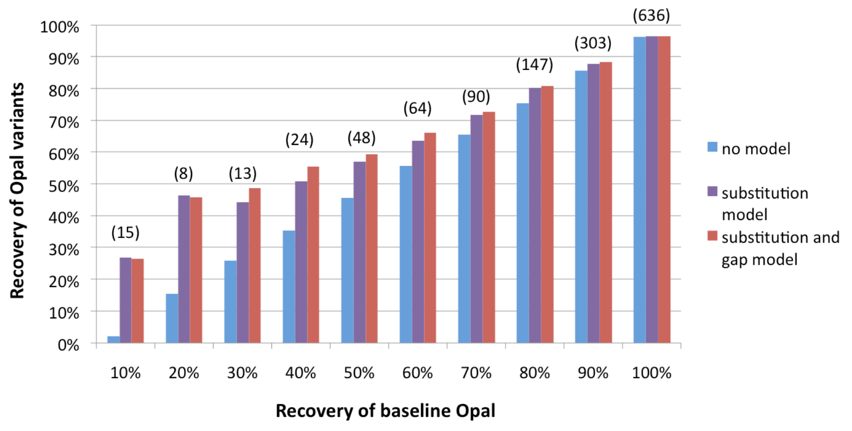

Figure 3 Average recovery of model variants in `Opal` versus binned baseline recovery.

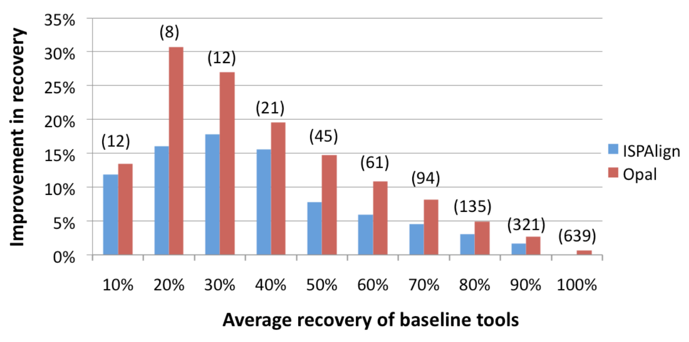

Figure 4 Improvement in recovery using secondary structure within alignment tools.

Note that the substitution model provides the largest boost in recovery, which is generally greatest for the hardest examples.

Finally, Figure 4 compares the improvement in recovery achieved using secondary structure in two multiple alignment tools: `ISPAlign` [23] and `Opal`. (`ISPAlign` has the best recovery among the competing tools `PRALINE`, `SPEM`, and `PROMALS` that use secondary structure.) `ISPAlign` was modified to not add sequence profiles or intermediate sequences (which are not used by `Opal`), so the effect of its secondary structure scoring model could be isolated. The recovery boost in `ISPAlign` is measured with respect to `ProbCons`, since without secondary structure `ISPAlign` uses `ProbCons` to compute alignments. (`ProbCons` and baseline `Opal` have comparable accuracy [36].) The same benchmarks are used, binned by hardness. The boost in recovery for `Opal` is much greater than for `ISPAlign`, suggesting that the more involved secondary structure scoring model used in `Opal` may be more effective than the simpler model used in `ISPAlign`.

6 Conclusion

We have presented new models for protein sequence alignment that incorporate predicted secondary structure, and have shown through experimental results on reference alignments that when model parameters are learned using inverse alignment, the models significantly boost the accuracy of both pairwise and multiple alignment of distant protein sequences. Incorporating secondary structure into the substitution scoring function provides the largest benefit, with distributed prediction giving the most accurate substitution scores, while the new gap penalty functions provide a lesser yet still substantial benefit. Comparing with other multiple alignment tools that incorporate secondary structure shows our models provide a larger increase in accuracy compared to not using secondary structure, which suggests that the additional complexity of our models is offset by their correspondingly greater increase in accuracy.

