Aligning protein sequences with predicted secondary structure

John Kececioglu[*] Eagu Kim[†] Travis Wheeler[‡]

11 September 2009

Abstract Accurately aligning distant protein sequences is notoriously difficult. Since the amino acid sequence alone often does not provide enough information to obtain accurate alignments under the standard alignment scoring functions, a recent approach to improving alignment accuracy is to use additional information such as *secondary structure*. We make several advances in alignment of protein sequences annotated with predicted secondary structure: (1) more accurate *models* for scoring alignments, (2) efficient *algorithms* for optimal alignment under these models, and (3) improved *learning* criteria for setting model parameters through inverse alignment, as well as (4) in-depth *experiments* evaluating model variants on benchmark alignments. More specifically, the new models use secondary structure predictions and their confidences to modify the scoring of both substitutions and gaps. All models have efficient algorithms for optimal pairwise alignment that run in near-quadratic time. These models have many parameters, which are rigorously learned using *inverse alignment* under a new criterion that carefully balances score error and recovery error.

We then evaluate these models by studying how accurately an optimal alignment under the model recovers benchmark reference alignments that are based on the known three-dimensional structures of the proteins. The experiments show that these new models provide a significant boost in accuracy over the standard model for distant sequences. The improvement for pairwise alignment is as much as 15% for sequences with less than 25% identity, while for multiple alignment the improvement is more than 20% for difficult benchmarks whose accuracy under standard tools is at most 40%.

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

[*] Corresponding author. Department of Computer Science, University of Arizona, Tucson AZ 85721, USA. Phone: (520)621-4526. Fax: (520)621-4246. Email: `kece@cs.arizona.edu`

[†] Work done while at the Department of Computer Science of the University of Arizona. Current address: Department of Biostatistics and Medical Informatics, University of Wisconsin, Madison WI 53706, USA. Phone: (608)265-2388. Fax: (608)265-7916. Email: `egkim@biostat.wisc.edu`

[‡] Work done while at the Department of Computer Science of the University of Arizona. Current address: Janelia Farm Research Campus, Ashburn VA 20147, USA. Phone: (571)209-4000. Fax: (571)209-4051. Email: `travis@traviswheeler.com`

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` (Thompson et al. 1994), `T-Coffee` (Notredame et al. 2000), and `MUSCLE` (Edgar 2004);

(b) *polishing*, which iteratively refines the alignment by realigning subgroups, and is used by `PRRN` (Gotoh 1996), `MAFFT` (Katoh et al. 2005), `MUSCLE`, `ProbCons` (Do et al. 2005), and `Opal` (Wheeler and Kececioglu 2007a); and

(c) *consistency*, both in its combinatorial (Notredame et al. 2007) and probabilistic (Durbin et al. 1998, Do et al. 2005) settings, which favors matches in the alignment that have high support, and is employed by `T-Coffee`, `MAFFT`, `ProbCons`, `SPEM` (Zhou and Zhou 2005), `PROMALS` (Pei and Grishin 2007), and `ISPAlign` (Lu and Sze 2008).

These techniques all operate on the input sequences alone, and do not require additional sources of information outside the input data.

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` (Simossis and Heringa 2005), `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, as 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 (`SPEM`, `PRALINE`, `PROMALS`, and `ISPAlign`) 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 (Aydin et al. 2006) 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 (Zhou and Zhou 2005, Simossis and Heringa 2005, Lu and Sze 2008), or have a large number of parameters that are estimated by counting frequencies of observed events in comparatively small sets of reference alignments (Lüthy et al. 1991, Pei and Grishin 2007, Söding 2005). Our new models have multiple parameters whose values must be set, and we show that recently developed techniques for parameter learning by inverse alignment (Kececioglu and Kim 2006, Kim and Kececioglu 2008) can be used to find values for these parameters that are optimal in a well-defined sense. Finally, experimental results 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

Since structure is often conserved among related proteins even when sequence similarity is lost (Sander and Schneider 1991), 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 (Jones 1991). Predicted secondary structure can be incorporated into the alignment model quite naturally 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 et al. (1991), who applied the log-odds scoring methodology of Dayhoff et al. (1978) 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 different log-odds substitution score matrices, where a given matrix applies when both of the aligned residues have the *same* structural type. Due to the nature of the reference alignments used to derive these matrices, their work does not provide matrices that apply to substitutions between residues with differing structural types, like α versus β, or α versus other. 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, SPEM, PROMALS, and ISPAlign. We briefly summarize their approaches.

PRALINE (Simossis and Heringa 2005) uses the three substitution matrices of Lüthy et al. (1991) when the aligned residues have the same predicted structure type, plus the BLOSUM62 matrix (Henikoff and Henikoff 1991) when they have differing types. PRALINE also employs four pairs of hand-chosen affine gap penalties (a gap open penalty paired with a gap extension penalty), with one pair per matrix.

SPEM (Zhou and Zhou 2005) modifies a standard substitution matrix by adding a bonus of x to the substitution score when the aligned residues have the same predicted structural 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 (Pei and Grishin 2007) employs a hidden Markov model approach where match, insert, and delete states also emit secondary structure types (in addition to emitting amino acids). The emission and transition probabilities for each of these states are set by counting observed 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 single hand-chosen weighting constant.

ISPAlign (Lu and Sze 2008) uses the hidden Markov model of ProbCons (Do et al. 2005), and modifies its match states to emit secondary structure types as well. 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 order of increasing accuracy, these tools are ranked: PRALINE, SPEM, PROMALS, and ISPAlign. It should be pointed out that these tools also incorporate other approaches besides predicted secondary structure, such as sequence profiles and intermediate sequences, and that they differ in their underlying methods for constructing an alignment.

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 an ensemble 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 (Jones 1999).) 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 parametric alignment (Gusfield and Stelling 1996, Sun et al. 2004, Kececioglu and Kim 2006) applied to training sets of benchmark reference alignments.

Overview

In the next section, we present several new models for scoring alignments of protein sequences based on their predicted secondary structure. Section 3 then 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, and also introduces a new learning criterion that we call average discrepancy. Section 5 then presents experimental results that compare the accuracy of these models when used for both pairwise and multiple alignment. Finally Section 6 concludes, and outlines an avenue for further research.

