Evolved Matrix Operations for Post-Processing Protein Secondary Structure Predictions

Varun Aggarwal and Robert M. MacCallum
Stockholm Bioinformatics Center, Stockholm University, 106 91 Stockholm, Sweden
{varun,maccallr}@sbc.su.se

January 2004

Abstract

Predicting the three-dimensional structure of proteins is a hard problem, so many have opted instead to predict the secondary structural state (usually helix, strand or coil) of each amino acid residue. This should be an easier task, but it now seems that a ceiling of around 76% per-residue three-state accuracy has been reached. Further improvements will require the correct processing of so-called "long-range information". We present a novel application of genetic programming to evolve high-level matrix operations to post-process secondary structure prediction probabilities produced by the popular, state-of-the-art neural network-based PSIPRED by David Jones. We show that global and long-range information may be used to increase three-state accuracy by at least 0.26 percentage points – a small but statistically significant difference. This is on top of the 0.14 percentage point increase already made by PSIPRED's built-in filters.

1 Introduction

Proteins are believed to fold to their native states through a process driven by both local sequence preferences and the consequences of long-range interactions which form along the way, for example during β-sheet formation. Currently, secondary structure prediction (SSP) techniques predict, at best, about 76% of residues correctly into one of three states: helix, strand or coil. This quite respectable level of performance can be seen as a reflection of the importance of local sequence information in the formation of structure – the predictors get optimal results when fed information from a sequence window of ± 7 residues only. However it is generally agreed that substantial advances in SSP will require long-range information to be incorporated effectively. In the last few years, a number of efforts have been made in this direction[12, for a review], but it seems difficult to break the 76% ceiling. Very recently however, Meiler and Baker[9] extracted long-range information from predicted three-dimensional structures, and fed this back into their secondary structure predictor. For proteins smaller than 150 residues, they report a substantial improvement in SSP accuracy (around 4-5%), although the computational demands are very high (many hours) for each prediction.

Here we explore a more modest approach in which we post-process (and hopefully improve) existing secondary structure predictions from the program PSIPRED[4] with the aid of long-range information. While connectionist machine learning techniques, like the neural networks (NNs) used in PSIPRED, have been shown to be the most effective way to map local sequence information into secondary structural state, we propose that a more rule-based approach may be best for the purpose of post-processing. For example, nearly all proteins

obey this simple rule: they have either zero or ≥ 2 strands (because unpaired strands cannot exist in isolation). For input data, we may choose between using symbols or continuous variables. We chose the latter, because it is easy to take a $3 \times N$ matrix of helix, strand and coil "probabilities" (for each protein of N residues) directly from PSIPRED. See the left side of Figure 1 for an overview of PSIPRED. Our aim is to produce a set of rules/operations/filters that transform these matrices so that they produce better predictions. Genetic programming[7, for an introduction] (GP) is a convenient tool for this task, since it can produce solutions of varying size and complexity, and we do not have to make many assumptions about the nature of the solution.

Although the idea of evolved matrix operations was introduced to GP some time ago by Montana[10], we have found only one practical application in the literature[6]. Although high-level data manipulation libraries and languages, such as MATLAB, are designed to make life easier for human programmers and scientists, they should also be useful in GP. In this work we claim to be the first to integrate such a data language (namely PDL) seamlessly into GP. We should also point out that another GP system, GPLAB[13], also has the potential to work on matrix data types using MATLAB operators, although as currently provided it handles only scalars.

The following section describes the datasets used to benchmark our post-processors and our novel approach to matrix-manipulating GP. In Section 3 we present some necessary preliminary results concerning discrete filtering of secondary structure predictions. We then investigate the "added value" of long-range information by performing local and global averaging on the inputs for GP runs which evolve matrix transformations. We show that this averaging leads to an improvement in SSP accuracy of around 0.22 percentage points (which is significant when compared to no averaging). The results also suggest an upper limit of about 150 residues to the extent of long-range interactions. Finally, by combining multiple post-processors we can make improvements of at least 0.26 percentage points compared to standard PSIPRED on our datasets.

