Predicting core columns of protein multiple sequence alignments for improved parameter advising

Dan DeBlasio and John Kececioglu

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

Abstract. In a computed protein multiple sequence alignment, the *coreness* of a column is the fraction of its substitutions that are in so-called core columns of the gold-standard reference alignment of its proteins. In benchmark suites of protein reference alignments, the core columns of the reference are those that can be confidently labeled as correct, usually due to all residues in the column being sufficiently close in the spatial superposition of the folded three-dimensional structures of the proteins. When computing a protein multiple sequence alignment in practice, a reference alignment is not known, so its coreness can only be predicted.

We develop for the first time a *predictor* of column coreness for protein multiple sequence alignments. This allows us to predict which columns of a computed alignment are core, and hence better estimate the alignment's accuracy. Our approach to predicting coreness is similar to nearest-neighbor classification from machine learning, except we transform nearest-neighbor distances into a coreness prediction via a regression function, and we learn an appropriate distance function through a new optimization formulation that solves a large-scale linear programming problem. We apply our coreness predictor to *parameter advising*, the task of choosing parameter values for an aligner's scoring function to obtain a more accurate alignment of a specific set of sequences. We show that for this task, our predictor strongly outperforms other column-confidence estimators from the literature, and affords a substantial boost in alignment accuracy.

1 Introduction

The accuracy of a multiple sequence alignment computed on a benchmark set of input sequences is usually measured with respect to a *reference alignment* that represents the gold-standard alignment of the sequences. For protein sequences, reference alignments are typically determined by structural superposition of the known three-dimensional structures of the proteins in the benchmark. The accuracy of a computed alignment is then defined to be the fraction of pairs of residues aligned in the so-called *core columns* of the reference alignment that are also present in columns of the computed alignment. Core columns represent

those in the reference that are deemed to be reliable, and can be objectively defined as those columns containing a residue from every input sequence such that the pairwise distances between these residues in the structural superposition of the proteins are all within some threshold (typically a few angstroms). In short, given a known reference alignment whose columns are labeled as either core or non-core, we can determine the accuracy of any other computed alignment of its proteins by evaluating the fraction of aligned residue pairs from these core columns that are recovered. For a given column in a computed alignment, we can also define the *coreness* value of the column to be the fraction of its aligned residue pairs that are in core columns of the reference alignment. (Note that column coreness is a fully objective quantity when core columns are identified through superposition of protein structures, as in `PALI` [1] benchmarks.) A coreness value of 1 means the column of the computed alignment corresponds to a core column of the reference alignment.

When aligning sequences in practice, obviously such a reference alignment is not known, and the accuracy of a computed alignment, or the coreness of its columns, can only be estimated. A good *accuracy estimator* for computed alignments is extremely useful [7]. It can be leveraged to: pick among alternate alignments of the same sequences the one of highest estimated accuracy, for example, to choose good parameter values for an aligner's scoring function as in *parameter advising* [15]; or to select the best result from a collection of different aligners, yielding a natural *ensemble aligner* that can be far more accurate than any individual aligner in the collection [5].

Similarly, a good *coreness predictor* for columns in a computed alignment can be used to: mask out unreliable regions of the alignment before computing an evolutionary tree; or to improve an alignment accuracy estimator by concentrating its evaluation function on columns of higher predicted coreness, thereby boosting the performance of parameter advising. In fact, in principle a perfect coreness predictor would itself yield an ideal accuracy estimator.

In this paper, we develop for the first time a column coreness predictor for protein multiple sequence alignments. Our approach to predicting coreness is similar in some respects to nearest-neighbor classification from machine learning, except we transform nearest-neighbor distance into a coreness prediction via a regression function, and we learn an appropriate distance function through a new optimization formulation that solves a large-scale linear programming problem. We evaluate the performance of our new coreness predictor by applying it to the task of parameter advising in multiple sequence alignment.

Related work To our knowledge, this is the first fully general attempt to directly predict the coreness of columns in computed protein alignments. Tools are available that assess the quality of columns in an alignment, and can be categorized into: (a) those that only identify columns as unreliable, for removal from further analysis; and (b) those that compute a column quality score, which can be thresholded. Tools that simply mask unreliable columns include `GBLOCKS` [3], `TrimAL` [2], and `ALISCORE` [16]. Popular quality-score tools are `Noisy` [8], `ZORRO` [21], `TCS` [4], and `GUIDANCE` [17].

