Support Vector Training of Protein Alignment Models

Chun-Nam John Yu[1], Thorsten Joachims[1], Ron Elber[1], and Jaroslaw Pillardy[2]

[1] Dept. of Computer Science, Cornell University, Ithaca NY 14853, USA
{cnyu,tj,ron}@cs.cornell.edu
[2] Cornell Theory Center, Cornell University, Ithaca NY 14853, USA
jarekp@tc.cornell.edu

Abstract. Sequence to structure alignment is an important step in homology modeling of protein structures. Incorporation of features like secondary structure, solvent accessibility, or evolutionary information improve sequence to structure alignment accuracy, but conventional generative estimation techniques for alignment models impose independence assumptions that make these features difficult to include in a principled way. In this paper, we overcome this problem using a Support Vector Machine (SVM) method that provides a well-founded way of estimating complex alignment models with hundred-thousands of parameters. Furthermore, we show that the method can be trained using a variety of loss functions. In a rigorous empirical evaluation, the SVM algorithm outperforms the generative alignment method SSALN, a highly accurate generative alignment model that incorporates structural information. The alignment model learned by the SVM aligns 47% of the residues correctly and aligns over 70% of the residues within a shift of 4 positions.

Keywords: Machine learning, Pairwise sequence alignment, Protein structure prediction

1 Introduction

Sequence to structure alignment is a crucial step in building accurate three-dimensional protein models in homology modeling. Most alignment methods are based on dynamic programming on a linear cost model. In the simplest cases, such as alignment with the BLOSUM matrices, the linear model specifies costs for substituting one amino acid with another and for inserting a gap. The choice of these costs greatly determines the quality of alignments. While it is well understood how to estimate the substitution costs for such simple models, sequence to structure alignment requires more complex cost models to take advantage of the structural information available. For these complex cost models, conventional generative estimation techniques like Hidden Markov Models (HMM) are difficult to use due to the independence assumptions they make, for example, in assuming that the amino acid sequence and the secondary structure labels of a protein are generated by independent processes.

This paper explores a Support Vector Machine (SVM) method for learning an application specific cost model from training data for sequence to structure alignment. The advantages of this SVM method over conventional generative techniques are threefold. First, unlike conventional generative estimation techniques, the SVM method does not require independence assumptions among features and therefore provides a well-founded way to learn cost models where each aligned position is described not only by its amino acid identity, but by a potentially high-dimensional feature vector. This feature vector may describe additional properties of the aligned position (e.g. predicted secondary structure) as well as properties of surrounding aligned positions (e.g. the previous amino acid is hydrophobic). This provides great flexibility in building expressive models. Second, the SVM method inherits the benefits of conventional classification SVMs, in particular its robustness to overfitting for high-dimensional and sparse data. Third, the SVM method allows optimizing for different loss functions. This allows accounting for uncertainty in the training data, as well as specifying which types of alignment errors are more costly than others.

The work reported in the following shows that the SVM algorithm can be used to learn highly accurate alignment models for sequence to structure alignments. It also provides the first large-scale implementation and empirical validation of this SVM alignment algorithm, extending the basic algorithm first proposed in [1, 2] to include loss functions. We show that this SVM algorithm can effectively learn alignment models with hundreds of thousands of features, outperforming the accuracy of state-of-the-art generative estimation techniques [3]. Finally, we show that loss functions can be incorporated into the SVM training problem while maintaining polynomial runtime guarantees. We find that the use of application-dependent loss functions during training, in particular by only counting alignment errors if the shift from the correct residue is more than 4, is effective in modeling the uncertainty in the training data. The training and alignment program of this work is available for download at `http://svmlight.joachims.org`.

2 Related Work

Conventional estimation techniques for alignment models (see e.g. [4–8]) take the view of a generative probabilistic model. A generative alignment model (e.g. HMM) aims to model the process that generates the data as the joint probability distribution $P(S, T, Y)$, where $\mathbf{s} = (s^1, ..., s^{|\mathbf{s}|})$ and $\mathbf{t} = (t^1, ..., t^{|\mathbf{t}|})$ are two sequences and $\mathbf{y}$ is the alignment. We denote the length of a sequence with $|.|$. If $P(S, T, Y)$ (or a good estimate thereof) is known,

$$\text{argmax}_{\mathbf{y}}\, P(S = \mathbf{s}, T = \mathbf{t}, Y = \mathbf{y}) \tag{1}$$

predicts an alignment $\mathbf{y}$ from two sequences $\mathbf{s}$ and $\mathbf{t}$. To make estimation of $P(S, T, Y)$ tractable, it is decomposed by making independence assumptions on the process that generates $\mathbf{s}$ and $\mathbf{t}$. While this leads to efficient and simple estimation problems, the independence assumptions restrict the interactions within the sequences $\mathbf{s}$ and $\mathbf{t}$ that we could model.

Machine learning research over the last decade has provided substantial evidence that discriminative learning (e.g. SVMs, MaxEnt classifiers) typically produces more accurate rules than generative learning (e.g. naïve Bayes classifiers, HMMs) (see e.g. [9–11]). This can be explained as follows. Since $P(Y|S,T)$ is already sufficient for making an optimal prediction

$$\mathrm{argmax}_{\mathbf{y}}\, P(Y=\mathbf{y}|S=\mathbf{s},T=\mathbf{t}) \ , \tag{2}$$

modeling the joint distribution of the input sequences S and T is not necessary, and generative methods might be wasting effort in trying to do so. Discriminative learning applied to the alignment problem would directly estimate $P(Y|S,T)$ or a related discriminant function, thus focusing on the relevant part of the estimation problem.

Only few approaches to discriminative training of alignment models exist to date. While not motivated from this learning theoretical perspective, work on *inverse alignment* is closely related, since our SVM method can be viewed as solving an inverse alignment problem. Inverse alignment is the task of finding a cost model under which a given alignment algorithm outputs a desired alignment $\mathbf{y}$ for sequences $\mathbf{s}$ and $\mathbf{t}$. This problem was first formulated in [12]. They discuss inverse alignment in the context of parametric sequence alignment and identify geometric properties of the space of cost model. In more detail, the work in [13] analyzes the space of models and shows that some aspects of its complexity grow only polynomially.

The first concrete algorithm for inverse sequence alignment was proposed in [14]. While they prove that their algorithm finds a consistent cost model in polynomial time, their algorithm is limited to particular cost models with at most 3 parameters.