2 Methods and Data

2.1 Training Data

When benchmarking structure prediction methods it is of critical importance to ensure that the methods have not become good at predicting just one particular group of proteins (unless membership of that group can also be predicted confidently). In supervised learning there is the further complication that performance can be overestimated if test-set proteins are related in some way to proteins used for training. We avoid these problems firstly by taking proteins from an ASTRAL[2] subset of protein domain sequences of known structure from SCOP[11] release 1.55. Within this subset, no pair of sequences share more than 10% identical amino acids after being aligned with each other. This is a stringent cutoff which counters the over-representation problem, but we go further and ensure that our training and testing sets do not contain more than 10 members of the same SCOP superfamily (these are believed to have common ancestry). Finally, and most importantly, we ensure that no members of the same SCOP superfamily are present in both training and testing sets. The result is a training set containing 911 domains and a test set with 477 domains. The data is available on request from the authors. Unless otherwise stated, each run uses a unique training set of 500 randomly selected domains (from the original 911 training domains), and performance figures are calculated for the entire test set (for individuals selected on the basis of training performance).

2.2 Baseline PSIPRED Predictions

We used PSIPRED[4] version 2.3 with default parameters to generate our baseline sec-

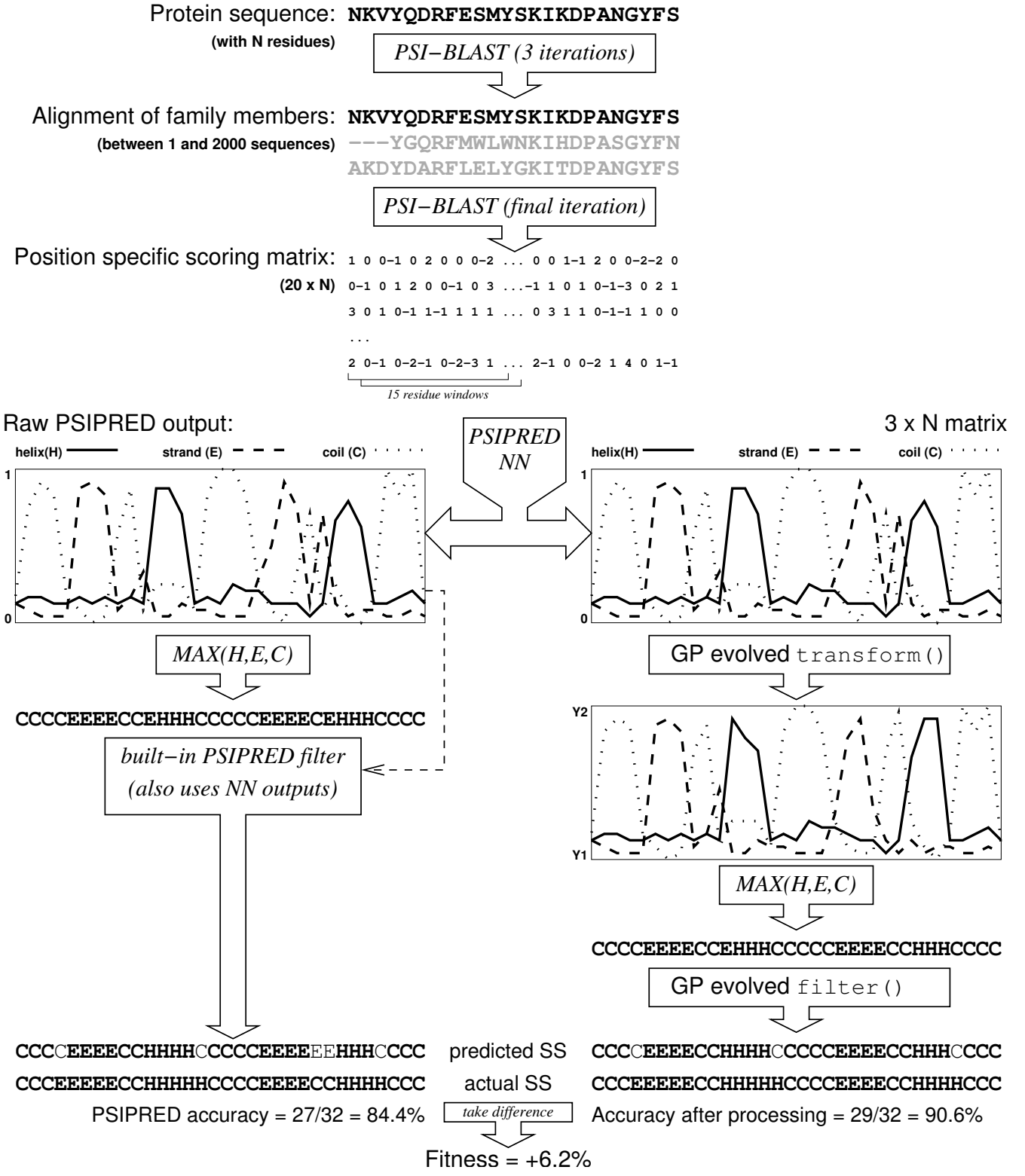


Figure 1: Fictional example showing the flow of data through the standard PSIPRED approach (left side) and through our evolved matrix `transform()` and symbolic `filter()` functions (right side). *MAX(H,E,C)* represents the simple function which converts a $3 \times N$ matrix of real values for helix, strand and coil into a string of `H`, `E` and `C` characters respectively according to the maximum value for each residue.

ondary structure predictions. PSIPRED uses PSI-BLAST[1] to find similar sequences in a sequence database (we used the set of non-redundant sequences called "nr", downloaded from the NCBI on 22 July 2002). PSIPRED takes the positional amino acid frequency matrix generated by PSI-BLAST as input, and outputs the prediction in horizontal and vertical format. We need the vertical format file (".ss2" suffix) because it contains the raw NN outputs for each secondary structural state. The vertical format file also contains the "final" three-state prediction which does not correspond exactly to the maximal network output, but is instead the result of applying PSIPRED's filtering rules (discussed in Section 3.1) to the network outputs.