Our experiments compare our coreness predictor to TCS and ZORRO: the most recent tools that provide quality scores, as opposed to masking columns. GUIDANCE requires four or more sequences, which excludes many benchmarks. Noisy is dominated by an earlier version of GUIDANCE, which along with ALISCORE and GBLOCKS are in turn dominated by ZORRO.

Plan of the paper Section 2 next describes how we learn our coreness predictor. Section 3 then explains how we use predicted coreness to improve accuracy estimation for protein alignments. Section 4 evaluates our approach to coreness prediction by applying the improved accuracy estimator to alignment parameter advising. Section 5 concludes.

2 Learning a coreness predictor

To describe how we learn a column coreness predictor, we first discuss our *representation* of alignment columns, and our grouping of consecutive columns into *window classes*. We then present our *regression function* for predicting coreness, which transforms the nearest-neighbor distance from a window to a class into a coreness value. Finally, we describe how we learn this window distance function by solving a large-scale *linear programming* problem.

2.1 Representing alignment columns

The information used by our coreness predictor, beyond the multiple sequence alignment itself, is an annotation of its protein sequences by predicted secondary structure (which can be obtained in a preprocessing step by running the sequences through a standard protein secondary structure prediction tool such as PSIPRED [12]). When inputting a column from such an annotated alignment to our coreness predictor, we need a column representation that, while capturing the association of amino acids and predicted secondary structure types, is also independent of the number of sequences in the column. This is necessary as our predictor will be trained on example alignments of particular sizes, yet the resulting predictor must apply to alignments with arbitrary numbers of sequences.

Let Σ be the 20-letter amino acid alphabet, and $\Gamma = \{\alpha, \beta, \gamma\}$ be the secondary structure alphabet, corresponding respectively to types α-*helix*, β-*strand*, and *other* (also called *coil*). We encode the association of an amino acid $c \in \Sigma$ with its predicted secondary structure type $s \in \Gamma$ by an ordered pair (c, s) that we call a *state*, from the set $Q = (\Sigma \times \Gamma) \cup \{\xi\}$. Here $\xi = (\texttt{-}, \texttt{-})$ is the *gap state*, where the dash symbol '-' $\notin \Sigma$ is the alignment *gap character*.

We represent a multiple alignment column as a distribution over the set of states Q, which we call its *profile* (mirroring standard terminology [9, p. 101]). We denote the profile C for a given column by a function $C(q)$ on states $q \in Q$ satisfying $C(q) \geq 0$ and $\sum_{q \in Q} C(q) = 1$. Most secondary structure prediction tools output a confidence value (not a true probability) that an amino acid in a protein sequence has a given secondary structure type. For a column of amino

acids $(c_1 \cdots c_k)$ in a multiple alignment of k sequences, denote the *confidence* that amino acid c_i has structure type $s \in \Gamma$ by $p_i(s) \geq 0$, where $\sum_{s \in \Gamma} p_i(s) = 1$. For non-gap state $q = (a, s) \neq \xi$, profile C has value $C(q) := \sum_{i \,:\, c_i = a} p_i(s) \,/\, k$. In other words, $C(q)$ is the normalized total confidence across the column in state $q \neq \xi$. For gap state $q = \xi$, the profile value is $C(\xi) := \big|\{i \,:\, c_i = \text{`-'}\}\big| \,/\, k$, the relative frequency of gap characters in the column.

2.2 Classes of column windows

In protein benchmarks, a column of a reference alignment is labeled core if its residues are all sufficiently close in the superposition of the proteins' three-dimensional structures. The folded structure around a residue is a function of nearby residues in the protein. Consequently, to predict the coreness of a column in a computed alignment, we need contextual information from nearby columns. We gather this context for a column by forming a window of consecutive columns centered on it. Formally, a *window* W of width $w \geq 1$ is a sequence of $2w{+}1$ consecutive column profiles $C_{-w} \cdots C_{-1} C_0 C_{+1} \cdots C_{+w}$ centered around profile C_0.

We define the following set of *window classes* $\mathcal{C}$, depending on whether the columns in a labeled training window are known to be core or non-core in the reference alignment. We denote a column labeled core by `C`, and a column labeled non-core by `N`. For window width $w = 1$ (which has three consecutive columns), such labeled windows correspond to strings of length 3 over alphabet $\{\texttt{C}, \texttt{N}\}$. The three classes of *core windows* are `CCC`, `CCN`, `NCC`; the three classes of *non-core windows* are `CNN`, `NNC`, `NNN`. (A window is considered core or non-core depending on the label of its center column. We exclude windows `NCN` and `CNC`, as these almost never occur in reference alignments.) Together these six classes comprise set $\mathcal{C}$. We call the five classes with at least one core column `C` in the window, *structured classes*; the one class with no core columns is the *unstructured class*, denoted by $\perp = \texttt{NNN}$.