The work presented in this paper follows the Structural SVM algorithm first proposed in [1, 2] for sequence alignment and later generalized to a wide class of multivariate prediction problems [15, 10]. In this paper we present the first large-scale empirical evaluation of this type of algorithm for alignment. We also extend the algorithm to optimize particular loss functions, and show how the resulting optimization problems can be solved in polynomial time.

Related to our SVM approach are Conditional Random Fields (CRFs), which have recently been proposed for sequence alignment as well [16, 17]. While CRFs share with SVMs the benefits of discriminative training, they do not allow the use of application-dependent loss functions.

Independent of the work on Structural SVMs in the machine learning community, recently an algorithm for inverse alignment was proposed in [18]. Their formulation of the problem is similar to a Structural SVM and the algorithm resembles the cutting-plane method used for training Structural SVMs. However, their approach is based on a linear programming formulation instead of the quadratic programming formulation used in SVMs. Furthermore, their approach to handling infeasibilities in the resulting optimization problem is different and it is unclear how it relates to an intuitively meaningful loss function. While they give empirical results, they are on a scale of 10 training examples and 212 features.

In the following, we will explore models trained over thousands of examples and hundred-thousands of features.

3 Sequence Alignment

We begin by introducing the class of alignment models considered in this paper. Since we focus on the problem of sequence to structure alignment, we will describe the model in these terms. However, the methods can obviously be used for other applications as well. Let $(\mathbf{s}, \mathbf{t})$ be a pair of target and template sequence that we wish to align. For an alignment $\mathbf{y}$ of $(\mathbf{s}, \mathbf{t})$, we write $\mathbf{y}$ as a sequence of alignment operations $(y^1, y^2, ..., y^{|\mathbf{y}|})$. Each y^k is an alignment operation of the form (i, j), where i, j are positions of characters in $\mathbf{s}$ and $\mathbf{t}$ respectively, or a special gap symbol '$-$'.

We consider alignment algorithms (e.g. [19]) that optimize a linear scoring function $D_{\mathbf{w}}(\mathbf{y}, \mathbf{s}, \mathbf{t}) = \mathbf{w} \cdot \Psi(\mathbf{y}, \mathbf{s}, \mathbf{t})$, where Ψ is a function that maps an alignment $\mathbf{y}$ of $\mathbf{s}$ and $\mathbf{t}$ to a feature vector, and $\mathbf{w}$ is a given cost vector. Note that $\mathbf{w}$ contains the parameters of the alignment model that we will learn. We require that $\Psi(\mathbf{y}, \mathbf{s}, \mathbf{t})$ be linear in the individual alignment operations y^k within $\mathbf{y}$, written in terms of equations,

$$\Psi(\mathbf{y}, \mathbf{s}, \mathbf{t}) = \sum_{k=1}^{|\mathbf{y}|} \phi(y^k, \mathbf{s}, \mathbf{t}) \quad , \tag{3}$$

where ϕ is a function that maps each individual alignment operation onto a feature vector. Note that this feature vector can be any function that depends on the operation y^k and the *full* target and template sequences, not just the current positions that are aligned by y^k. In general, alignment algorithms compute

$$\underset{y \in \mathcal{Y}}{\operatorname{argmax}} \left[\mathbf{w} \cdot \Psi(\mathbf{y}, \mathbf{s}, \mathbf{t})\right] = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \left[\sum_{k=1}^{|\mathbf{y}|} \mathbf{w} \cdot \phi(y^k, \mathbf{s}, \mathbf{t})\right] \tag{4}$$

to determine the alignment, where $\mathcal{Y}$ is the set of all possible (local or global, as desired) alignments between $\mathbf{s}$ and $\mathbf{t}$. This is typically computed using dynamic programming (e.g. [19]). Note that our setting includes the common scenarios of alignment with substitution matrices such as BLOSUM as a special case, where the function $\phi(y^k, \mathbf{s}, \mathbf{t})$ return a sparse vector with exactly one '1' that corresponds to the particular substitution or gap score in $\mathbf{w}$. However, we will consider richer feature mappings ϕ that go beyond amino-acid identity and that include structural information of the template sequence.

4 Discriminative Training of Alignment Models

In the above section, the vector $\mathbf{w}$ parameterizes the scoring function D and has crucial influence on the quality of alignments between $\mathbf{s}$ and $\mathbf{t}$. In the following, we aim to learn $\mathbf{w}$ from a set of training examples

$$\mathcal{S} = ((\mathbf{s}_1, \mathbf{t}_1, \mathbf{y}_1), (\mathbf{s}_2, \mathbf{t}_2, \mathbf{y}_2), ..., (\mathbf{s}_n, \mathbf{t}_n, \mathbf{y}_n)) \tag{5}$$

of sequence pairs $(\mathbf{s}_i, \mathbf{t}_i)$ for which the (approximately) correct alignment $\mathbf{y}_i$ is known. This training set is assumed to be generated independently and identically distributed (i.i.d.) according to some unknown distribution $P(S, T, Y)$. Thinking of a sequence alignment algorithm as a function,

$$h_{\mathbf{w}}(\mathbf{s}, \mathbf{t}) = \text{argmax}_{\mathbf{y} \in \mathcal{Y}} \left[\mathbf{w} \cdot \Psi(\mathbf{y}, \mathbf{s}, \mathbf{t}) \right] \tag{6}$$

maps a given sequence pair $(\mathbf{s}, \mathbf{t})$ to an alignment $\mathbf{y}$. Our goal is to find a parameter vector $\mathbf{w}$ so that the predicted alignment $h_{\mathbf{w}}(\mathbf{s}, \mathbf{t})$ matches the correct alignment on new test data as well as possible. In particular, we want to find a $\mathbf{w}$ that minimizes the expected loss (i.e. risk)

$$R_P(h_{\mathbf{w}}) = \int \Delta(\mathbf{y}, h_{\mathbf{w}}(\mathbf{s}, \mathbf{t})) \; dP(S, T, Y) \ , \tag{7}$$

where $\Delta(\mathbf{y}, \mathbf{y}')$ is a user defined (non-negative) loss function that quantifies how "bad" it is to predict $\mathbf{y}'$ when $\mathbf{y}$ is the correct alignment. For example, one may choose $\Delta(\mathbf{y}, \mathbf{y}')$ to be 1 minus the Q-score (Q-score is the fraction of match operations from $\mathbf{y}$ that are also contained in $\mathbf{y}'$).