Three-state SSP accuracy is most often reported as the fraction of correctly predicted residues in the entire dataset, and is denoted $Q3$. We use the DSSP program[5] in the same way as Jones[4] to objectively define the "correct" secondary structure from 3D coordinates. The baseline PSIPRED $Q3$ values for the training and test sets used here are 79.89% and 78.69%. It is not surprising that these $Q3$ values are higher than the widely reported 76%, because our training and test sets are likely to include the same or similar proteins that were used to train the PSIPRED NNs. This unavoidable overtraining should not affect our results, because it is consistent across all our data.

2.3 GP Implementation

We have used the open-source PerlGP system[8] in this study. It has a tree representation, is strongly typed, and evolves programs which follow a user-defined grammar. Because the evolved code is evaluated by the Perl interpreter, we can include calls to the powerful Perl Data Language[14] (PDL) with no extra work. The grammar provides production rules for a piece of Perl code containing two subroutine definitions: `transform()` and `filter()`. Prior to fitness evaluation, each individual converts its genome to a string of code and evaluates it (with Perl's `eval()`), thereby redefining these two functions for that individual. During fitness evaluation, these two functions are called (from a non-evolved function) on data for each protein in the training set. The final result is a secondary structure prediction, which is sent to the fitness function. Figure 1 outlines the flow of data and demonstrates how our fitness measure, $\Delta Q3$, is calculated simply as the difference in correctness between our prediction and the original PSIPRED prediction: $\Delta Q3 = Q3_{GP} - Q3_{PSIPRED}$.

PerlGP's default parameters were used throughout (including population size 2000 and tournaments of 50 individuals of which the fittest 20 reproduce), except for crossover and mutation rates, which were set to 1/100 and 1/200 events per node, respectively. In all cases, duplicate runs were performed for fixed times (see Section 3) in parallel on "identical" machines on a Linux cluster. No migration between populations was allowed unless otherwise stated.

2.4 PDL and Evolved Matrix Manipulation

PDL[14] is one of a number of high-level numerical data manipulation languages, of which MATLAB is perhaps the best known. These languages allow effortless use of arithmetic, trigonometric, and other functions on multi-dimensional arrays of data. The arrays are treated just like scalars, in syntactic terms. For example, if x is a $100 \times 100 \times 100$ floating point array, then operations like $y = x + \log(1 + \mathrm{abs}(x)))$ are possible. In this case, operations are performed element-wise and the result is an array with the same dimensions as x. Routines are usually also available for manipulating the matrix dimensions, slicing, rotating, and so on.