2.3 The coreness regression function

We learn a coreness predictor by fitting a regression function that measures the similarity between a column's window and training examples of windows with known coreness, and transforms this similarity into a coreness value.

The similarity of windows $V = V_{-w} \cdots V_w$ and $W = W_{-w} \cdots W_w$ is expressed in terms of the similarity of their corresponding column profiles V_i and W_i. We measure the dissimilarity of two such profiles from window class c at position i, using class- and position-specific *substitution scores* $\sigma_{c,i}(p, q)$ on pairs of states p, q. (Section 2.4 describes how we learn these scores.) Given substitution scores $\sigma_{c,i}$, the *distance* between windows V and W from class $c \in \mathcal{C} - \{\perp\}$ is,

$$d_c(V, W) \;:=\; \sum_{-w \,\leq\, i \,\leq\, +w} \;\sum_{p, q \,\in\, Q} V_i(p)\; W_i(q)\; \sigma_{c,i}(p, q)\,.$$

These positional $\sigma_{c,i}$ allow distance function d_c to score dissimilarity higher at positions i near the center of the window, and lower towards its edges.

The *regression function* that predicts the coreness of a column first forms a window W centered on the column, and then performs the following.

(1) (*Find distance to closest class*) Across all labeled training windows, in all structured window classes, find the training window that has smallest class-specific distance to W. Call this closest window V, its class c, and their distance $\delta = d_c(V, W)$.

(2) (*Transform distance to coreness*) If class c is a core class, return the coreness value given by transform function $f_{\text{core}}(\delta)$. Otherwise, return value $f_{\text{non}}(\delta)$.

To transform the *nearest-neighbor distance* δ from Step (1) into a coreness value in Step (2), we use *logistic functions* for f_{core} and f_{non}. We fit these logistic curves to empirically-measured average-coreness values at nearest-neighbor distances collected for either core or non-core training examples, using the curve fitting tools in `SciPy` [13]. As Figure 1 in Section 4.1 later shows, these logistic transform functions fit actual coreness data remarkably well.

2.4 Learning the distance function by linear programming

We now describe the linear program used to learn the distance function on column windows. The linear program learns a *class-specific* distance function d_c for each window class $c \in \mathcal{C}$.

To construct the linear program, we partition the *training set* $\mathcal{T}$ of labeled windows by window class: subset $T_c \subseteq \mathcal{T}$ contains all training windows of class $c \in \mathcal{C}$. We then form a smaller *training sample* $S_c \subseteq T_c$ for each class c by choosing a random subset of T_c with a specified cardinality $|S_c|$.

The *constraints* of the linear program fall in several categories. For a sample training window $W \in S_c$, we identify other windows $V \in T_c$ from the same class c in the full training set that are close to W (under a default distance $\widetilde{d}_c$). We call these close windows V from the same class c, *targets*. Similarly for $W \in S_c$, we identify other windows $U \in T_b$ from a different class $b \neq c$ in the full training set that are also close to W (under $\widetilde{d}_b$). We call these other close windows U from a different class b, *impostors*. More formally, the *neighborhood* $\mathcal{N}_c(W, i)$ for a structured class $c \in \mathcal{C} - \{\perp\}$ denotes the set of i-nearest-neighbors to W (not including W) from training set T_c under the class-specific *default distance* function $\widetilde{d}_c$. (The default distance function that we use in our experiments is described in Section 4.1.) The constraints of the linear program find distance functions that for a sample window $W \in S_c$, *pull in* targets $V \in \mathcal{N}_c(W, i)$ by making $d_c(V, W)$ small, and *push away* impostors $U \in \mathcal{N}_b(W, i)$ for $b \neq c$ by making $d_b(U, W)$ large.

The *target constraints* for each sample window $W \in S_c$ from each structured class $c \in \mathcal{C} - \{\perp\}$, and each target window $V \in \mathcal{N}_c(W, k)$, are,

$$e_{VW} \;\geq\; d_c(V, W) \;-\; \tau\,, \tag{1}$$

$$e_{VW} \;\geq\; 0\,, \tag{2}$$

where e_{VW} is a target *error variable* and τ is a *threshold variable*. In the above, quantity $d_c(V,W)$ is a linear expression in the *substitution score variables* $\sigma_{c,i}(p,q)$, so constraint (1) is a linear inequality in the variables. Intuitively, we would like condition $d_c(V,W) \leq \tau$ to hold (so W will be considered to be in its correct class c); in the solution, variable e_{VW} will equal $\max\{d_c(V,W)-\tau,\, 0\}$, the amount of error by which this ideal condition is violated.