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 (see Gusfield 1997, p. 240). The general form of f is then

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

where $\widetilde{\gamma}, \widetilde{\lambda}$ are respectively the 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 (Jones 1999), [18], 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, 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

$$\begin{aligned} 0 \;\le\; \gamma_1 \;\le\; \ldots \;\le\; \gamma_\ell, \\ 0 \;\le\; \lambda_1 \;\le\; \ldots \;\le\; \lambda_\ell. \end{aligned}$$

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 the residue at a given position in a gap is involved in secondary structure, and

(ii) which gap positions are considered when determining the level of disruption.

We call the first aspect the *degree* of secondary structure, and the second aspect the *context* of the gap.

2.2.1 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. For both kinds of predictions, we want to measure the degree to which residue position i in sequence A is involved in secondary structure. This *degree* $\Psi_A(i)$

is a value in the range $[0, 1]$, where 0 corresponds to no involvement in secondary structure, and 1 corresponds to full involvement.

We consider three approaches for measuring the degree $\Psi_A(i)$, which depend on whether the prediction is lumped or distributed, and whether the value is binary or continuous.

Lumped-binary approach This approach assumes a lumped prediction, and produces a binary value for the degree:

$$\Psi_A(i) \;\; := \;\; \begin{cases} 1, & S_A[i] \;\in\; \{\alpha, \beta\}; \\ 0, & \text{otherwise.} \end{cases}$$

Distributed-binary approach This approach assumes a distributed prediction, and produces a binary degree:

$$\Psi_A(i) \;\; := \;\; \begin{cases} 1, & P_A(i, \alpha) \;+\; P_A(i, \beta) \;>\; P_A(i, \phi); \\ 0, & \text{otherwise.} \end{cases}$$

Distributed-continuous approach This approach assumes a distributed prediction, and produces the real value,

$$\Psi_A(i) \;\; := \;\; P_A(i, \alpha) \;+\; P_A(i, \beta).$$

For each of these approaches, the degree measure is then aggregated over the positions in the gap context, as discussed next.

2.2.2 Specifying the gap context

The gap context specifies the positions that functions H and h consider when respectively measuring the *global* and *local* secondary structure level. Both H and h make use of a degree function $\Psi(i)$, as defined above. In the definitions of H and h below, the notation $\langle x \rangle$ maps a real value $x \in [0, 1]$ to the discrete levels $1, 2, \ldots, \ell$ by

$$\langle x \rangle \;\; := \;\; \begin{cases} \lfloor x\ell \rfloor + 1, & x \in [0, 1); \\ 1, & x = 1. \end{cases}$$

To measure the *local* level h at position i in sequence S, we consider a small window $W(i)$ of consecutive positions centered around i. We first average the secondary structure degree over the positions in the window,

$$d(S, i) \;\; := \;\; \frac{1}{\bigl|W(i)\bigr|} \sum_{p \,\in\, W(i)} \Psi_S(p),$$

and then map this average degree to a discrete level:

$$h(S,i) \;\; := \;\; \Big\langle d(S,i) \Big\rangle .$$

Informally, the local level h at position i reflects the average secondary structure degree Ψ for the residues in a window around i. Generally all windows have the same width $\big|W(i)\big| = w$, except when position i is too close to the left or right end of S to be centered in a window of width w, in which case $W(i)$ naturally shrinks on that 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. This same context will apply to all the gaps in an alignment.

Deletion context This context views the disruption caused by a gap (S,i,j,T,k) in terms of the secondary structure lost by the *deletion* of 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,

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

An important property of this model is that for substrings $S[i : j]$ and $S[i' : j']$ where $[i,j] \subseteq [i',j']$, their gap costs satisfy $g(S,i,j,T,k) \le g(S,i',j',T,k')$, independent of k and k'. In other words, the deletion context gap cost is monotonically increasing as the substring for deletion is extended.

Insertion context This 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

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

which gives the gap cost function,

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

Mixed context This context combines the above global measure H of the insertion context with the local measure h of the deletion context, which gives

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

This completes the description of the scoring models that we consider for alignment of protein sequences with predicted secondary structure.

To summarize, the *parameters* of the scoring models are:

- the 210 substitution costs σ_{ab},
- the 6 substitution modifiers μ_{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 models depend 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.

We next develop efficient algorithms for computing optimal alignments under these models in near-quadratic time. Section 4 then explains how to learn parameter values for these models through inverse parametric alignment, which makes use of these efficient algorithms for optimal alignment.

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* involving residues $A[i]$ and $B[j]$, or
- a *gap* involving either 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,

$$C(i,j) \;=\; \min \begin{cases} C(i-1,j-1) \;+\; s(A,i,B,j), \\[2ex] \displaystyle\min_{1\le k\le i} \Big\{ C(k-1,j) \;+\; g(A,k,i,B,j) \Big\}, \\[2ex] \displaystyle\min_{1\le k\le j} \Big\{ C(i,k-1) \;+\; g(B,k,j,A,i) \Big\}. \end{cases} \tag{7}$$

(To simplify the presentation, this ignores boundary conditions and the special case of external gaps, which are straightforward.) For two sequences of length n, the standard 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.

3.1 Insertion and mixed contexts

For the *insertion context*, function g given by Equation (5) is very close to a standard affine gap cost function, with a per-gap open cost and a per-residue extension cost. The same technique developed by Gotoh (1982) for alignment with standard affine gap costs can be used for the insertion context to reduce the total time to compute an optimal alignment to $O(n^2)$. The basic idea is to:

(1) keep 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) consider the last two columns of an alignment to decide whether or not the gap open cost should be charged for an insertion or deletion involving a residue.

For the *mixed context*, given by Equation (6), this approach also leads to an $O(n^2)$ time algorithm.

More precisely, let

- $S(i, j)$ be the cost of an optimal alignment of $A[1\!:\!i]$ and $B[1\!:\!j]$ that ends with a *substitution* of $A[i]$ by $B[j]$,
- $I(i, j)$ be the cost of an optimal alignment of the same prefixes that ends with an *insertion* involving $B[j]$, and
- $D(i, j)$ be the cost of an optimal alignment of these prefixes that ends with a *deletion* involving $A[i]$.

Then $C(i, j)$ is $\min\Bigl\{S(i, j),\ I(i, j),\ D(i, j)\Bigr\}$.