For a number of reasons, Perl is highly inefficient at doing calculations on large arrays. This prompted a group of mostly astrophysicists to develop the PDL Perl library so that large arrays could be stored compactly and manipulated easily. The guts of PDL are coded in C (like many Perl modules) and it is very fast.

An outline of the evolved `transform()` function is shown in Figure 2(a). This is where the evolved matrix operations are performed using PDL. The function inputs one or more $3 \times N$ PDL variables, initialises a $3 \times N$ "memory" or register matrix (with the `zeroes` method), makes a copy of the first input, and then manipulates this copy (and the memory array) before returning the result. The inputs are described in more detail in Section 3. The grammar guiding the generation of the PDL manipulating code is explained in Figure 2(b), and a piece of code from a best of run individual is given in Figure 2(c).

The reader should note that the grammar used here does not allow arbitrary matrix arithmetic in the sense that submatrices of different sizes are manipulated (the application does not seem to need this). PerlGP could be used to evolve PDL code for such manipulations however, as long as safeguards were put in place to handle dimension incompatibilities (as in [6]).

3 Results and Discussion

3.1 Symbolic Filters

In the following sections we will be evolving the `transform()` function to manipulate the matrix of NN outputs from PSIPRED, with the aim to produce better predictions. In PSIPRED, the final prediction is not a simple winner-takes-all transformation of the NN outputs, but involves a few hand-coded filters to clean up some of the noise (which would produce infeasible secondary structures). One of these filters involves the NN outputs, while the others are boolean rules to flip the secondary structure (of a lone strand residue surrounded by coil or helix, for example). Unfortunately, it is not possible to re-use PSIPRED's filters in this study (even though the source code is available) because the transformed matrices are not guaranteed to be in the range 0 to 1, and the filter using NN outputs might not function as intended.

Instead we used GP to find a usable set of *symbolic* filters (i.e. discrete state transitions) based on Perl's search-and-replace function (see Fig. 3(b) for an example of the type of filters which could be evolved). We investigated the effect of allowing different numbers of evolved filters. The results are summarised in Figure 3(a). The `filter()` function chosen for use in the rest of this study (and shown in Fig. 3(b)) performs very similarly to PSIPRED's filters ($\Delta Q3 \approx -0.02\%$).

3.2 Evolved Transforms and Long-Range Information

3.2.1 Global and local means.

In Section 1, the cooperativity of β-sheet formation was discussed, and it was suggested that if the number of predicted strands is very low, then those strands may be incorrectly predicted. In practice though, perhaps the number of residues in predicted strands is more relevant, or maybe the real-valued PSIPRED NN outputs. Our previous unpublished work suggested that the global mean of each PSIPRED NN output was possibly more useful than element-based or residue based information. Back then, our approach was too slow to allow enough data to be gathered to do a proper statistical analysis. Now, with fast, evolved PDL transforms we can perform enough runs to test the null hypothesis that using the mean PSIPRED data makes no difference at all.

But does a global mean make sense from a protein point of view? Natural proteins often contain multiple domains, that is, subunits which can fold independently and may have been swapped around during evolution. β-sheets are rarely shared between domains, and the domains within a protein may have sharply contrasting secondary structural content. Furthermore, very long-range contacts are rare: in our set of training proteins, only 3.2% of intramolecular contacts (including strand-strand pairings) are made between residues more than 300 residues apart. Because it is not always possible to split a sequence correctly into subdomains prior to SSP, local averaging has also been investigated using various window sizes.

```
sub transform {
  # get the arguments
  my ($pdl1, $pdl2) = @_;
  # initialise $mem
  my $mem = $pdl1->zeroes;
  # initialise $out
  my $out = $pdl1->copy;
  # modify $out and $mem
  Statement; # expand using
             # grammar  -->
  # then return the result
  return $out;
}
```