There remain many avenues for further research. In particular, given that improved substitution scores provided the largest boost in accuracy, it would be interesting to learn a model with a substitution scoring function $\sigma(a, b, c, d)$ that directly scores each pairing $\big\{(a, c),\, (b, d)\big\}$ of amino acids a, b with secondary structure types c, d respectively. Such a model has 1,830 substitution parameters σ_{abcd} alone, and will require a very large training set, combined with a careful procedure for fitting default values for unobserved parameters.

Acknowledgements We thank Matt Cordes and Chuong Do for helpful discussions, and the reviewers for their suggestions. This research was supported by the US National Science Foundation through Grant DBI-0317498.

References

1. Aydin, Z., Y. Altunbasak and M. Borodovsky. “Protein secondary structure prediction for a single-sequence using hidden semi-Markov models.” *BMC Bioinformatics* 7:178, 1–15, 2006.

2. Bahr, A., J.D. Thompson, J.C. Thierry and O. Poch. "BAliBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations." *Nucleic Acids Research* 29:1, 323–326, 2001.
3. Balaji, S., S. Sujatha, S.S.C. Kumar, and N. Srinivasan. "PALI: a database of alignments and phylogeny of homologous protein structures." *Nucleic Acids Research* 29:1, 61–65, 2001.
4. de Berg, M., M. van Kreveld, M. Overmars and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*, 2nd edition. Springer-Verlag, Berlin, 2000.
5. Cook, W., W. Cunningham, W. Pulleyblank, and A. Schrijver. *Combinatorial Optimization.* John Wiley and Sons, New York, 1998.
6. Dayhoff, M.O., R.M. Schwartz and B.C. Orcutt. "A model of evolutionary change in proteins." In M.O. Dayhoff, editor, *Atlas of Protein Sequence and Structure* 5:3, National Biomedical Research Foundation, Washington DC, 345–352, 1978.
7. Do, C.B., M.S. Mahabhashyam, M. Brudno and S. Batzoglou. "ProbCons: probabilistic consistency based multiple sequence alignment." *Genome Research* 15, 330–340, 2005.
8. Do, C.B., S. Gross and S. Batzoglou. "CONTRAlign: discriminative training for protein sequence alignment." Proceedings of the 10th *Conference on Research in Computational Molecular Biology* (RECOMB), Springer LNBI 3909, 160–174, 2006.
9. Durbin, R., S. Eddy, A. Krogh and G. Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, Cambridge, 1998.
10. Edgar, R.C. "MUSCLE: multiple sequence alignment with high accuracy and high throughput." *Nucleic Acids Research* 32, 1792–1797, 2004.
11. Galil, Z. and R. Giancarlo. "Speeding up dynamic programming with applications to molecular biology." *Theoretical Computer Science* 64, 107–118, 1989.
12. Gotoh, O. "An improved algorithm for matching biological sequences." *Journal of Molecular Biology* 162, 705–708, 1982.
13. Griggs, J.R., P. Hanlon, A.M. Odlyzko and M.S. Waterman. "On the number of alignments of k sequences." *Graphs and Combinatorics* 6, 133–146, 1990.
14. Grötschel, M., L. Lovász and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin, 1988.
15. Gusfield, D. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, 1997.
16. Henikoff, S. and J.G. Henikoff. "Amino acid substitution matrices from protein blocks." *Proceedings of the National Academy of Sciences USA* 89, 10915–10919, 1992.
17. Jones, D.T. "Protein secondary structure prediction based on position-specific scoring matrices." *Journal of Molecular Biology* 292, 195–202, 1999.
18. Katoh, K., K.I. Kuma, H. Toh and T. Miyata. "MAFFT version 5: improvement in accuracy of multiple sequence alignment." *Nucleic Acids Research* 33, 511–518, 2005.
19. Kececioglu, J. and E. Kim. "Simple and fast inverse alignment." Proceedings of the 10th *Conference on Research in Computational Molecular Biology* (RECOMB), Springer LNBI 3909, 441–455, 2006.
20. Kececioglu, J. and D. Starrett. "Aligning alignments exactly." Proceedings of the 8th ACM *Conference on Research in Computational Molecular Biology*, 85–96, 2004.