For the *mixed context*, these quantities satisfy the recurrences,

$$S(i,j) \;=\; \min\left\{\begin{array}{l} S(i{-}1, j{-}1), \\ I(i{-}1, j{-}1), \\ D(i{-}1, j{-}1) \end{array}\right\} \;+\; s(A, i, B, j),$$

$$I(i,j) \;=\; \min\left\{\begin{array}{l} S(i, j{-}1) \;+\; \gamma\Bigl(H(A, i)\Bigr), \\ I(i, j{-}1), \\ D(i, j{-}1) \;+\; \gamma\Bigl(H(A, i)\Bigr) \end{array}\right\} \;+\; \lambda\Bigl(h(B, j)\Bigr), \tag{8}$$

$$D(i,j) \;=\; \min\left\{\begin{array}{l} S(i{-}1, j) \;+\; \gamma\Bigl(H(B, j)\Bigr), \\ I(i{-}1, j) \;+\; \gamma\Bigl(H(B, j)\Bigr), \\ D(i{-}1, j) \end{array}\right\} \;+\; \lambda\Bigl(h(A, i)\Bigr). \tag{9}$$

(To simplify the presentation, this again ignores boundary conditions and the special case of external gaps.)

For the *insertion context*, these quantities satisfy identical recurrences, except that in Equation (8) the final additive term becomes $\lambda(H(A, i))$, while in Equation (9) this term becomes $\lambda(H(B, j))$.

Aligning two sequences of lengths m and n by evaluating these recurrences in constant time at each entry (i, j) of a table with $O(mn)$ entries, and then recovering an optimal alignment from the values in the table, gives the following result.[1]

[1] At an entry (i, j), functions H and h can be directly evaluated in constant time when windows have constant width w. For non-constant w, we can precompute $d(S, i)$ for both input strings S and

Theorem 1 (Insertion and mixed context running time) *For both the insertion and mixed contexts, optimally aligning two sequences of lengths m and n takes $O(mn)$ time.*

3.2 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 (1988) and Galil and Giancarlo (1989). 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 inner 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 $\widetilde{F}_j(i)$ and hence $C(i,j)$ 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 (1997, 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 the 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 $F(j)$. A given 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* (de Berg et al. 2000).

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 $k \le j$. The lower envelope for $j' \ge j$ is represented as a partition of the interval $[j,n]$ into maximal subintervals such that across each subinterval the lower envelope is given by exactly one candidate's curve.

all positions i using $O(m+n)$ time preprocessing, and then evaluate H and h in constant time by table lookup.

The partition can be described by a list of the right endpoints of the subintervals, say $p_1 < \ldots < p_t = n$ for t subintervals, together with a list of the corresponding candidates $k_1, \ldots, k_t$ that specify the lower envelope at these subintervals. Note that once the partition is known at column j, the value $F(j)$ is given by the curve for the candidate corresponding to the first subinterval $[j, p_1]$ of the partition, namely $G(k_1, j)$. When advancing to column $j{+}1$ in the row, the partition is updated by considering the effect of the curve for candidate $j{+}1$ on the lower envelope.

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 $c \geq b$. Then at all $d \geq c$,*

$$G(a,d) \ < \ G(b,d).$$

Proof The key is to show that the difference

$$\Big(G(b,d) \ - \ G(b,c)\Big) \ - \ \Big(G(a,d) \ - \ G(a,c)\Big) \tag{10}$$

is nonnegative, as adding $G(b,c) - G(a,c)$ implies the lemma. Let function $M(S,x,y)$ be the maximum of $h(S,p)$ over the interval $p \in [x,y]$. For the deletion context, quantity (10) above equals

$$\Big(\gamma\Big(M(S,b,d)\Big) \ - \ \gamma\Big(M(S,b,c)\Big)\Big) \ - \ \Big(\gamma\Big(M(S,a,d)\Big) \ - \ \gamma\Big(M(S,a,c)\Big)\Big). \tag{11}$$

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 (11) above is nonnegative, which proves the lemma. □

A consequence of Lemma 1 is that the curves for any pair of candidates cross *at most once*, which we formalize in the following lemma. (In contrast, for the standard affine gap cost model, candidate curves do not cross, so in that model a single candidate determines the lower envelope.) This property is the key to efficiently updating the lower envelope.

Below, for candidates a, b and interval x, we write $a \preceq_x b$ if $G(a,j) \leq G(b,j)$ for all j in x. We write $a \prec_x b$ if $a \preceq_x b$ and $G(a,j) < G(b,j)$ for some j in x.

Lemma 2 (Crossing once property) *For candidates $a < b$ on interval x, either $a \preceq_x b$, $b \preceq_x a$, or x splits into successive intervals y, z such that $b \prec_y a$ and $a \prec_z b$. Moreover if $G(b,c) \leq G(a,c)$ at the right endpoint c of x, then $b \preceq_x a$.*

Proof Essentially, there cannot be three positions $i < j < k$ in x such that the candidate whose curve is below the other at i, j, k is respectively b, a, b. In such a case, Lemma 1 contradicts $G(b,k) < G(a,k)$.

For the last part of the lemma, suppose $G(b,c) \leq G(a,c)$ at the right endpoint c. If $G(a,j) < G(b,j)$ at some j in x, Lemma 1 gives a contradiction at c. So $b \preceq_x a$. □

Using this crossing once 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. In general 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 by Lemma 2 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, by Lemma 2 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 and space 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* (de Berg et al. 2000) 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 2 (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 these alignment scoring models using the *inverse alignment* technique with a new optimization criterion. (See Kim (2008) for a survey on inverse alignment.) This new criterion extends the approach of Kim and Kececioglu (2007) to incorporate a so-called loss function as introduced by

Yu, Joachims, Elber and Pillardy (2007). Informally, 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 of the alignment scoring model that makes the reference alignments score as close as possible to optimal scoring alignments of their sequences. The approach of Kim and Kececioglu (2007) finds a parameter assignment that minimizes the average absolute error in score across the examples, while Yu et al. (2007) 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 known three-dimensional structures for 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 the unreliable columns in a partial alignment, the alignment is effectively left unspecified. In a *complete alignment*, all columns are reliable. An *example* is said to be *complete* or *partial* depending on whether its reference alignment is complete or partial. We follow the presentation of Kim and Kececioglu (2008), which first develops an algorithm for inverse alignment from complete examples, and then extends it to partial examples.

4.1 Minimizing discrepancy on complete examples

In the following, we define inverse alignment from complete examples under a new optimization criterion that we call *discrepancy*. This discrepancy criterion generalizes the score error approach of Kim and Kececioglu (2007) and the recovery error approach of Yu et al. (2007). In our experiments, it is superior to both.