The *impostor constraints* for each sample window $W \in S_c$ from each structured class $c \in \mathcal{C} - \{\perp\}$, and each impostor window $V \in \mathcal{N}_b(W,\ell)$ from each structured class $b \in \mathcal{C} - \{\perp\}$ with $b \neq c$, are,

$$f_W \;\geq\; \tau \;-\; d_b(V,W) \;+\; 1\,, \tag{3}$$

$$f_W \;\geq\; 0\,, \tag{4}$$

where f_W is an impostor error variable. Intuitively, we would like condition $d_b(V,W) > \tau$ to hold (so W will not be considered to be in the incorrect class b), which we can express by $d_b(V,W) \geq \tau + 1$ using a *margin* of 1. (Since the scale of the distance functions is arbitrary, we can always pick a unit margin without loss of generality.) In the solution to the linear program, variable f_W will equal $\max_{b \in \mathcal{C}-\{\perp\},\, V \in \mathcal{N}_b(W,\ell)}\{\tau - d_b(V,W) + 1,\, 0\}$, the largest amount of error by which this condition is violated for W across all b and V.

We also have impostor constraints for each completely non-core window $W \in T_\perp$, and each core window $V \in \mathcal{N}_b(W,\ell)$ from each structured core class b (as we do not want W to be considered core), which are of the same form as inequalities (3) and (4) above.

The *triangle inequality constraints*, for each structured class $c \in \mathcal{C} - \{\perp\}$, each window position $-w \leq i \leq w$, and all states $p,q,r \in Q$ (including the gap state ξ), are: $\sigma_{c,i}(p,r) \;\leq\; \sigma_{c,i}(p,q) \;+\; \sigma_{c,i}(q,r)$. A consequence of these constraints is that the resulting distance functions d_c also satisfy the triangle-inequality property. (We omit the proof due to page limits.) This property allows us to use faster metric-space data structures for computing the nearest-neighbor distance δ from Section 2.3.

The remaining constraints, for classes c, positions i, and states p and q, are: $\sigma_{c,i}(p,q) = \sigma_{c,i}(q,p)$, $\sigma_{c,i}(p,p) \leq \sigma_{c,i}(p,q)$, $\sigma_{c,i}(p,q) \geq 0$, $\sigma_{c,i}(\xi,\xi) = 0$, and $\tau \geq 0$, which ensure the distance functions are symmetric and non-negative.

Finally, the *objective function* minimizes the average error over all training sample windows. Formally, we minimize,

$$\alpha\, \tfrac{1}{|\mathcal{C}|-1} \sum_{c \in \mathcal{C}-\{\perp\}} \tfrac{1}{|S_c|} \sum_{W \in S_c} \tfrac{1}{k} \sum_{V \in \mathcal{N}_c(W,k)} e_{VW} \;+\; (1-\alpha)\, \tfrac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \tfrac{1}{|S_c|} \sum_{W \in S_c} f_W\,,$$

where $0 \leq \alpha \leq 1$ is a blend parameter controlling the weight on target error versus impostor error. We note that in an optimal solution to this linear program, variables $e_{VW} = \max\{d_c(V,W)-\tau, 0\}$ and $f_W = \max_{V,b}\{\tau - d_b(V,W)+1, 0\}$, since inequalities (1)–(4) ensure the error variables are at least these values, while minimizing the above objective function ensures they will not exceed them. Thus solving the linear program finds distance functions d_c, given by substitution

scores $\sigma_{c,i}(p, q)$, that minimize the average over the training windows $W \in S_c$ of the amount of violation of our ideal conditions $d_c(V, W) \leq \tau$ for targets $V \in T_c$ and $d_b(V, W) > \tau$ for impostors $V \in T_b$.

3 Applying coreness to accuracy estimation

The `Facet` alignment accuracy estimator [15] is a linear combination of efficiently-computable feature functions that are positively correlated with true accuracy. As mentioned earlier, the accuracy of a computed alignment is measured only with respect to core columns of the reference alignment. We leverage our coreness predictor to improve the `Facet` estimator by: (1) creating a new feature function that attempts to directly estimate accuracy, and (2) concentrating the evaluation of existing feature functions on columns with high predicted coreness.