(a)

```
Statement ::= Statement ; [newline] Statement
Statement ::= MatrixLHS *= Matrix
Statement ::= slice(MatrixLHS, Column) *= Vector
MatrixLHS ::= $out | $mem
Column    ::= 0 | 1 | 2
Matrix    ::= Matrix * Matrix
Matrix    ::= Matrix * Vector
Matrix    ::= Matrix * Scalar
Matrix    ::= rotate_horiz(Matrix, Scalar)
Matrix    ::= rotate_vert(Matrix, Scalar)
Matrix    ::= $out | $mem | $pdl1 | $pdl2
Vector    ::= slice(Matrix, Column)
Vector    ::= Vector * Vector | Vector * Scalar
Vector    ::= sumover(Matrix) | minimum(Matrix)
Vector    ::= rotate_vert(Vector, Scalar)
Scalar    ::= Scalar * Scalar
Scalar    ::= sum(Vector) | min(Vector)
Scalar    ::= 0 | 1 | 2 | 3 ...
```

(b)

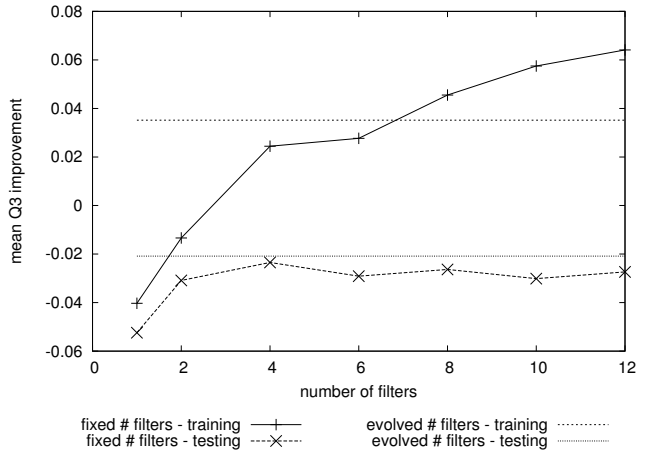
```
$out *= ((($pdl2 + rotate_horiz($pdl2,2) * slice($pdl2, 1)) + ((($pdl2 +
        rotate_horiz($pdl2,2) * slice($pdl2,1)) + pdldiv($mem,$mem)) +
        ($pdl1 * $pdl2))) + (($pdl1 * $pdl2) * $pdl1));
$out -= pow(slice($mem,1),2); # does nothing ($mem still zero)
```

(c)

Figure 2: Generating the PDL `transform()` function. The basic layout of the function is given in (a). The "evolved statements" are generated initially by following production rules from a grammar similar to the one outlined in (b) which is given in Backus-Naur form. The grammar is heavily edited for brevity – for example, all standard arithmetic operations are allowed (not just the element-wise multiplication shown). Note in particular that the `rotate_vert()` function allows access to neighbouring residue information. In (c), an example of evolved contained in `transform()` is given.

A series of GP runs were performed using the `filter()` function derived in Section 3.1 (and shown in Figure 3(b)), and an evolved `transform()` function taking two inputs, the raw PSIPRED output and a locally or globally averaged version of the same data. The averaging is done separately for helix, strand and coil in a circular fashion (wrap-around ends). In one set of runs, the second input was identical to the first (no averaging). For each configuration, mean $\Delta Q3$ over 50 runs after 6h was calculated on the test set data using the final best-of-tournament individuals (selected on training fitness).

The somewhat noisy peak in Figure 4 suggests that applying a local mean with radius 150 to the raw PSIPRED outputs could give better results than applying a global mean. We have two estimates of $\Delta Q3$ for each input treatment. They are, with standard deviations, 0.234% (0.0423) and 0.219% (0.0482) for local mean (150 residue window radius) and global mean respectively. A two-sided t-test gives the result $d = 1.724$, from which we conclude that the difference between the two $\Delta Q3$ means is not significant at the 5% level. Iterated local averaging using smaller window radii gives similar results to global averaging (see also Fig. 4). Because our data originates from the mainly domain-based SCOP database, it is possible that more conclusive results could be obtained using datasets containing more multi-domain proteins. Interestingly, the peak at 150 agrees quite well with the mean size of the proteins in our training set (169 residues). With either local or global means, the improvement in $Q3$ after 6h is significantly greater than that obtained using untreated inputs ($d = 17.5$).