Following the principle of (Structural) Empirical Risk Minimization [20], finding a $\mathbf{w}$ that predicts well on new data can be achieved by minimizing the empirical loss (i.e. the training error) $R_{\mathcal{S}}(h_{\mathbf{w}}) = \sum_{i=1}^{n} \Delta(\mathbf{y}_i, h_{\mathbf{w}}(\mathbf{s}_i, \mathbf{t}_i))$ on the training set $\mathcal{S}$. This leads to the computational problem of finding the $\mathbf{w}$ which minimizes $R_{\mathcal{S}}(h_{\mathbf{w}})$ as follows.

5 Structural SVMs for Sequence Alignment

In the framework of structural SVMs [10], we formulate the problem of finding the parameters $\mathbf{w}$ that minimizes the empirical loss $R_{\mathcal{S}}(h_{\mathbf{w}})$ of the sequence alignment algorithm as the following optimization problem:

$$\min_{\mathbf{w}, \xi} \ \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i \tag{8}$$

$$s.t. \ \forall i \in \{1, .., n\} \, \forall \hat{\mathbf{y}} \in \mathcal{Y}_i \backslash \{\mathbf{y}_i\} : \mathbf{w} \cdot (\Psi(\mathbf{y}_i, \mathbf{s}_i, \mathbf{t}_i) - \Psi(\hat{\mathbf{y}}, \mathbf{s}_i, \mathbf{t}_i)) \geq \Delta(\mathbf{y}_i, \hat{\mathbf{y}}) - \xi_i$$

The objective is the conventional regularized risk used in SVMs. The constraints state that the score $\mathbf{w} \cdot \Psi(\mathbf{y}_i, \mathbf{s}_i, \mathbf{t}_i)$ of the correct alignment $\mathbf{y}_i$ must be greater than the score $\mathbf{w} \cdot \Psi(\hat{\mathbf{y}}, \mathbf{s}_i, \mathbf{t}_i)$ of all alternative alignments $\hat{\mathbf{y}}$. Note that $\mathcal{Y}_i$ (i.e. the set of all possible alignments for example i) depends on whether the alignment is local or global. However, the optimization problem is well-formed in either case.

Unlike the formulation in [2], our new formulation includes a loss function $\Delta(\mathbf{y}_i, \hat{\mathbf{y}})$ that scales the desired difference in score. Intuitively, the larger the loss of an incorrect alignment $\hat{\mathbf{y}}$, the further should the score be away from that of the correct alignment $\mathbf{y}_i$. This method for including a loss function is analogous to proposals for other structured prediction problems [21, 10]. ξ_i is a slack variable shared among constraints from the same example, since in general the constraint system is not feasible. Following the proof in [10], it is easy to see that following result holds.

Theorem 1. *If $(\mathbf{w}^*, \xi^*)$ is the solution of the optimization problem in (8), the sum of the slack variables ξ_i^* is an upper bound on the training loss, $R_{\mathcal{S}}(h_{\mathbf{w}}) \leq \sum_{i=1}^{n} \xi_i^*$.*

This shows that our formulation minimizes training loss, while the SVM-style regularization with the norm of $\mathbf{w}$ in the objective provides protection against overfitting for high-dimensional $\mathbf{w}$. The parameter C allows the user to control the trade-off between training error and regularization.

5.1 Efficient Training Algorithm

While it is easy to see that the optimization problem in (8) is convex, it unfortunately has an exponential number of constraints. This results from the fact that there are exponentially many "wrong" alignments $\mathcal{Y}_i \backslash \{\mathbf{y}_i\}$ for each given pair of sequences $(\mathbf{s}_i, \mathbf{t}_i)$. Any attempt to solve this type of optimization problem using standard methods that require enumerating all constraints is obviously not tractable for sequences and training sets of interesting size.

Despite the exponential size, however, it has been shown that cutting-plane algorithms can be used to efficiently approximate the optimal solution of this type of optimization problem [10]. An adaptation of this algorithm to the problem of sequence alignment is given in Fig. 1. The algorithm iteratively constructs a subset of all constraints from (8) until this subset constrains the feasible region enough to ensure an ϵ-accurate solution. The desired precision ϵ is provided by the user. In particular, the algorithm starts with an empty set K of constraints. It then adds the most violated constraint among the exponentially many for each example. If no constraint exists that is violated by more than ϵ, the algorithm terminates. Otherwise, it solves the optimization problem over the current set K and repeats. Adapting the result from [10], it can be proved that only a polynomial number of constraints will be added before the algorithm converges.

Theorem 2. *For any $\epsilon > 0$, $C > 0$, and any training sample $\mathcal{S} = ((\mathbf{s}_1, \mathbf{t}_1, \mathbf{y}_1), \ldots, (\mathbf{s}_n, \mathbf{t}_n, \mathbf{y}_n))$, the algorithm in Fig. 1 converges after adding at most $\max\left\{2n\bar{\Delta}/\epsilon, 8C\bar{\Delta}R^2/\epsilon^2\right\}$ constraints to K, where $R = \max_{i,\mathbf{y}} ||\Psi(\mathbf{y}, \mathbf{s}_i, \mathbf{t}_i)||$ and $\bar{\Delta}$ is an upper bound on the loss function $\Delta(\mathbf{y}_i, \hat{\mathbf{y}})$.*

One crucial aspect of the algorithm, however, is the use of an oracle (often called a separation oracle in optimization theory) that can find the most violated constraint among the exponentially many in polynomial time. This is equivalent to the argmax problem in the algorithm. The following section shows that this argmax can be computed efficiently for a large class of loss functions.

```
Input: sequence pairs $(\mathbf{s}_1,\mathbf{t}_1),...,(\mathbf{s}_n,\mathbf{t}_n)$, correct alignments $\mathbf{y}_1,...,\mathbf{y}_n$, tolerated error
  $\epsilon \geq 0$.
$K = \emptyset$, $\mathbf{w} = 0$, $\xi = 0$
repeat
    – $K_{\text{org}} = K$
    – for $i$ from 1 to $n$
        • $\hat{\mathbf{y}} = \text{argmax}_{\hat{\mathbf{y}} \in \mathcal{Y}_i \setminus \{\mathbf{y}_i\}} [\Delta(\mathbf{y}_i, \hat{\mathbf{y}}) + \mathbf{w} \cdot (\Psi(\hat{\mathbf{y}}, \mathbf{s}, \mathbf{t}) - \Psi(\mathbf{y}_i, \mathbf{s}_i, \mathbf{t}_i))]$
        • if $\mathbf{w} \cdot (\Psi(\mathbf{y}_i, \mathbf{s}_i, \mathbf{t}_i) - \Psi(\hat{\mathbf{y}}, \mathbf{s}, \mathbf{t})) < \Delta(\mathbf{y}_i, \hat{\mathbf{y}}) - \xi_i - \epsilon$
            * $K = K \cup \{\mathbf{w} \cdot (\Psi(\mathbf{y}_i, \mathbf{s}_i, \mathbf{t}_i) - \Psi(\hat{\mathbf{y}}, \mathbf{s}, \mathbf{t})) \geq \Delta(\mathbf{y}_i, \hat{\mathbf{y}}) - \xi_i - \epsilon\}$
            * $(\mathbf{w}, \xi) = \text{argmin}_{\mathbf{w},\xi} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi_i$ subject to $K$.
until $(K = K_{\text{org}})$
Output: $\mathbf{w}$
```

Fig. 1. Cutting-plane algorithm for solving the SVM optimization problem.