21. Kim, E. and J. Kececioglu. "Inverse sequence alignment from partial examples." Proceedings of the 7th ISCB/EATCS *Workshop on Algorithms in Bioinformatics*, Springer LNBI 4645, 359–370, 2007.
22. Kim, E. and J. Kececioglu. "Learning scoring schemes for sequence alignment from partial examples." IEEE/ACM *Transactions on Computational Biology and Bioinformatics* 5:4, 546–556, 2008.
23. Lu, Y. and S.-H. Sze. "Multiple sequence alignment based on profile alignment of intermediate sequences." *Journal of Computational Biology* 15:7, 676–777, 2008.
24. Lüthy, R., A.D. McLachlan and D. Eisenberg. "Secondary structure-based profies: use of structure-conserving scoring tables in searching protein sequence databases for structural similarities." *Proteins: Structure, Function, and Genetics* 10, 229–239, 1991.
25. Makhorin, A. `GNU` Linear Programming Kit, release 4.8, 2005. `http://www.gnu.org/software/glpk`
26. Miller, W. and E.W. Myers. "Sequence comparison with concave weighting functions." *Bulletin of Mathematical Biology* 50, 97–120, 1988.
27. Mizuguchi, K., C.M. Deane, T.L. Blundell and J.P. Overington. "`HOMSTRAD`: a database of protein structure alignments for homologous families." *Protein Science* 7, 2469–2471, 1998.
28. Notredame, C., D.G. Higgins and J. Heringa. "`T-Coffee`: a novel method for fast and accurate multiple sequence alignment." *Journal of Molecular Biology* 302, 205–217, 2000.
29. Pei, J. and N.V. Grishin. "`MUMMALS`: multiple sequence alignment improved by using hidden Markov models with local structural information." *Nucleic Acids Research* 34, 4364–4374, 2006.
30. Pei, J. and N.V. Grishin. "`PROMALS`: towards accurate multiple sequence alignments of distantly related proteins." *Bioinformatics* 23:7, 802–808, 2007.
31. Sander, C. and R. Schneider. "Database of homology-derived protein structures and the structural meaning of sequence alignment." *Proteins: Structure, Function, and Genetics* 9, 56–68, 1991.
32. Simossis, V.A. and J. Heringa. "`PRALINE`: a multiple sequence alignment toolbox that integrates homology-extended and secondary structure information." *Nucleic Acids Research* 33, W289–294, 2005.
33. Söding, J. "Protein homology detection by HMM-HMM comparison." *Bioinformatics* 21:7, 951–960, 2005.
34. Thompson, J.D., D.G. Higgins and T.J. Gibson. "`CLUSTAL W`: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice." *Nucleic Acids Research* 22, 4673–4680, 1994.
35. Van Walle, I., I. Lasters and L. Wyns. "`Align-m`: A new algorithm for multiple alignment of highly divergent sequences." *Bioinformatics* 20, 1428–1435, 2004.
36. Wheeler, T.J. and J.D. Kececioglu. "Multiple alignment by aligning alignments." Proceedings of the 15th ISCB *Conference on Intelligent Systems for Molecular Biology* (ISMB), *Bioinformatics* 23, i559–i568, 2007.
37. Wheeler, T.J. and J.D. Kececioglu. `Opal`: software for aligning multiple biological sequences. Version 0.3.7, 2007. `http://opal.cs.arizona.edu`
38. Yu, C.-N., T. Joachims, R. Elber and J. Pillardy. "Support vector training of protein alignment models." *Journal of Computational Biology* 15:7, 867–880, 2008.
39. Zhou, H. and Y. Zhou. "`SPEM`: improving multiple sequence alignment with sequence profiles and predicted secondary structures." *Bioinformatics* 21, 3615–3621, 2005.