For 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, where each *example* $\mathcal{A}_i$ aligns a corresponding pair of sequences $\mathcal{S}_i$. We denote the average *length* of the sequences in $\mathcal{S}_i$, say the pair A and B, by $L(\mathcal{S}_i) := \frac{1}{2}|A| + \frac{1}{2}|B|$.

For an alignment $\mathcal{B}_i$ of the sequences $\mathcal{S}_i$ for example $\mathcal{A}_i$, let function $d(\mathcal{A}_i, \mathcal{B}_i)$ be the number of reliable columns of example $\mathcal{A}_i$ that are *not* present in alignment $\mathcal{B}_i$, divided by the total number of reliable columns in the example.[2] In other words, function d measures the *error* in recovering example $\mathcal{A}_i$ by alignment $\mathcal{B}_i$ as a fraction between 0 and 1 (which correspond respectively to no error and total error). Yu et al. (2007) call d a loss function.

[2] Since d will be applied to partial examples in Section 4.2, we emphasize that the counts in its numerator and denominator are with respect to reliable columns only.

Definition 1 (Inverse alignment under discrepancy 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

$$x^* \;:=\; \operatorname*{argmin}_{x \in \mathcal{D}} D(x),$$

where

$$D(x) \;:=\; \frac{1}{k} \sum_{1 \le i \le k} \max_{\mathcal{B}_i \text{ of } \mathcal{S}_\mathrm{i}} \left\{ (1-\alpha)\, \frac{f_x(\mathcal{A}_i) \;-\; f_x(\mathcal{B}_i)}{L(\mathcal{S}_i)} \;+\; \alpha\; d(\mathcal{A}_i, \mathcal{B}_i) \right\}. \tag{12}$$

In the above sum, the maximum is over all alignments $\mathcal{B}_i$ of the example sequences $\mathcal{S}_i$, 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* for the examples under parameters x. □

The discrepancy criterion reduces to the approach of Kim and Kececioglu (2007) when $\alpha = 0$, and to the approach of Yu et al. (2007) when $\alpha = \frac{1}{2}$, upon removing the length normalization of score error by setting $L(\mathcal{S}_i) \equiv 1$. 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. 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 under the discrepancy criterion 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

$$(0, -1, 0, 0) \;\le\; (\sigma_{ab}, \mu_{cd}, \gamma_i, \lambda_i) \;\le\; (1, 1, 1, 1),$$

and the inequalities

$$\begin{aligned} \sigma_{aa} &\le \sigma_{ab}, \\ \mu_{cc} &\le 0, \\ \mu_{cd} &\ge 0, \\ \gamma_i &\le \gamma_{i+1}, \\ \lambda_i &\le \lambda_{i+1}. \end{aligned}$$

The remaining inequalities in the linear program 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

$$(1-\alpha)\,\frac{f_x(\mathcal{A}_i) \;-\; f_x(\mathcal{B}_i)}{L(\mathcal{S}_i)} \;\;+\;\; \alpha\; d(\mathcal{A}_i, \mathcal{B}_i) \;\;\leq\;\; \delta_i. \tag{13}$$

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

Finally, 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 variable δ_i to equal the term with index i in the summation for $D(x)$ in Equation (12). 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 (13), 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 (Griggs et al. 1990). 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 (Grötschel et al. 1988). 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 all its 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 vector $\widetilde{x}$, 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 (13), where for scoring function f we use the parameter values in $\widetilde{x}$. If for this alignment $\mathcal{B}_i^*$ the left-hand side is at most δ_i, then all inequalities of form (13) 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 reliable columns of $\mathcal{A}_i$ to the recovery error $d(\mathcal{A}_i, \mathcal{B}_i)$. (More details are provided in Yu et al. (2007) and Kim and Kececioglu (2008).) 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* (Cook et al. 1998). This approach starts with a small subset $\mathcal{P}$ of all the inequalities in the full linear program $\mathcal{L}$. An optimal solution $\widetilde{x}$ of the linear program restricted to $\mathcal{P}$ is found, and the separation algorithm for the full 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 by the separation algorithm is added to $\mathcal{P}$, and the process is repeated. (See Kim and Kececioglu (2008) 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 subset $\mathcal{P}$ containing just the trivial inequalities that specify parameter domain $\mathcal{D}$.

4.2 Extending to 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, as follows.

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 inner minimum is over all possible completions $\overline{\mathcal{A}_i}$ of the partial examples.
□

Kim and Kececioglu (2007) 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 its unreliable regions that are optimal with respect to a default parameter choice $(x)_0$. (In practice, for $(x)_0$ we use a standard substitution matrix (Henikoff and Henikoff 1992) with appropriate gap penalties.) Then iterate the following process 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 of the partial example, using the current estimate for parameter values.

The analysis of Kim and Kececioglu (2008) also proves that for the new average discrepancy criterion, the discrepancy D of successive parameter estimates $(x)_j$ forms a monotonically decreasing sequence, and hence converges. In practice, we iterate until the improvement in discrepancy becomes too small, or a limit on the number of iterations is reached. As discrepancy improves across the iterations, recovery of the examples generally improves as well (Kim and Kececioglu 2008).

5 Experimental results

To evaluate these new alignment scoring models, we studied how well an optimal alignment under the model recovers a biological benchmark alignment, using parameters learned from reference alignments. The experiments study the models when used for both *pairwise alignment* and *multiple alignment*.

Table 1 Recovery rates comparing the substitution models alone

		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

5.1 Pairwise sequence alignment

For the experiments on pairwise alignment, we collected examples from the following suites of reference alignments: BAliBASE (Bahr et al. 2001), HOMSTRAD (Mizuguchi et al. 1998), PALI (Balaji et al. 2001), and SABMARK (Van Walle et al. 2004). From each suite, we selected a subset of 100 pairwise alignments as examples. We denote these sets of examples by:

- B for the BAliBASE subset,
- H for the HOMSTRAD subset,
- P for the PALI subset,
- S for the SABMARK subset,
- U for the union $B \cup H \cup P \cup S$, and
- $\overline{X}$ for the complement $U - X$ of each set $X \in \{B, H, P, S\}$.