(a)

```
sub filter {
  my $ss = shift;
  $ss =~ s/EHE/EEE/g;
  $ss =~ s/EHC/ECH/g;
  $ss =~ s/CEC/CCC/g;
  $ss =~ s/CHC/CCC/g;
  return $ss;
}
```

(b)

Figure 3: GP is used to find search-and-replace rules to "clean up" raw PSIPRED output. (a) Mean final fitness values ($\Delta Q3$ after 2h, 15 runs each) for training and test sets are shown for different fixed numbers of filters and for runs where the number of filters was variable/evolved. On the test set, unfiltered PSIPRED output would give $\Delta Q3 = -0.144\%$. Before overtraining becomes a problem, our symbolic filters almost reach $\Delta Q3 = -0.02\%$, which we believe is satisfactory. (b) The evolved filter used in subsequent experiments ($\Delta Q3 \approx -0.02\%$).

3.2.2 Majority post-processors.

We performed 50 GP runs for a longer time (24h) using the whole training set (previously this was sampled, see Section 2.1). The inputs were treated with a local mean with window radius 150. Overtraining was judged not to have occurred. The 50 best-of-final-tournament post-processors have a mean $\Delta Q3$ of 0.242% on the test set and 0.264% on training (further details in Table 1). We combine all the post-processors, good and bad, into one using majority voting (at each residue position). The resulting post-processor has a $\Delta Q3$ of 0.30% on test data, and 0.26% on training. From this we conclude that $\Delta Q3$ on a large unseen data set would be approximately 0.26–0.3%.

Put into perspective, however, a 0.3% increase corresponds to just one "improved" residue in every 333 – less than one residue per protein on average! Indeed, there are 122 proteins in our test set which undergo no change in $Q3$ at all. Figure 5 shows a histogram of the non-zero per-protein $\Delta Q3$ values over the test set. Many larger changes occur in the positive direction, and these are not restricted only to the shorter proteins (for which large changes are more likely). For example, for the SCOP domain `d1g73a_` (157 residues) there is a 10.8% rise in $Q3$ (from 67.5% to 78.3%).

3.2.3 Per-state accuracy.

If we measure the contribution made by each type of secondary structure to the 0.3% increase in $Q3$ we see (in Table 1) that only the prediction of helix and coil residues are improved, at the expense of strand accuracy (strand residues comprise only 20% of proteins on average). It would clearly be desirable to have a post-processor which made small improvements in all three states. We have performed preliminary experiments with modified fitness functions to encourage this. So far we have evolved a majority post-processor with $\Delta Q3 = 0.24$ ($\Delta Q_{helix} = 0.11$, $\Delta Q_{strand} = 0.11$, $\Delta Q_{coil} = 0.41$). The modified fitness landscapes seem to be harder to search but we anticipate better overall solutions.

Table 1: Results for the majority post-processor (using 50 individuals trained for 24h)

	individual post-processors			majority post-processor			
dataset	mean $\Delta Q3$	min $\Delta Q3$	max $\Delta Q3$	$\Delta Q3$	ΔQ_{helix}	ΔQ_{strand}	ΔQ_{coil}
training set	0.264	0.203	0.301	0.26	0.22	-1.86	1.31
test set	0.242	0.124	0.299	0.30	0.24	-1.75	1.41

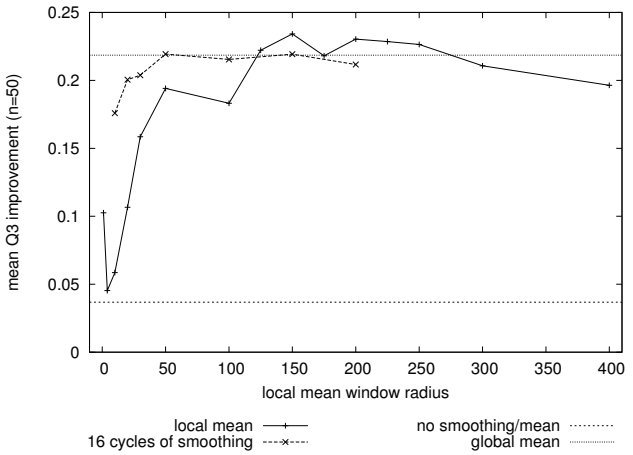


Figure 4: Locally averaged PSIPRED outputs are given as input for GP evolved `transform()` function. The mean test set fitnesses ($\Delta Q3$) from 50 runs are shown for various windows sizes. Averaging is applied either once, 16 times, or not at all (as a control). The results using a globally meaned input are also given. Local means do not perform significantly better than the global mean at the 5% level (see text), despite the possible peak around window radius 150.