5.2 Loss Functions

We first introduce the Q-loss function, which we denote as $\Delta_{\text{Q}}(\mathbf{y}_i, \hat{\mathbf{y}})$ for the loss between a correct alignment $\mathbf{y}_i$ and a predicted alignment $\hat{\mathbf{y}}$. The Q-loss measures the proportion of incorrect matches in a predicted alignment, which we want to minimize. Writing $\hat{\mathbf{y}}$ as a sequence of alignment operations $\hat{\mathbf{y}} = (\hat{y}^1, \hat{y}^2, ..., \hat{y}^{|\hat{\mathbf{y}}|})$, we can decompose the Q-loss $\Delta_{\text{Q}}(\mathbf{y}_i, \hat{\mathbf{y}})$ into a sum of losses on individual alignment operations $\Delta_{\text{Q}}(\mathbf{y}_i, \hat{\mathbf{y}}) = 1 - \sum_{k=1}^{|\hat{\mathbf{y}}|} \delta_{\text{Q}}(\mathbf{y}_i, \hat{y}^k)$. The function $\delta_{\text{Q}}(\mathbf{y}_i, \hat{y}^k)$ returns $1/M$ when $\hat{y}^k$ is a match contained in the correct alignment $\mathbf{y}_i$, and 0 otherwise. M is the number of matches in $\mathbf{y}_i$. With this decomposition, we can rewrite the computation of the most violated constraint in the cutting plane algorithm as:

$$\underset{\hat{\mathbf{y}} \in \mathcal{Y}_i \setminus \{\mathbf{y}_i\}}{\text{argmax}} \left[\sum_{k=1}^{|\hat{\mathbf{y}}|} (\mathbf{w} \cdot \phi(\hat{y}^k, \mathbf{s}_i, \mathbf{t}_i)) - \delta_{\text{Q}}(\mathbf{y}_i, \hat{y}^k) \right] + 1 - \mathbf{w} \cdot \Psi(\mathbf{y}_i, \mathbf{s}_i, \mathbf{t}_i) \ . \tag{9}$$

Since the non-constant term in the argmax decomposes into a sum over individual alignment operations $\hat{y}^k$, we can apply dynamic programming similar to (4) with operation costs modified to $\mathbf{w} \cdot \phi(\hat{y}^k, \mathbf{s}_i, \mathbf{t}_i) - \delta_{\text{Q}}(\mathbf{y}_i, \hat{y}^k)$ for each $\hat{y}^k$. Note that δ_{Q} can be computed efficiently using table lookup.

Another interesting loss function that we would like to consider is the Q_4-loss function, where we count a match as correct even if it is slightly shifted, in particular, shifted by not more than 4 positions. Suppose we have an alignment operation $\hat{y}^k = (i, j)$ in $\hat{\mathbf{y}}$. The shift of $\hat{y}^k$ is less than 4 if there is some match operation $y^l = (u, j)$ in $\mathbf{y}$ with $|i - u| \leq 4$. Similar to the Q-loss, we can decompose Q_4 linearly in its alignment operations as $\Delta_{\text{Q}_4}(\mathbf{y}_i, \hat{\mathbf{y}}) = 1 - \sum_{k=1}^{|\hat{\mathbf{y}}|} \delta_{\text{Q}_4}(\mathbf{y}_i, \hat{y}^k)$, where $\delta_{\text{Q}_4}(\mathbf{y}_i, \hat{y}^k)$ is $1/M$ if $\hat{y}^k$ is a match operation contained in $\mathbf{y}$ with a shift of 4 or less, and 0 otherwise. The same dynamic programming algorithm applies to the computation of most violated constraint with Q_4-loss, since Q_4-loss decomposes in the same way as the Q-loss. Again, note that δ_{Q_4} can again be computed efficiently by table lookup of matched characters in the range $(i-4, j), ..., (i+4, j)$ within the correct alignment.

6 Experiments

The following experiments evaluate the SVM alignment algorithm on a sequence to structure alignment task. We evaluate whether the algorithm can effectively and efficiently learn complex alignment models with hundred-thousands of features, how optimizing to different loss functions might help, and how the algorithm compares to conventional methods.

In all our experiments, we train the algorithm on a training set, select any parameters and models based on a validation set, and then report performance on an independent test set. Beyond the choice of model $\phi(y^k, \mathbf{s}_i, \mathbf{t}_i)$ and loss function $\Delta(\mathbf{y}, \mathbf{y}')$, our method has only a single parameter to tune, namely the regularization parameter C. We train alignment models with C ranging from 1 to 2^{15}, in powers of 2, all to precision $\epsilon = 0.01$. We then pick the best model based on the performance on the validation set, and report its performance on the test set.

The training and validation sets are the same as those used in [3]. The data set contains 1379 target sequences, and each target sequence $\mathbf{s}$ has one or more template structures $\mathbf{t}$ associated with it. Structural alignments between target and template are generated using the CE program [22], and one example $(\mathbf{s}, \mathbf{t}, \mathbf{y})$ is generated whenever the structural alignment $\mathbf{y}$ between the structure of $\mathbf{s}$ and $\mathbf{t}$ has a CE Z-score of at least 4.5. The data set is randomly split into two sets, namely a training set with examples from 690 targets and a validation set with examples from 689 targets. The resulting training set contains 5119 examples (i.e. pairwise alignments) while the validation set contains 5169 examples.