Gap level $\ell = 8$, window width $w = 7$, and discrepancy weight $\alpha = 0.1$ are used throughout. To predict secondary structure we used PSIPRED (Jones 1999), and to solve linear programs we used GLPK (Makhorin 2005). For the pairwise alignment experiments of this section (and the multiple alignment experiments of Section 5.2), recovery is measured by the so-called SPS score (Bahr et al. 2001). For pairwise alignment recovery, this score is the percentage of residue pairs in the core blocks of the reference alignment that are correctly aligned in the computed alignment. (For multiple alignment recovery in Section 5.2, this same score is measured across all induced pairwise alignments of the reference multiple 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 μ_{cd} on substitution costs. (In our tables, recovery rates are in percentages.) 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; these four penalties were learned as well. The new substitution models are also compared with the standard substitution 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 benchmarks. As the table shows, the highest

Table 2 Recovery rates comparing the gap models alone

	Gap context model		
Structure 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

recovery is achieved using the *distributed prediction* model.

Table 2 studies the effect of the gap models alone, without substitution modifiers. (The standard substitution model is used.) The table compares each of the three models of gap context, combined with each of the three measures of the degree of secondary structure. To simplify the presentation for these comparisons between all context and degree models, the test and training sets are both U. (Later Table 3 shows the best of these gap models across all four disjoint test and training sets.) 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 Recovery rates on adding the new substitution and gap models

		Substitution and gap model		
Training set	Test set	none	sub	sub + 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

Table 3 shows the improvement in recovery over the standard model (without 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 with secondary structure modifiers μ_{cd} gives a large improvement in recovery. Adding the new gap model gives a further, but smaller, improvement.

The results in these tables have shown aggregate recovery, which effectively is an average across all the reference alignments in a test set, but do not show how the new substitution and gap models improve the recovery for sequences of varying similarity.

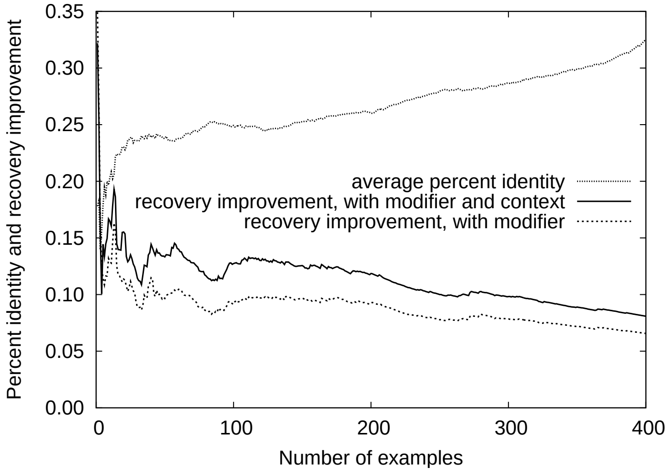

Figure 1 Improvement in recovery rate in relation to percent identity.

Figure 1 plots the improvement in recovery for pairwise alignment on benchmark reference alignments that are ranked according to the similarity of their sequences. To explain the plot, we first sorted the reference alignments in set U in order of decreasing normalized alignment cost (Wheeler and Kececioglu 2007a), where this measure divides the cost of an alignment by the average length of its sequences (which in general is a more robust measure of dissimilarity than percent identity). For this measure, alignment cost was scored under the standard alignment model using the default parameter values of the multiple alignment tool `Opal` (Wheeler and Kececioglu 2007b). Each value of n along the horizontal axis corresponds to using as a test set the first n reference alignments 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 benchmark reference alignments with the lowest percent identity. For benchmarks with less than 25% identity, the combined improvement in recovery using both substitution modifiers and gap context is as much as 15%.

Additional experiments show the effect of the number of examples in the training set on both the recovery rate and the running time for parameter learning. To vary the number of examples in our prior test and training sets, for each set $X \in \{B, H, P, S\}$ and various i, we formed a subset $X_i \subseteq X$ containing exactly i examples. (Notice that $X_{100} = X$, since our original sets X each have 100 examples.) Similarly, with respect

Table 4 Effect of training set size on recovery rate

		Index i		
Training set	Test set	25	50	100
$\overline{B_i}$	B_{100}	79.36	80.67	82.88
$\overline{H_i}$	H_{100}	81.83	83.48	84.62
$\overline{P_i}$	P_{100}	75.26	75.10	77.32
$\overline{S_i}$	S_{100}	72.81	74.65	75.52

Table 5 Effect of training set size on running time and cutting planes

	Index i					
	25		50		100	
Training set	time	planes	time	planes	time	planes
$\overline{B_i}$	39:09	15,448	40:37	14,234	45:03	13,838
$\overline{H_i}$	46:47	17,841	37:29	14,864	49:58	14,789
$\overline{P_i}$	54:55	18,619	52:01	16,008	38:22	14,440
$\overline{S_i}$	54:50	17,831	48:44	15,313	52:53	15,356

to the union $U_i := B_i \cup H_i \cup P_i \cup S_i$, we also formed the complement $\overline{X_i} := U_i - X_i$ for each X.

Table 4 shows how the size of the training set affects the recovery rate. Note that each training set $\overline{X_i}$ has $3i$ examples, so as index i doubles across the table, the size of the training set also doubles. (An advantage of learning parameters by inverse alignment is that the linear programming approach can handle large training sets.) For increasingly larger training sets, the parameters learned are applied to the same disjoint test set $X_{100} = X$. Generally, using more examples yields higher recovery rates.

For these same training sets, Table 5 shows the running time and number of cutting planes in the linear program for parameter learning by inverse alignment. (The number of cutting planes reported is the maximum across the iterations of the approach to partial examples described in Section 4.2, which is a measure of the size of the largest linear program solved.) Running times are in minutes and seconds on a Pentium IV running at 3 GHz with 1 GB of RAM. Note that all experiments completed in less than one hour. In particular, solving the largest instances with 300 examples took at most 55 minutes and used at most around 18,600 cutting planes. (Keeping the linear programs to this size is nontrivial, and requires a careful implementation of the cutting plane algorithm (see Kim 2008).) It is noteworthy that as the number of examples increases, the running time and size of the linear program remains comparatively stable, even though the number of examples is doubling. Interestingly, the number of cutting planes actually *decreases* with more examples, suggesting that a greater variety of examples may enable the algorithm to find more effective cutting planes.