3.1 Creating a new coreness feature

Our new feature function on alignments, *Predicted Alignment Coreness*, is similar to the so-called total-column score sometimes used to measure alignment accuracy. Predicted Alignment Coreness counts the number of columns in the alignment that are predicted to be core, by taking a window W around each column, and determining whether its *predicted coreness* $\chi(W)$ exceeds a threshold κ. This count of predicted core columns is normalized by an estimate of the number of core columns in the unknown reference alignment of the sequences. Formally, for computed alignment $\mathcal{A}$ of sequences $\mathcal{S}$, the Predicted Alignment Coreness feature function is $F_{\texttt{AC}}(\mathcal{A}) := \left|\{W \in \mathcal{A} : \chi(W) \geq \kappa\}\right| / L(\mathcal{S})$.

Normalizer $L(\mathcal{S})$ is designed to be positively correlated with the number of core columns in the reference alignment for sequences $\mathcal{S}$. We consider functions L that are linear combinations of products of at most three factors from the following: aggregate measures of the lengths of sequences in $\mathcal{S}$ (their minimum, mean, and maximum length); ratios of the longest-common-subsequence length for pairs of sequences, divided by an aggregate length measure (a form of "percent identity"); and ratios of the difference in maximum and minimum length, divided by an aggregate length measure (a form of "percent indel"). Finally, we obtain $L(\mathcal{S})$ by solving a linear program to find coefficients of the linear combination that minimizes the L_1-norm with the true number of core columns.

3.2 Augmenting former features by coreness

We also augment some of the features currently in `Facet` to concentrate their evaluation on columns of higher predicted coreness. A full description of all feature functions in `Facet` is in [15]; we use predicted coreness to augment: Secondary Structure Blockiness, Secondary Structure Identity, Amino Acid Identity, and Average Substitution Score. Each of these functions computes a feature value that in essence is a sum over substitutions in a column; in the modified feature, this is now a *weighted* sum over columns weighted by predicted coreness.

4 Assessing the coreness predictor

We evaluate our new approach to coreness prediction, and its use in accuracy estimation for alignment parameter advising, through experiments on a collection of protein multiple sequence alignment benchmarks. A full description of the benchmarks, and the universe of parameter choices for parameter advising, is in [15]. Briefly, the benchmarks in our experiments consist of reference alignments of protein sequences largely induced by structurally aligning their known three-dimensional structures. We use the `BENCH` suite of Edgar [10], supplemented by a selection from the `PALI` suite of Balaji et al. [1]. Our full benchmark collection consists of 861 reference alignments.

We use twelve-fold *cross-validation* to assess both column classification with our coreness predictor, and parameter advising with our augmented accuracy estimator. To correct for the overabundance of easy-to-align benchmarks when assessing parameter advising, we bin the benchmarks according to *difficulty*, measured by the true accuracy of their alignment computed by the `Opal` aligner [19, 20] under its default parameter setting. We ensure folds are balanced in their representation of benchmarks from all difficulty bins. For each fold, we generate a *training set* and *testing set* of example alignments by running `Opal` on each benchmark for each parameter choice from a fixed universe of settings.

4.1 Constructing the coreness predictor

We first discuss learning distance functions and fitting transform functions.

Learning the distance function To keep the linear program manageable, each training fold and each structured class has a touchstone that is a sample of 4,000 window examples. When learning the distance functions and testing the accuracy of our coreness predictor, we use training and testing samples of 2,000 window examples representing all classes (including the unstructured class), drawn from our training and testing example alignments.

We form the initial sets of targets and imposters for the linear program by either: (1) using a default distance function whose positional substitution score is a convex combination of (a) the `VTML200` substitution score on the states' amino acids (transformed to a dissimilarity value in the range $[0, 1]$) and (b) the identity of the states' secondary structure types, with positions weighted so the center column has twice the weight of its flanking columns; or (2) randomly sampling example windows from the appropriate classes to form targets and impostors.