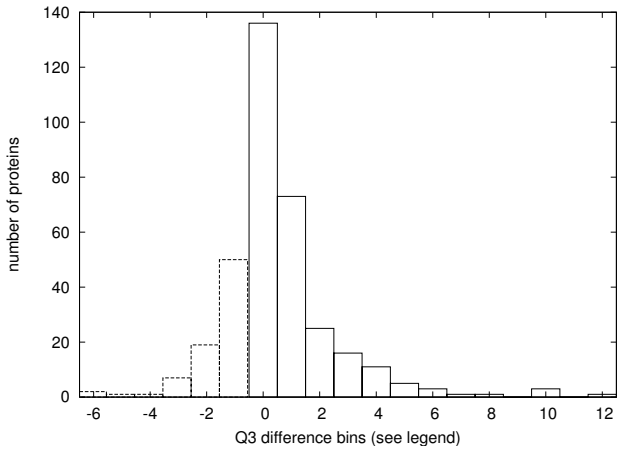


Figure 5: Histogram of $\Delta Q3$ for 477 test set proteins using a majority predictor which uses 50 individuals evolved for 24h. The bins are the rounded integer values of *per-protein* $\Delta Q3$. The solid bars indicate positive changes, the dashed bars indicate a negative change, and the 122 proteins with zero change in $Q3$ are not shown.

4 Final Remarks

We have made a small but statistically significant increase in SSP accuracy, as measured by $Q3$ on our test set, using evolved post-processors for PSIPRED which make use of long-range information (global and local averaging). Indisputable evidence that we have actually improved SSP could only come from extensive blind testing, such as in the EVA continuous evaluation experiment[3]. Because a typical end-user would not appreciate a 0.26% increase in $Q3$, this work should be viewed as the first of a number of steps on the path to better SSP by post-processing. In future work we will provide GP with higher-level matrix manipulation functions, such as window averaging functions (here we applied them offline), or Fourier transforms. We also plan to incorporate other sources of information, including contact map predictions.

References

[1] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. H. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nuc. Ac. Res.*, 25:3389–3402, 1997.

[2] S. E. Brenner, P. Koehl, and M. Levitt. The ASTRAL compendium for protein structure and sequence analysis. *Nuc. Ac. Res.*, 28(1):254–256, 2000.

[3] V. A. Eyrich, M. A. Marti-Renom, D. Przybylski, M. S. Madhusudhan, A. Fiser, F. Pazos, A. Valencia, A. Sali, and B. Rost. EVA: continuous automatic evaluation of protein structure prediction servers. *Bioinformatics*, 17(12):1242–1243, Dec 2001.

[4] D. T. Jones. Protein secondary structure prediction based on position- specific scoring matrices. *J. Mol. Biol.*, 292:195–202, 1999.

[5] W. Kabsch and C. Sander. Dictionary of protein secondary structure — pattern-recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22:2577–2637, 1983.

[6] M. Keijzer. *Scientific Discovery using Genetic Programming*. PhD thesis, Danish Technical University, Lyngby, Denmark, March 2002.

[7] J. R. Koza. *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT press, Cambridge, MA, 1992.

[8] R. M. MacCallum. Introducing a Perl Genetic Programming System: and Can Meta-evolution Solve the Bloat Problem? In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 369–378, 2003.

[9] J. Meiler and D. Baker. Coupled prediction of protein secondary and tertiary structure. *Proc. Natl. Acad. Sci. USA*, 100(21):12105–12110, Oct 2003.

[10] D. J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman, Inc., 10 Moulton Street, Cambridge, MA 02138, USA, 7 May 1993.

[11] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP — a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, 247:536–540, 1995.

[12] B. Rost. Review: protein secondary structure prediction continues to rise. *J. Struct. Biol.*, 134(2-3):204–218, 2001.

[13] S. Silva. GPLAB - A Genetic Programming Toolbox for MATLAB. http://www.itqb.unl.pt:1111/gplab.

[14] C. Soeller, R. Schwebel, T. J. Lukka, T. Jenness, D. Hunt, K. Glazebrook, J. Cerney, and J. Brinchmann. The Perl Data Language. http://pdl.perl.org.