5.2 Multiple sequence alignment

To evaluate these models when used for multiple sequence alignment, we incorporated them into the multiple alignment tool `Opal` (Wheeler and Kececioglu 2007b). `Opal` is unique in that it constructs multiple sequence alignments by exploiting a fast algorithm for optimally aligning two multiple alignments under the sum-of-pairs scoring function with affine gap penalties (Wheeler and Kececioglu 2007a). (While the problem of optimally aligning two alignments is theoretically NP-complete for affine gap penalties (see Starrett 2008), `Opal` builds upon an exact algorithm for aligning alignments that is remarkably fast in practice on biological sequences (Kececioglu and Starrett 2004).) Since its subalignments have optimal score in this sense, `Opal` is a good testbed for comparing the new 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 (Henikoff and Henikoff 1992) with carefully chosen affine gap penalties (specifically, a gap open and extension penalty for internal gaps, and another open and extension penalty for terminal gaps). We modified baseline `Opal` to incorporate predicted secondary structure, still using the `BLOSUM62` substitution matrix, but now combined with parameters learned using our best scoring model: the *distributed prediction* model for substitution modifiers, with the *distributed-continuous degree* measure and *mixed context* for gap costs. (Modifying the fast algorithm for optimally aligning alignments within `Opal` to accommodate this more complex gap model is nontrivial, but outside the scope of this paper (see Wheeler 2009).)

When studying the scoring models applied to multiple alignment, we rank the benchmark reference alignments in our experiments according to what we call their *hardness*: the average of their recovery rates under the baseline tools `MAFFT`, `ProbCons`, and `Opal`, all of which have equivalent recovery rates (Wheeler and Kececioglu 2007a) and are among the most accurate tools available for multiple alignment that do not use secondary structure. This is in contrast to the common practice of ranking benchmarks by their percent identity. As shown in Figure 2, which gives a scatter plot of hardness versus percent identity for all reference alignments in the `BAliBASE`, `HOMSTRAD`, and `PALI` suites, once percent identity falls below 30%, there is little correlation between the percent identity of a benchmark and its difficulty: a benchmark with low percent identity can be easy (100% recovery) or hard (0% recovery). Consequently, in what follows we use recovery rate under a baseline tool as our measure for ranking benchmarks.

Figure 3 gives a scatter plot of the improvement in the recovery rate of `Opal` over its baseline version when using the secondary structure scoring model, for all benchmarks in `BAliBASE`, `HOMSTRAD`, and `PALI`. Note that for the harder benchmarks (with lower baseline recovery), the boost in recovery using secondary structure tends to be greater. Notice also that using secondary structure occasionally worsens the recovery (which should be expected, since structure prediction is imperfect, and no universal parameter choice will improve all benchmarks).

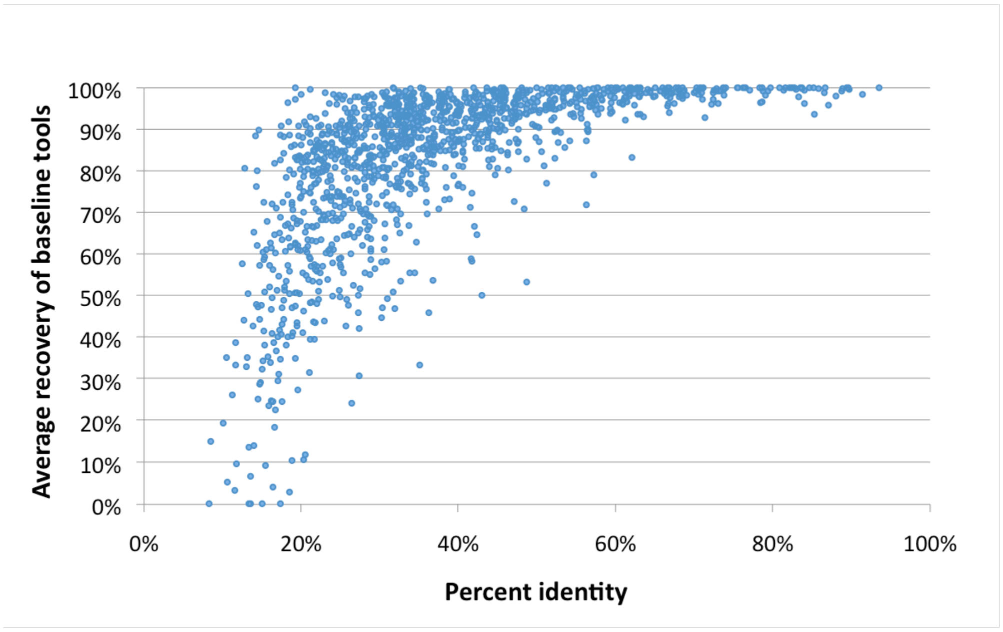

Figure 2 Hardness versus percent identity for benchmark reference alignments.

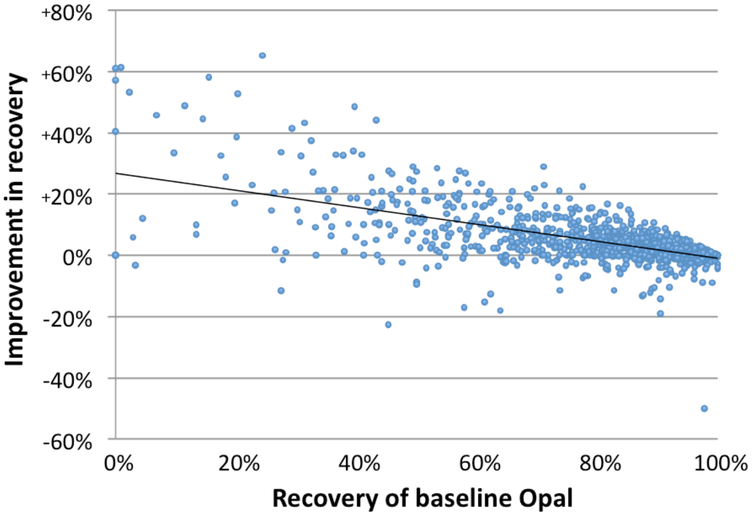

Figure 3 Improvement in recovery rate using secondary structure in `Opal`.