The test set is based on a database of protein structures that is used by the modeling program LOOPP (`http://cbsuapps.tc.cornell.edu/loopp.aspx`). We select 4185 structures from the new PDB structures released between June 2005 and June 2006 via clustering. These structures serve as target sequences in our test set and none of them appear in the training or validation sets since they were developed earlier. Each of these 4185 structures is aligned against all other structures using the structural alignment program TM-align [23]. Pairs that score 0.5 or better are considered homologous and are added to the test set. The selected pairs are then aligned by the structural alignment program CE. Only alignments that have CE Z-score higher than 4.5 are included in the final test case, providing a total of 29764 alignments to consider.

As described in the following, we use structural annotations as features in our alignment models. The structural annotations of all the target sequences, i.e., the secondary structure and the relative exposed surface area, are predicted by the SABLE program [24]. There are 3 types of secondary structure predicted and the relative exposed surface areas are binned into 4 types. The structural annotations used in the template structures are computed by the program DSSP

[25]. The secondary structures are binned into 5 types while the exposed surface areas are binned into 6 types.

6.1 Can the SVM algorithm learn complex models effectively?

In our first set of experiments we evaluate models $\phi(y^k, \mathbf{s}, \mathbf{t})$ of increasing complexity and number of features. Our focus is on exploring how far we can push the complexity of the model and still be able to train them efficiently and effectively.

We use $R^i_{\mathbf{s}}, S^i_{\mathbf{s}}, A^i_{\mathbf{s}}$ to denote the residue, predicted secondary structure, and predicted exposed surface area at the ith position of the target sequence, and $R^j_{\mathbf{t}}, S^j_{\mathbf{t}}, A^j_{\mathbf{t}}$ to deonte the residue, actual secondary structure, and actural exposed surface area at the jth position of the template structure. We also use $\mathbf{R}, \mathbf{S}, \mathbf{A}$ to denote the set of possible values for residue, secondary structure, and exposed surface area.

Substitution Cost Models For the substitution costs, we consider the following six models for $\phi(y^k, \mathbf{s}, \mathbf{t})$. Since the length of alignments of different examples varies greatly, we normalize each ϕ by dividing with $|\mathbf{s}| + |\mathbf{t}|$.

Simple: In this alignment model we only consider the subsitution cost of single features. Let $y^k = (i, j)$ be a match operation. We define $\phi(y^k, \mathbf{s}, \mathbf{t})$ to be

$$\begin{aligned}
&\phi_{Simple}(y^k, \mathbf{s}, \mathbf{t}) \\
&= \sum_{r_1, r_2 \in \mathbf{R}} \mathbb{I}[R^i_{\mathbf{s}} = r_1, R^j_{\mathbf{t}} = r_2] + \sum_{r_1 \in \mathbf{R}, s_2 \in \mathbf{S}} \mathbb{I}[R^i_{\mathbf{s}} = r_1, S^j_{\mathbf{t}} = s_2] + \sum_{r_1 \in \mathbf{R}, a_2 \in \mathbf{A}} \mathbb{I}[R^i_{\mathbf{s}} = r_1, A^j_{\mathbf{t}} = a_2] \\
&+ \sum_{s_1 \in \mathbf{S}, r_2 \in \mathbf{R}} \mathbb{I}[S^i_{\mathbf{s}} = s_1, R^j_{\mathbf{t}} = r_2] + \sum_{s_1, s_2 \in \mathbf{S}} \mathbb{I}[S^i_{\mathbf{s}} = s_1, S^j_{\mathbf{t}} = s_2] + \sum_{s_1 \in \mathbf{S}, a_2 \in \mathbf{A}} \mathbb{I}[S^i_{\mathbf{s}} = s_1, A^j_{\mathbf{t}} = a_2] \\
&+ \sum_{a_1 \in \mathbf{A}, r_2 \in \mathbf{R}} \mathbb{I}[A^i_{\mathbf{s}} = a_1, R^j_{\mathbf{t}} = r_2] + \sum_{a_1 \in \mathbf{A}, s_2 \in \mathbf{S}} \mathbb{I}[A^i_{\mathbf{s}} = a_1, S^j_{\mathbf{t}} = s_2] + \sum_{a_1, a_2 \in \mathbf{A}} \mathbb{I}[A^i_{\mathbf{s}} = a_1, A^j_{\mathbf{t}} = a_2] \ ,
\end{aligned}$$

where $\mathbb{I}[\rho]$ is a function that returns a vector with '1' in the position designated to ρ if the boolean expression ρ is true, and returns '0' otherwise and in all other positions. For example, $\mathbb{I}[R^3_{\mathbf{s}} = \text{'A'}, S^7_{\mathbf{t}} = \text{'}\alpha\text{'}]$ returns '1' in the particular dimension corresponding to $\mathbb{I}[R^i_{\mathbf{s}} = \text{'A'}, S^j_{\mathbf{t}} = \text{'}\alpha\text{'}]$, if $y^k = (3, 7)$ aligns the residue alanine 'A' in $\mathbf{s}$ with an alpha helix 'α' in $\mathbf{t}$. Otherwise, it returns '0' in this dimension. For all other dimensions it always returns '0'. Note that each such dimension corresponds to a particular position in cost vector $\mathbf{w}$. Note also that each feature vector $\phi_{Simple}(y^k, \mathbf{s}, \mathbf{t})$ has exactly 9 '1's corresponding to the 9 terms in the sum, and is zero elsewhere.

Anova2: In this more complex feature vector we take the interactions between pairs of structural annotations at the same position in the sequence into account.

We define $\phi(y^k, \mathbf{s}, \mathbf{t})$ to be

$$\phi_{\mathit{Anova2}}(y^k, \mathbf{s}, \mathbf{t}) = \sum_{r_1, r_2 \in \mathbf{R}, s_1, s_2 \in \mathbf{S}} \mathbb{I}[R^i_{\mathbf{s}} = r_1, S^i_{\mathbf{s}} = s_1, R^j_{\mathbf{t}} = r_2, S^j_{\mathbf{t}} = s_2]$$

$$+ \sum_{\substack{r_1 \in \mathbf{R}, a_2 \in \mathbf{A}, \\ s_1, s_2 \in \mathbf{S}}} \mathbb{I}[R^i_{\mathbf{s}} = r_1, S^i_{\mathbf{s}} = s_1, S^j_{\mathbf{t}} = s_2, A^j_{\mathbf{t}} = a_2] + \sum_{\substack{r_1, r_2 \in \mathbf{R}, s_1 \in \mathbf{S}, \\ a_2 \in \mathbf{A}}} \mathbb{I}[R^i_{\mathbf{s}} = r_1, S^i_{\mathbf{s}} = s_1, A^j_{\mathbf{t}} = a_2, R^j_{\mathbf{t}} = r_2]$$