When learning the distance function we use 2 targets and 150 impostors per class for each window in the training sample. Once a distance function is learned, we *iterate* the process by using the learned distance function to recompute the sets of targets and impostors for another instance of the linear program that is in turn solved to learn a new distance function. For the receiver operating characteristic (ROC) curve, we give the *area under the curve* (AUC) measure on both training and testing data, for successive iterations of distance learning,

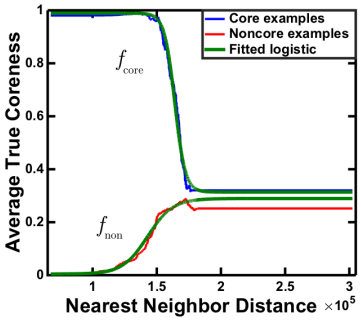

Fig. 1. Fit of the logistic transform functions to the average true coreness of training examples.

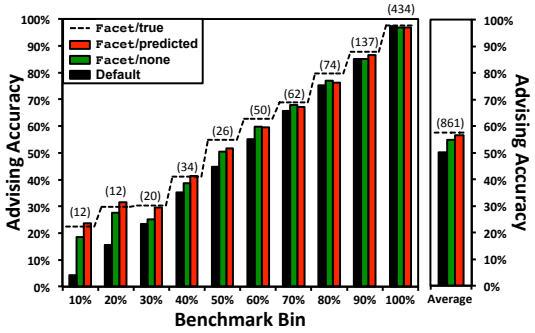

Fig. 2. Average parameter advising accuracy within difficulty bins for greedy advisor sets of cardinality 7.

starting from the default distance function. Across the first 5 iterations, the successive *training* AUC is $86.3, 93.9, 98.9, 99.3, 99.3$; the corresponding *testing* AUC is $83.8, 82.5, 84.9, 84.8, 85.0$. Note that the training AUC increases steadily for the first four iterations, though this translates into only a slight improvement in testing AUC; after this fifth iteration, no further improvement is seen. While iterating distance learning markedly improves our core column predictor on the training examples, it is overfitting and does not generalize well to testing examples, most likely due to the smaller training sample and touchstone we used to reduce the time for solving the linear program. We also found that using *random examples* for the targets and imposters led to much better generalization, namely a training and testing AUC of 85.8 and 88.7, so we use these distances (without iterating) when evaluating results on parameter advising.

Transforming distance to coreness Figure 1 shows the fitted logistic functions f_{core} and f_{non} used to transform nearest-neighbor distance to predicted coreness, superimposed on the underlying true coreness data for one fold of training examples. The horizontal axis is nearest-neighbor distance δ, while the vertical axis is the average true coreness of training examples at that distance. The blue and red curves respectively show the average true coreness of training examples for which the nearest neighbor is in either a core class or a structured non-core class. The top and bottom green curves respectively show the logistic transform functions for the core and non-core classes fitted to this training data. Note that the green logistic curves fit the data quite well.

4.2 Improving parameter advising

A *parameter advisor* and has two components: (1) an *accuracy estimator*, which estimates the accuracy of a computed alignment, and (2) an *advisor set*, which is a set of candidate assignments of values to the aligner's parameters. The advisor picks the choice of parameter values from the advisor set for which the aligner yields the alignment of highest estimated accuracy. In our experiments,

we evaluate the true accuracy of the Opal aligner [19, 20] combined with a parameter advisor using Facet (the best accuracy estimator for advising from the literature [15]), augmented by our new coreness predictor as well as by TCS and ZORRO. We compare these parameter advising results to previous results using unmodified Facet as well TCS (the next-best accuracy estimator for advising). We also compare against augmenting Facet by *true* coreness, which provides a limit for an unattainable perfect coreness predictor.

The choice of advisor set is crucial for parameter advising, as the performance of an advisor is limited by the quality of the alignments generated by this set of parameter choices. We consider two types of advisor sets [6]: estimator-independent *oracle sets*, which are optimal for a conceptual oracle advisor that uses true accuracy as its estimator; and estimator-aware *greedy sets*, which tend to perform better than oracle sets in practice, but are tuned to a specific accuracy estimator. Finding such advisor sets requires specifying a universe of possible parameter choices; we use the universe of 243 parameter choices from [6].

As mentioned earlier, we bin alignments according to difficulty to correct for the overabundance of easy-to-align benchmarks. Figure 2 lists in parentheses above the bars the number of benchmarks in each bin. When reporting advising accuracy, we give the true accuracy of the alignments chosen by advisor, uniformly averaged over *bins* (rather than uniformly averaging over benchmarks). With this equal weighting of bins, an advisor that uses only the single optimal default parameter choice will achieve an average advising accuracy of roughly 50% (illustrated in Figure 3). This establishes, as a point of reference, advising accuracy 50% as the *baseline* against which to compare advising performance.