Figure 4 shows the recovery rates on these same benchmarks for three variants of `Opal`: (1) the baseline version with no secondary structure, (2) using the new substitution model but the baseline gap model, and (3) using the new substitution model and the new gap model. Benchmarks are binned according to their recovery again for base-

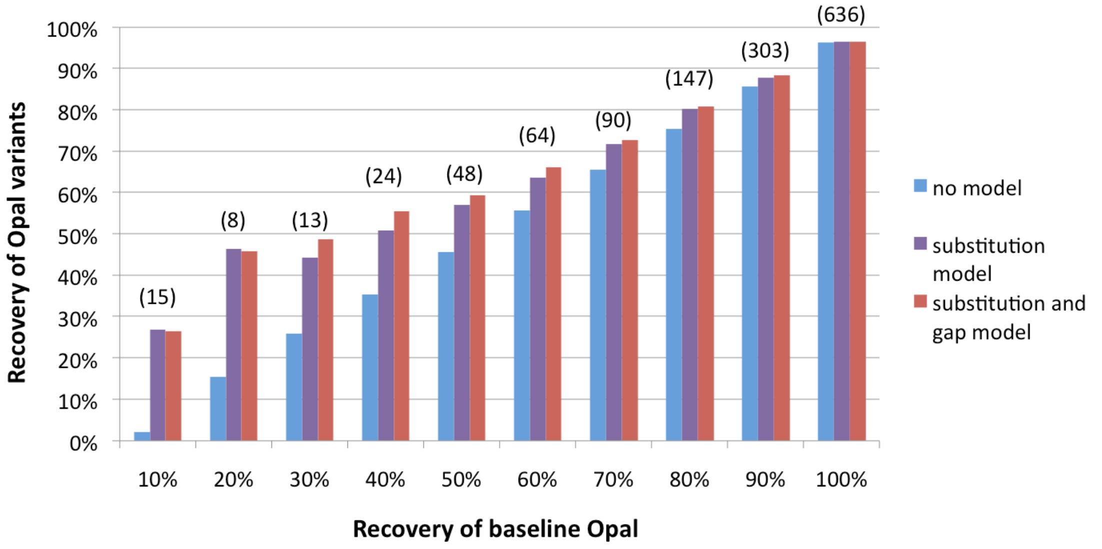

Figure 4 Recovery rates of models in `Opal` versus binned baseline recovery.

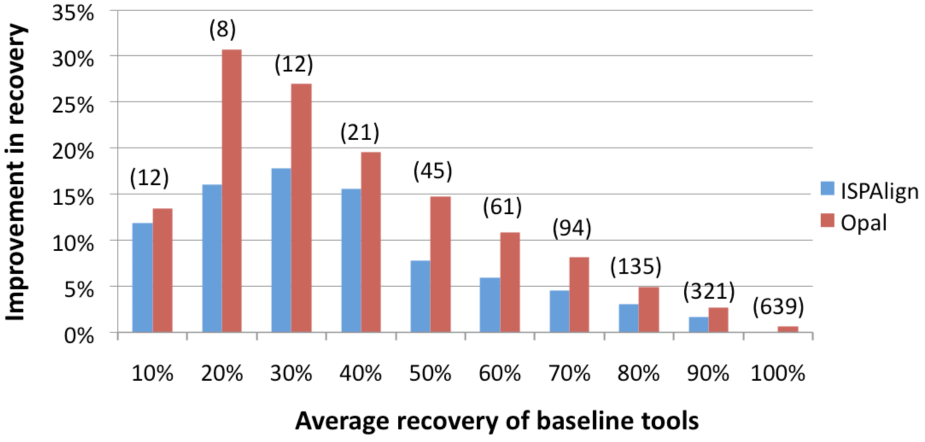

Figure 5 Improvement in recovery rate using secondary structure in alignment tools.

line `Opal`. Bin x contains benchmarks whose baseline recovery is in interval $(x-10, x]$. Bars show the recovery rate for a variant averaged over the benchmarks in its bin. In parentheses are the number of benchmarks in each bin. Note that the substitution model provides the largest boost in recovery, which is generally greatest for the hardest benchmarks.

Finally, Figure 5 compares the improvement in recovery rate achieved by using secondary structure in two multiple alignment tools: `ISPAlign` (Lu and Sze 2008) and `Opal`. (`ISPAlign` has the best recovery among the competing alignment tools `PRALINE`, `SPEM`, and `PROMALS` that use secondary structure. `ISPAlign`, like `Opal` in this comparison, uses `PSIPRED` to predict secondary structure.) To make an equitable comparison of secondary structure models, `ISPAlign` was modified to not annotate the input sequences with profiles of amino acid exchanges, and to not augment the input with intermediate sequences, both of which are found through database searches (and are not used by `Opal`), so the effect of the `ISPAlign` secondary structure scoring model could be isolated. Since without secondary structure `ISPAlign` effectively uses `ProbCons` to compute its alignments, the recovery boost in `ISPAlign` is measured with respect to `ProbCons`. (Recall that `ProbCons` and baseline `Opal` have equivalent accuracy (Wheeler and Kececioglu 2007a).) The same benchmarks from `BAliBASE`, `PALI`, and `HOMSTRAD` are used, binned by hardness. (Note that these standard suites contain comparatively few hard benchmarks, so simply reporting the improvement averaged over all benchmarks will not convey the true strength of the models, as most of these benchmarks already are easy.) For the hardest benchmarks of at most 40% baseline recovery, the improvement in recovery for `Opal` is more than 20%. The boost in recovery for `Opal` is generally much greater than for `ISPAlign`, suggesting that the new secondary structure scoring model may be making more effective use of secondary structure information 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 benchmark 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 that 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.

Further research

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. An earlier conference version of this paper appeared as Kim et al. (2009).

Disclosure statement

No competing financial interests exist.

References

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

[2] Bahr, A., Thompson, J.D., Thierry, J.C., and Poch, O. 2001. `BAliBASE` (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. *Nucleic Acids Research* 29:1, 323–326.

[3] Balaji, S., Sujatha, S., Kumar, S.S.C., and Srinivasan, N. 2001. `PALI`: a database of alignments and phylogeny of homologous protein structures. *Nucleic Acids Research* 29:1, 61–65.

[4] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. 2000. *Computational Geometry: Algorithms and Applications*, 2nd edition. Springer-Verlag, Berlin.

[5] Cook, W., Cunningham, W., Pulleyblank, W., and Schrijver, A. 1998. *Combinatorial Optimization*. John Wiley and Sons, New York.