$$+ \sum_{\substack{r_2 \in \mathbf{R}, a_1 \in \mathbf{A}, \\ s_1, s_2 \in \mathbf{S}}} \mathbb{I}[S^i_{\mathbf{s}} = s_1, A^i_{\mathbf{s}} = a_1, R^j_{\mathbf{t}} = r_2, S^j_{\mathbf{t}} = s_2] + \sum_{\substack{s_1, s_2 \in \mathbf{S}, \\ a_1, a_2 \in \mathbf{A}}} \mathbb{I}[S^i_{\mathbf{s}} = s_1, A^i_{\mathbf{s}} = a_1, S^j_{\mathbf{t}} = s_2, A^j_{\mathbf{t}} = a_2]$$

$$+ \sum_{\substack{r_2 \in \mathbf{R}, s_1 \in \mathbf{S}, \\ a_1, a_2 \in \mathbf{A}}} \mathbb{I}[S^i_{\mathbf{s}} = s_1, A^i_{\mathbf{s}} = a_1, A^j_{\mathbf{t}} = a_2, R^j_{\mathbf{t}} = r_2] + \sum_{\substack{r_1, r_2 \in \mathbf{R}, s_2 \in \mathbf{S}, \\ a_2 \in \mathbf{A}}} \mathbb{I}[A^i_{\mathbf{s}} = a_1, R^i_{\mathbf{s}} = r_1, R^j_{\mathbf{t}} = r_2, S^j_{\mathbf{t}} = s_2]$$

$$+ \sum_{\substack{r_1 \in \mathbf{R}, s_2 \in \mathbf{S}, \\ a_1, a_2 \in \mathbf{A}}} \mathbb{I}[A^i_{\mathbf{s}} = a_1, R^i_{\mathbf{s}} = r_1, S^j_{\mathbf{t}} = s_2, A^j_{\mathbf{t}} = a_2] + \sum_{\substack{r_1, r_2 \in \mathbf{R}, \\ a_1, a_2 \in \mathbf{A}}} \mathbb{I}[A^i_{\mathbf{s}} = a_1, R^i_{\mathbf{s}} = r_1, A^j_{\mathbf{t}} = a_2, R^j_{\mathbf{t}} = r_2] \ .$$

For example, the term $\mathbb{I}[R^i_{\mathbf{s}} = r_1, S^i_{\mathbf{s}} = s_1, R^j_{\mathbf{t}} = r_2, S^j_{\mathbf{t}} = s_2]$ returns '1' in the appropriate position, if $y^k = (i, j)$ aligns residue of type r_1 in secondary structure s_1 in the target with residue of type r_2 in secondary structure s_2 in the template. These features capture pairwise interaction of structural annotations within the same sequence.

Tensor: In this even more complex alignment model we consider the interaction of all three structural annotations. Note that there is only one non-zero feature in this feature vector.

$$\phi_{\mathit{Tensor}}(y^k, \mathbf{s}, \mathbf{t}) = \sum_{r_1, r_2 \in \mathbf{R}, s_1, s_2 \in \mathbf{S}, a_1, a_2 \in \mathbf{A}} \mathbb{I}[R^i_{\mathbf{s}} = r_1, S^i_{\mathbf{s}} = s_1, A^i_{\mathbf{s}} = a_1, R^j_{\mathbf{t}} = r_2, S^j_{\mathbf{t}} = s_2, A^j_{\mathbf{t}} = a_2]$$

Simple+Anova2: This alignment model is the union of the features in the *Simple* and the *Anova2* alignment models, i.e. $\phi_{\mathit{Simple}}(y^k, \mathbf{s}, \mathbf{t}) + \phi_{\mathit{Anova2}}(y^k, \mathbf{s}, \mathbf{t})$.

Simple+Anova2+Tensor: This alignment model is the union of all features in the first three alignment models, i.e. $\phi_{\mathit{Simple}}(y^k, \mathbf{s}, \mathbf{t}) + \phi_{\mathit{Anova2}}(y^k, \mathbf{s}, \mathbf{t}) + \phi_{\mathit{Tensor}}(y^k, \mathbf{s}, \mathbf{t})$.

Window: On top of the *Simple+Anova2+Tensor* feature vector, we add several terms involving the substitution score of a sliding window of features centered around positions i and j.

$$\begin{aligned}\phi_{\mathit{Window}}(y^k, \mathbf{s}, \mathbf{t}) = {} & \phi_{\mathit{Simple}}(y^k, \mathbf{s}, \mathbf{t}) + \phi_{\mathit{Anova2}}(y^k, \mathbf{s}, \mathbf{t}) + \phi_{\mathit{Tensor}}(y^k, \mathbf{s}, \mathbf{t}) \\ & + \sum_{r_1, r_2, r_3 \in \mathbf{R}, r_4, r_5, r_6 \in \mathbf{R}} \mathbb{I}[R^{i-1}_{\mathbf{s}} = r_1, R^i_{\mathbf{s}} = r_2, R^{i+1}_{\mathbf{s}} = r_3, R^{j-1}_{\mathbf{t}} = r_4, R^j_{\mathbf{t}} = r_5, R^{j+1}_{\mathbf{t}} = r_6] \\ & + \sum_{s_1, \ldots, s_5 \in \mathbf{S}, s_6, \ldots, s_{10} \in \mathbf{S}} \mathbb{I}[S^{i-2}_{\mathbf{s}} = s_1, ..., S^{i+2}_{\mathbf{s}} = s_5, S^{j-2}_{\mathbf{t}} = s_6, ..., S^{j+2}_{\mathbf{t}} = s_{10}] \\ & + \sum_{a_1, \ldots, a_7 \in \mathbf{A}, a_8, \ldots, a_{14} \in \mathbf{A}} \mathbb{I}[A^{i-3}_{\mathbf{s}} = a_1, ..., A^{i+3}_{\mathbf{s}} = a_7, A^{j-3}_{\mathbf{t}} = a_8, ..., A^{j+3}_{\mathbf{t}} = a_{14}]\end{aligned}$$

The first sliding window term counts the occurence of substituting a triplet of residues (r_1, r_2, r_3) in the target with another triplet (r_4, r_5, r_6) in the template. The other two terms counts the occurence of substitution of two windows of secondary structures of length 5, and the occurence of substitution of two windows of surface area type of length 7 respectively. To reduce dimensionality of these features, we bin the residues into 7 groups ({A,G,P,S,T}, {C}, {D,E,N,Q}, {F,W,Y}, {H,K,R}, {I,L,M,V}, {X}, where X stands for missing value and ends of sequences), and the surface area into 2 values, exposed or buried.