The augmented Facet estimator We use our coreness predictor to modify the Facet accuracy estimator by including the new Predicted Alignment Coreness feature of Section 3.1, and augmenting existing feature functions by coreness as in Section 3.2. We learned coefficients for these feature functions using the difference-fitting technique described in [15]. The new alignment accuracy estimator that uses our coreness predictor has non-zero coefficients for: the new feature, Predicted Alignment Coreness $F_{\mathtt{AC}}$; two features augmented by predicted coreness, Amino Acid Identity $F'_{\mathtt{AI}}$, and Secondary Structure Identity $F'_{\mathtt{SI}}$; and four original unaugmented features, Gap Open Density $F_{\mathtt{GO}}$, Secondary Structure Agreement $F_{\mathtt{SA}}$, Amino Acid Identity $F_{\mathtt{AI}}$, and Secondary Structure Blockiness $F_{\mathtt{BL}}$. The resulting augmented accuracy estimator is: $(0.512)\,F_{\mathtt{GO}} + (0.304)\,F'_{\mathtt{SI}} + (0.157)\,F_{\mathtt{SA}} + (0.109)\,F_{\mathtt{AI}} + (0.096)\,F_{\mathtt{BL}} + (0.025)\,F'_{\mathtt{AI}} + (0.013)\,F_{\mathtt{AC}}$.

Improvement in advising accuracy We assess the parameter advising performance of our augmented Facet estimator ("Facet/predicted") by comparing it to unaugmented Facet ("Facet/none"), as well as Facet augmented by TCS ("Facet/TCS"), ZORRO ("Facet/ZORRO"), and true coreness ("Facet/true"). We also compare against TCS, the next-best estimator from the literature.

Parameter advising performance using oracle and greedy advisor sets is shown in Figures 3 and 4. In both figures, the horizontal axis is advisor set cardinality, while the vertical axis is advising accuracy for testing folds, averaged across

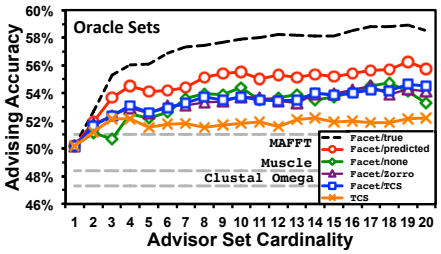

Fig. 3. Advising accuracy on *oracle sets* with modified `Facet` or `TCS` estimators.

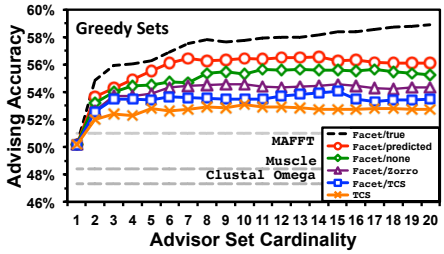

Fig. 4. Advising accuracy on *greedy sets* with modified `Facet` or `TCS` estimators.

bins. The curves show performance with the `Opal` aligner [19, 20]. For reference, the default alignment accuracy for three other popular aligners, `MAFFT` [14], `MUSCLE` [11], and `Clustal Omega` [18], is also shown with dashed horizontal lines.

Figure 3 shows that on *oracle* advisor sets, `Facet`/predicted compared to `Facet`/none boosts the average accuracy of parameter advising by nearly 3%. This increase is in addition to the improvement of `Facet` over `TCS`.

Figure 4 shows that on *greedy* advisor sets, `Facet`/predicted at cardinality 7 boosts advising accuracy by 2%. (Note the curves are higher for greedy sets than oracle sets.) The accuracy for `Facet`/predicted is about halfway between `Facet`/none and `Facet`/true (the perfect predictor). Interestingly, `Facet`/`TCS` and `Facet`/`ZORRO` actually have worse accuracy than `Facet`/none.

Advising accuracy within difficulty *bins* for greedy sets of cardinality 7 is shown earlier in Figure 2. In this bar chart, for the bin at each difficulty on the horizontal axis, advising accuracy averaged over just the benchmarks in the bin is shown on the vertical axis. The final chart on the right gives accuracy averaged across all bins. Note that on the most difficult benchmarks, `Facet`/predicted boosts accuracy over `Facet`/none by more than 5%.

For reference, advising accuracy uniformly-averaged over *benchmarks* (rather than bins), on greedy sets of cardinality 10, is: for `Facet`/none, 81.9%; and for `Facet`/predicted, 82.1%. On these same benchmarks, the corresponding average accuracy of other popular aligners, using their default parameter settings, is: `Clustal Omega`, 77.3%; `MUSCLE`, 78.1%; `MAFFT`, 79.4%; and `Opal`, 80.5%.