[6] Dayhoff, M.O., Schwartz, R.M., and Orcutt, B.C. 1978. A model of evolutionary change in proteins, 345–352. In Dayhoff, M.O., editor, *Atlas of Protein Sequence and Structure* 5:3, National Biomedical Research Foundation, Washington DC.

[7] Do, C.B., Mahabhashyam, M.S., Brudno, M., and Batzoglou, S. 2005. `ProbCons`: probabilistic consistency based multiple sequence alignment. *Genome Research* 15, 330–340.

[8] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. 1998. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, Cambridge.

[9] Edgar, R.C. 2004. `MUSCLE`: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research* 32, 1792–1797.

[10] Galil, Z., and Giancarlo, R. 1989. Speeding up dynamic programming with applications to molecular biology. *Theoretical Computer Science* 64, 107–118.

[11] Gotoh, O. 1982. An improved algorithm for matching biological sequences. *Journal of Molecular Biology* 162, 705–708.

[12] Gotoh, O. 1996. Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology* 264, 823–838.

[13] Griggs, J.R., Hanlon, P., Odlyzko, A.M., and Waterman, M.S. 1990. On the number of alignments of k sequences. *Graphs and Combinatorics* 6, 133–146.

[14] Grötschel, M., Lovász, L., and Schrijver, A. 1988. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin.

[15] Gusfield, D. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York.

[16] Gusfield, D., and Stelling, P. 1996. Parametric and inverse-parametric sequence alignment with `XPARAL`. *Methods in Enzymology* 266, 481–494.

[17] Henikoff, S., and Henikoff, J.G. 1992. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences USA* 89, 10915–10919.

[18] Jones, D.T. 1999. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of Molecular Biology* 292, 195–202.

[19] Katoh, K., Kuma, K.I., Toh, H., and Miyata, T. 2005. `MAFFT` version 5: improvement in accuracy of multiple sequence alignment. *Nucleic Acids Research* 33, 511–518.

[20] Kececioglu, J., and Kim, E. 2006. Simple and fast inverse alignment. Proceedings of the 10th *Conference on Research in Computational Molecular Biology* (RECOMB), Springer LNBI 3909, 441–455.

[21] Kececioglu, J., and Starrett, D. 2004. Aligning alignments exactly. Proceedings of the 8th ACM *Conference on Research in Computational Molecular Biology*, 85–96.

[22] Kim, E. 2007. *Inverse Parametric Alignment for Accurate Biological Sequence Comparison.* PhD dissertation, Department of Computer Science, The University of Arizona.

[23] Kim, E., and Kececioglu, J. 2007. Inverse sequence alignment from partial examples. Proceedings of the 7th ISCB/EATCS *Workshop on Algorithms in Bioinformatics*, Springer LNBI 4645, 359–370.

[24] Kim, E., and Kececioglu, J. 2008. Learning scoring schemes for sequence alignment from partial examples. IEEE/ACM *Transactions on Computational Biology and Bioinformatics* 5:4, 546–556.

[25] Kim, E., Wheeler, T., and Kececioglu, J. 2009. Learning models for aligning protein sequences with predicted secondary structure. Proceedings of the 13th *Conference on Research in Computational Molecular Biology* (RECOMB), Springer LNBI 5541, 512–531.

[26] Lu, Y., and Sze, S.-H. 2008. Multiple sequence alignment based on profile alignment of intermediate sequences. *Journal of Computational Biology* 15:7, 676–777.

[27] Lüthy, R., McLachlan, A.D., and Eisenberg, D. 1991. 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.

[28] Makhorin, A. 2005. GNU Linear Programming Kit, release 4.8.
http://www.gnu.org/software/glpk

[29] Miller, W., and Myers, E.W. 1988. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology* 50, 97–120.

[30] Mizuguchi, K., Deane, C.M., Blundell, T.L., and Overington, J.P. 1998. HOMSTRAD: a database of protein structure alignments for homologous families. *Protein Science* 7, 2469–2471.

[31] Notredame, C., Higgins, D.G., and Heringa, J. 2000. T-Coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology* 302, 205–217.

[32] Pei, J., and Grishin, N.V. 2007. PROMALS: towards accurate multiple sequence alignments of distantly related proteins. *Bioinformatics* 23:7, 802–808.

[33] Sander, C., and Schneider, R. 1991. Database of homology-derived protein structures and the structural meaning of sequence alignment. *Proteins: Structure, Function, and Genetics* 9, 56–68.

[34] Simossis, V.A., and Heringa, J. 2005. PRALINE: a multiple sequence alignment toolbox that integrates homology-extended and secondary structure information. *Nucleic Acids Research* 33, W289–294.

[35] Söding, J. 2005. Protein homology detection by HMM-HMM comparison. *Bioinformatics* 21:7, 951–960.

[36] Starret, D. 2008. *Optimal Alignment of Multiple Sequence Alignments.* PhD dissertation, Department of Computer Science, The University of Arizona.

[37] Sun, F., Fernández-Baca, D., and Yu, W. 2004. Inverse parametric sequence alignment. *Journal of Algorithms* 53, 36–54.

[38] Thompson, J.D., Higgins, D.G., and Gibson, T.J. 1994. `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.

[39] Van Walle, I., Lasters, I., and Wyns, L. 2004. `Align-m`: A new algorithm for multiple alignment of highly divergent sequences. *Bioinformatics* 20, 1428–1435.

[40] Wheeler, T.J. 2009. *Efficient Construction of Accurate Multiple Alignments and Large-Scale Phylogenies*. PhD dissertation, Department of Computer Science, The University of Arizona.

[41] Wheeler, T.J., and Kececioglu, J.D. 2007a. Multiple alignment by aligning alignments. Proceedings of the 15th ISCB *Conference on Intelligent Systems for Molecular Biology* (ISMB), *Bioinformatics* 23, i559–i568.

[42] Wheeler, T.J., and Kececioglu, J.D. 2007b. `Opal`: software for aligning multiple biological sequences. Version 0.3.7. `http://opal.cs.arizona.edu`

[43] Yu, C.-N., Joachims, T., Elber, R., and Pillardy, J. 2008. Support vector training of protein alignment models. *Journal of Computational Biology* 15:7, 867–880.

[44] Zhou, H., and Zhou, Y. 2005. `SPEM`: improving multiple sequence alignment with sequence profiles and predicted secondary structures. *Bioinformatics* 21, 3615–3621.