Gap Cost Model All alignment models above share the following gap model. Consider the cost of opening a gap between position i and $i+1$ in the target sequence $\mathbf{s}$ against position j in the template structure $\mathbf{t}$, as depicted by the following diagram

$$\begin{array}{llllllll} \text{Target} & & \mathbf{s}^i & - & - & \cdots & - & \mathbf{s}^{i+1} \\ \text{Template} & \cdots & \mathbf{t}^j & \mathbf{t}^{j+1} & \cdots & \mathbf{t}^{j+k} & \cdots & \end{array}$$

We allow the cost of opening a gap to depend on the structural type at position j in the template structure. It also depends on the structural type of the target sequence immediately before the gap at position i as well as the structural type immediately after the gap at position $i+1$. Suppose y^k is a gap operation that opens a gap between position i and $i+1$ in the target against position j in the template sequence. The feature vector for this gap operation is:

$$\begin{aligned} \phi_{Gap}(y^k, \mathbf{s}, \mathbf{t}) = & \sum_{r_1 \in R} \mathbb{G}[R_{\mathbf{t}}^j = r_1] + \sum_{s_1 \in S, a_1 \in A} \mathbb{G}[S_{\mathbf{t}}^j = s_1, A_{\mathbf{t}}^j = a_1] \\ & + \sum_{s1, s2 \in S, a_1, a_2 \in A} \mathbb{G}[S_{\mathbf{s}}^i = s_1, A_{\mathbf{s}}^i = a_1, S_{\mathbf{s}}^{i+1} = s_2, A_{\mathbf{s}}^{i+1} = a_2] \end{aligned}$$

$\mathbb{G}$ is analogous to $\mathbb{I}$, but we use a different symbol to indicate that it maps to a different set of dimensions. The first two terms create features for the residue types and joint features of secondary structure with exposed surface area at $\mathbf{t}^j$. The term $\mathbb{G}[S_{\mathbf{s}}^i = s_1, A_{\mathbf{s}}^i = a_1, S_{\mathbf{s}}^{i+1} = s_2, A_{\mathbf{s}}^{i+1} = a_2]$ considers the structure before and after the gap. For example, $\mathbb{G}[S_{\mathbf{s}}^i = \text{‘}\alpha\text{’}, A_{\mathbf{s}}^i = \text{‘}0\text{’}, S_{\mathbf{s}}^{i+1} = \text{‘}\alpha\text{’}, A_{\mathbf{s}}^{i+1} = \text{‘}1\text{’}]$ maps to the dimension for the cost of opening a gap between a position in an alpha-helix of surface type 0 with a consecutive position in the alpha-helix with surface type 1.

The case of opening a gap in the template involves exactly the same costs, with the role of target and template reversed.

Results Table 1 shows the Q-scores of the different alignment models trained with the SVM algorithm using Q-loss. As decribed above, we report the results for the value of C that optimizes performance on the validation set. The table also shows the number of features in each model. Note that the training and the validation set are composed of more difficult cases than the test set, which explains the generally higher Q-scores on the test set. All performance differences

Table 1. Q-score of the SVM algorithm for different alignment models.

	# Features	Training	Validation	Test
Simple	1020	26.83	27.79	39.89
Anova2	49634	42.25	35.58	44.98
Tensor	203280	52.36	34.79	42.81
Simple+Anova2	50654	42.29	35.34	44.74
Simple+Anova2+Tensor	253934	47.80	35.79	44.39
Window	447016	51.26	38.09	46.30

Table 2. Comparing training for Q-score with training for Q_4-score by test set performance.

Anova2	test Q	test Q_4
train Q	44.98	67.20
train Q_4	45.65	69.45

Window	test Q	test Q_4
train Q	46.30	68.33
train Q_4	47.65	70.71

on the test set are statistically significant according to the paired Wilcoxon test, except for the three closely related alignment models *Anova2*, *Simple+Anova2*, and *Simple+Anova2+Tensor*.

Table 1 shows that the *Simple* alignment model is too simple to fit the training data, indicated by the low Q-score on the training set. This alignment model perform considerably worse than the other alignment models. The more expressive *Anova2* model leads to substantial improvement in Q-score over *Simple* on both the valiation and test sets, showing that considering pairwise interaction between structural annotations is meaningful. The *Tensor* alignment model does worse than *Anova2*. There are signs of overfitting in the relatively high Q-score on the training set. However, the performance on the validation and test sets are respectable nonetheless. Adding the substitution costs in the alignment models together, as in *Simple+Anova2* and *Simple+Anova2+Tensor*, does not give us any gain in accuracy. Their performance on the validation and test sets are very close to *Anova2*. Only when we incorporate structural information in the local neighbourhood, as in the alignment model *Window*, do we see another jump in the Q-score on the test set. The Q-score of 46.30 in the *Window* alignment model is substantially better than the Q-score of 39.89 of the *Simple* alignment model that we started with. To provide a baseline, the Q-score of BLAST is 23.88 on the test set.

6.2 Is training to different loss functions beneficial?

The SVM method allows the use of different loss functions during training. The Q-loss used in the previous subsection is rather stringent and does not necessarily summarize the quality of an alignment well. For example, if all the aligned positions are shifted by just 1, the Q-loss will jump from 0 to 1, which is roughly the

Table 3. Comparing training for Q-score with training for Q_4-score by test set performance.

	Q on test	Q_4 on test
SVM (Window, Q_4)	47.65	70.71
SSALN	47.06	67.30
BLAST	23.88	28.44
TM-align	69.99	85.32

same Q-loss as that of a completely random alignment. Furthermore, the Q-loss does not account for the approximate nature of the training alignments, since there is typically no single exact alignment in sequence to structure alignment that is clearly correct.

Instead of Q-loss, we now consider the Q_4-loss function. Q_4-loss counts a residue as correctly aligned if the shift from its position in the reference alignment is no more than 4. The Q_4-loss function captures our intuition that small shifts in alignment could be tolerated, and such alignments should be differentiated from alignments that are completely wrong. We repeat our experiments on two alignment models from the last section, *Anova2* and *Window*, but this time we train them with Q_4 as the loss function. The results on the test set are shown in Table 2. For each table entry, we select C on the validation set with respect to the performance measure that is reported.