5 Conclusion

We have developed a column coreness predictor for protein multiple sequence alignments that uses a regression function on nearest neighbor distances for class distance functions learned by solving a new linear programming formulation. When applied to alignment accuracy estimation and parameter advising, the coreness predictor strongly outperforms other column confidence estimators from the literature, and provides a substantial boost in advising accuracy.

Acknowledgement This research was supported by NSF grant IIS-1217886.

References

1. Balaji, S., Sujatha, S., Kumar, S., Srinivasan, N.: PALI—a database of Phylogeny and ALIgnment of homologous protein structures. NAR 29(1), 61–65 (2001)
2. Capella-Gutierrez, S., Silla-Martinez, J.M., Gabaldón, T.: trimAl: a tool for automated alignment trimming in large-scale phylogenetic analyses. Bioinformatics 25(15), 1972–1973 (Aug 2009)
3. Castresana, J.: Selection of conserved blocks from multiple alignments for their use in phylogenetic analysis. Molecular Biol. and Evolution 17(4), 540–552 (May 2000)
4. Chang, J.M., Tommaso, P.D., Notredame, C.: TCS: A new multiple sequence alignment reliability measure to estimate alignment accuracy and improve phylogenetic tree reconstruction. Molecular Biology and Evolution 31, 1625–1637 (2014)
5. DeBlasio, D., Kececioglu, J.: Ensemble multiple sequence alignment via advising. Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB) pp. 452–461 (2015)
6. DeBlasio, D.F., Kececioglu, J.D.: Learning parameter sets for alignment advising. Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (BCB) pp. 230–239 (2014)
7. DeBlasio, D.F., Wheeler, T.J., Kececioglu, J.D.: Estimating the accuracy of multiple alignments and its use in parameter advising. Proc. of the 16th *Conference on Research in Computational Molecular Biology* (RECOMB) pp. 45–59 (2012)
8. Dress, A.W., Flamm, C., Fritzsch, G., Grünewald, S., Kruspe, M., Prohaska, S.J., Stadler, P.F.: Noisy: Identification of problematic columns in multiple sequence alignments. Algorithms for Molecular Biology 3(7) (2008)
9. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: Biological Sequence Analysis: Probablistic Models of Proteins and Nucleic Acids. Cambridge Univ. Press (1998)
10. Edgar, R.C.: BENCH. drive5.com/bench (Jan 2009)
11. Edgar, R.C.: MUSCLE: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics 5(113), 1–19 (2004)
12. Jones, D.T.: Protein secondary structure prediction based on position-specific scoring matrices. Journal of Molecular Biology 292(2), 195–202 (Sep 1999)
13. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python. http://www.scipy.org (2001), http://www.scipy.org/
14. Katoh, K., Kuma, K.i., Toh, H., Miyata, T.: MAFFT ver. 5: improvement in accuracy of multiple sequence alignment. Nuc. Acids Res. 33(2), 511–518 (Jan 2005)
15. Kececioglu, J., DeBlasio, D.: Accuracy estimation and parameter advising for protein multiple sequence alignment. Jour. of Comput. Bio. 20(4), 259–279 (Apr 2013)
16. Kück, P., Meusemann, K., Dambach, J., et al.: Parametric and non-parametric masking of randomness in sequence alignments can be improved and leads to better resolved trees. Frontiers in Zoology 7(10), 1–10 (2010)
17. Sela, I., Ashkenazy, H., Katoh, K., Pupko, T.: GUIDANCE2: accurate detection of unreliable alignment regions accounting for the uncertainty of multiple parameters. Nucleic Acids Research 43(W1), W7–W14 (2015)
18. Sievers, F., et al.: Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. Molec. Sys. Bio. 7(1), 539–539 (Jan 2011)
19. Wheeler, T.J., Kececioglu, J.D.: Multiple alignment by aligning alignments. Bioinformatics 23(13), i559–i568 (Jul 2007), proceedings of ISMB 2007
20. Wheeler, T.J., Kececioglu, J.D.: Opal: software for sum-of-pairs multiple sequence alignment. opal.cs.arizona.edu (Jan 2012)
21. Wu, M., Chatterji, S., Eisen, J.A.: Accounting for alignment uncertainty in phylogenomics. PLoS ONE 7(1), e30288 (2012)