Table 2 shows that the models trained on Q_4 show better Q_4 performance on the test set. More surprisingly, the models trained on Q_4 also show (statistically significantly) better Q-score on the test set. This gives evidence that Q_4 can indeed effectively account for the inaccuracy of the training alignments, instead of trying to model the noise. However, in situations where the alignments have higher sequence similarity or we are more confident of the alignments, the use of Q-loss or reducing the allowable shift of of 4 in Q_4 to lower values could be beneficial. The flexibility of the SVM regarding the selection of loss function would cater either of these situations.

6.3 How does the accuracy of SVM models compare to conventional methods?

As selected by validation performance, the best alignment model is *Window* trained on Q_4. Table 3 shows the test set performance of various other methods in comparison. SSALN [3] is one of the best current alignment algorithm trained using generative methods, and it outperforms alignment and threading algorithms like CLUSTALW, GenTHREADER, and FUGUE on a variety of benchmarks. It incorporates structural information in its substitution matrices, and contains a hand-tuned gap model. SSALN was trained on exactly the same training set and same set of structural annotations as our SVM model, so a direct comparison is particularly meaningful. The SVM model substantially outperforms SSALN with respect to Q_4-score, and is slightly better than SSALN on Q-score. The performance of BLAST is included to provide a baseline. The performance of the structural alignment program TM-align [23] is reported here to show its agreement with the CE alignments, and demonstrates the rather high inherent noise in the data.

7 Discussions and Conclusions

This paper explore an SVM method for learning complex alignment models for sequence to structure alignment. We show that the algorithm can learn high-dimensional models that include many features beyond residue identity while effectively controlling overfitting. Unlike generative methods, it does not require independence assumptions between features. The SVM method provides great modeling flexibility to biologists, allowing the estimation of models that include all available information without having to worrying about statistical dependencies between features. Furthermore, we show that one can incorporate different loss functions during training, which provides the flexibility to specify the costs of different alignment errors. The empirical results show that the SVM algorithm outperforms one of the best current generative models, and is practical to train on large datasets.

8 Acknowledgments

We thank Jian Qiu for his detailed answers on our many questions on SSALN, Tamara Galor-Neah for her help with the LOOPP program, and Phil Zigoris for his work on an early prototype of the SVM software. This work is supported by NIH Grants IS10RR020889 and GM67823, and by the NSF Award IIS-0412894.

References

1. Joachims, T.: Learning to align sequences: A maximum-margin approach. http://www.joachims.org (August 2003)
2. Joachims, T., Galor, T., Elber, R.: Learning to align sequences: A maximum-margin approach. In et al., B.L., ed.: New Algorithms for Macromolecular Simulation. Volume 49 of LNCS. Springer (2005) 57–68
3. Qiu, J., Elber, R.: SSALN: an alignment algorithm using structure-dependent substitution matrices and gap penalties learned from structurally aligned protein pairs. Proteins **62** (2006) 881–91
4. Bucher, P., Hofmann, K.: A sequence similarity search algorithm based on a probabilistic interpretation of an alignment scoring system. In: International Conference on Intelligent Systems for Molecular Biology (ISMB). (1996)
5. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological Sequence Analysis. Cambridge University Press (1998)
6. Henikoff, S., Henikoff, J.G.: Amino acid substitution matrices from protein blocks. Proceedings of the National Academy of Sciences **89** (1992) 10915–10919

7. Dayhoff, M.O., Schwartz, R.M., Orcutt, B.C.: A model of evolutionary change in proteins. Atlas of Protein Sequence and Structure **5** (1978) 345–352
8. Ristad, S.E., Yianilos, P.N.: Learning string edit distance. IEEE Transactions on Pattern Recognition and Machine Intelligence **Vol. 20(5)** (1998) 522–532
9. Joachims, T.: Text categorization with support vector machines: Learning with many relevant features. In: Proceedings of the European Conference on Machine Learning, Berlin, Springer (1998) 137 – 142
10. Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y.: Large margin methods for structured and interdependent output variables. Journal of Machine Learning Research (JMLR) **6** (September 2005) 1453 – 1484
11. Lafferty, J., McCallum, A., Pereira, F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: International Conference on Machine Learning (ICML). (2001)
12. Gusfield, D., Stelling, P.: Parametric and inverse-parametric sequence alignment with XPARAL. Methods in Enzymology **266** (1996) 481–494
13. Pachter, L., Sturmfelds, B.: Parametric inference for biological sequence analysis. In: Proceedings of the National Academy of Sciences. Volume 101. (2004) 16138–16143
14. Sun, F., Fernandez-Baca, D., Yu, W.: Inverse parametric sequence alignment. In: International Computing and Combinatorics Conference (COCOON). (2002)
15. Tsochantaridis, I., Hofmann, T., Joachims, T., Altun, Y.: Support vector machine learning for interdependent and structured output spaces. In: International Conference on Machine Learning (ICML). (2004)
16. Do, C.B., Gross, S.S., Batzoglou, S.: CONTRAlign: Discriminative training for protein sequence alignment. In: International Conference in Research on Computational Molecular Biology (RECOMB). (2006)
17. McCallum, A., Bellare, K., Pereira, F.: A conditional random field for discriminatively-trained finite-state string edit distance. In: Conference on Uncertainty in Artificial Intelligence. (2005)
18. Kececioglu, J.D., Kim, E.: Simple and fast inverse alignment. In Apostolico, A., Guerra, C., Istrail, S., Pevzner, P.A., Waterman, M.S., eds.: Annual International Conference on Research in Computational Molecular Biology (RECOMB). Volume 3909 of Lecture Notes in Computer Science., Springer (2006) 441–455
19. Smith, T., Waterman, M.: Identification of common molecular subsequences. Journal of Molecular Biology **147** (1981) 195–197
20. Vapnik, V.: Statistical Learning Theory. Wiley, Chichester, GB (1998)
21. Taskar, B., Guestrin, C., Koller, D.: Maximum-margin markov networks. In: Neural Information Processing Systems (NIPS). (2003)
22. Shindyalov, I.N., Bourne, P.E.: Protein structure alignment by incremental combinatorial extension(CE) of the optimal path. Protein Eng **11** (1998) 739–747
23. Zhang, Y., Skolnick, J.: TM-align: A protein structure alignment algorithm based on TM-score. Nucleic Acids Research **33** (2005) 2302–2309
24. Adamczak, R., Porollo, A., Meller, J.: Accurate prediction of solvent accessibility using neural networks-based regression. Proteins **56** (2004) 753–67
25. Kabsch, W., Sander, C.: Dictionary of protein secondary structure: pattern recognition of hydrogen bond and geometrical features. Biopolymers **22** (1983) 2577–2637