The Prediction of the Degree of Exposure to Solvent of Amino Acid Residues via Genetic Programming

Simon Handley

Computer Science Department
Stanford University
Stanford, CA 94305
(415) 723-4096 shandley@cs.stanford.edu

Abstract

In this paper I evolve programs that predict the degree of exposure to solvent (the *buriedness*) of amino acid residues given only the primary structure. I use genetic programming (Koza 1992; Koza 1994) to evolve programs that take as input the primary structure and that output the buriedness of each residue. I trained these programs on a set of 82 proteins from the Brookhaven Protein Data Bank (PDB) (Bernstein et al. 1977) and cross-validated them on a separate testing set of 40 proteins, also from the PDB. The best program evolved had a correlation of 0.434 between the predicted and observed buriednesses on the testing set.

Introduction

One of the most fundamental problems in molecular biology is the prediction of tertiary structure from primary structure: the *protein folding problem*. The goal of protein folding is the prediction of one feature of a folded protein (the 3D coordinates of its backbone atoms) from another feature (the sequence of amino acid residues that make up the protein). The protein folding problem is of enormous practical importance because the latter feature (the primary structure) is much easier to establish than the former (the tertiary structure).

A related problem is the *buriedness problem*: the prediction of the degree of exposure to the solvent (the *buriedness*) of each amino acid residue in a folded protein. Some amino acid residues will have a buriedness of 0%: these are in the core of the protein and are likely hydrophobic. Other residues will have a buriedness of 100%: these are on the surface of the protein and are probably hydrophilic.

The buriedness problem is interesting because it is a simplified version of the protein folding problem.

In this paper I will show that genetic programming (Koza 1992; Koza 1994) does find programs that predict the buriedness of residues. These programs work better than would be expected of randomly generated programs and there is very little externally imposed bias towards any particular sizes, shapes/architectures or compositions.

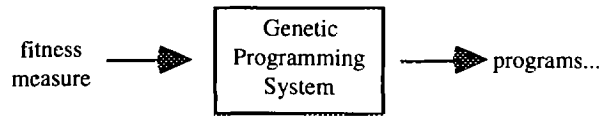
Previous Work

Holbrook et al. (Holbrook, Muskal, & Kim 1990; Holbrook, Muskal, & Kim 1993) trained neural networks to classify windows of contiguous residues (a central residue plus a fixed number of flanking residues on either side). This classification was either two-state ([0%...20%) and [20%...100%]) or three-state ([0%...5%), [5%...40%) and [40%...100%]). They trained by backpropagation both perceptrons and one-hidden-layer nets on a training set of 20 proteins from the Brookhaven Protein Data Bank (PDB) (Bernstein et al. 1977) and cross-validated on a testing set of 5 proteins, also from the PDB. They achieved a prediction accuracy of $Q_3 = 72.0\%$ on the two-state prediction problem and $Q_3 = 52.0\%$ for three-state prediction [1].

Genetic Programming

Genetic programming (Koza 1992; Koza 1994) is an inductive learning technique that is particularly suited to problems in which some underlying regularity or structure must be discovered.

Genetic programming can be thought of as a black box that takes in a fitness measure and returns a stream of programs that are (hopefully) fit according to the given measure:



Genetic programming starts with a population of randomly generated computer programs and a fitness measure and uses artificial selection and sexual

[1] Q_3 is the ratio of correct predictions over the total number of predictions.

Table 1. The Brookhaven identifiers of the 122 proteins used in this paper.

1CSE_E, 1CSE_I, 1ECO, 1FKF, 1FNR, 1FXD, 1GCR, 1GD1_R, 1GP1_B, 1GPB, 1HIP, 1HOE, 1HRH_B, 1HSA_D, 1HSA_E, 1HYP, 1IFB, 1LPE, 1LZ1, 1MBD, 1MSB_B, 1NN2, 1NXB, 1OVA_D, 1PAZ, 1PGX, 1PHH, 1PRC_C, 1PRC_M, 1PRC_H, 1R69, 1RBP, 1RHD, 1RNB_A, 1RNH, 1ROP_A, 1SN3, 1TFD, 1TIE, 1TPK_C, 1UBQ, 1UTG, 1WSY_A, 1WSY_B, 1YCC, 256B_B, 2AZA_B, 2CA2, 2CCY_B, 2CDV, 2CPP, 2CSC, 2ER7_E, 2FCR, 2GN5, 2HAD, 2HBG, 2HMZ_D, 2LH7, 2LIV, 2LTN_D, 2MCM, 2OVO, 2PAB_B, 2RHE, 2RSP_B, 2SAR_B, 2SDH_B, 2SGA, 2SIC_I, 2SNS, 2SOD_Y, 2STV, 2TRX_B, 2TS1, 2TSC_B, 2WRP_R, 2YHX, 3ADK, 3B5C, 3BCL, 3CHY, 3CLA, 3GAP_B, 3GRS, 3IL8, 3PGK, 3RUB_S, 3SDP_B, 451C, 4BP2, 4CPV, 4ENL, 4FXN, 4I1B, 4LZM, 4MDH_B, 4PFK, 4PTP, 4SGB_I, 4XIA_B, 5ABP, 5ACN, 5CPA, 5HVP_B, 5P21, 5PTI, 5RUB_B, 5RXN, 5TIM_B, 6LDH, 6TMN_E, 7RSA, 8ADH, 8ATC_C, 8ATC_D, 8CAT_B, 8DFR, 9ICD, 9PAP, 9RNT, 9WGA_B.

reproduction to produce increasingly fitter populations of computer programs.

A *run* of genetic programming is usually a fixed number of generations of such evolution. A *program* is a composition of functions and terminals. For example, `1` is a program that consists of a single terminal; `(+ 1 2)` is another program, it is a composition of two terminals (`1` and `2`) and a function (`+`).

The programs in each generation of a run of genetic programming are created either by copying them from the previous generation (with probability proportional to their fitness) or by crossing over two parental programs (also chosen with probability proportional to their fitness) and placing the two resulting child programs in the new generation.

The crossover operation works as follows. Let the two parental programs be

```
(+ (* 1 2) (* 3 4)), and
(+ (+ 5 (+ 6 7)) 8).
```

The crossover operator randomly chooses an internal point in each program. Two such choices are highlighted here:

```
(+ (* 1 2) (* 3 4)), and
(+ (+ 5 (+ 6 7)) 8).
```

The crossover operator then swaps these two fragments—`(* 3 4)` and `(+ 6 7)`—to produce the child programs:

```
(+ (* 1 2) (+ 6 7)), and
(+ (+ 5 (* 3 4)) 8).
```

Although the above black box has only one input—the fitness measure—there are actually four main inputs to a genetic programming system: the fitness measure, the class of programs, the population size and the maximum number of generations to be evolved.

Choosing The Class of Programs To Be Evolved

The most important input to a genetic programming system is the class of programs to be evolved. This means choosing a *function set* and a *terminal set* from which the programs will be made up. For example, the programs in the "Genetic Programming" section above are drawn from a function set of {+, *} and a terminal set of {1, 2, ..., 8}. The terminal set can be thought of as the inputs to the problem and the function set contains the types of computation you expect to have to do to those inputs.

For this problem, the minimum sufficient terminal set is the amino acid residue at each position in the primary structure (Anfinsen 1973). In addition, the programs have access to the hydrophilicity (Kidera et al. 1985) of each residue, the bulk (volume) of each residue and some random floating point constants.

The function set contains two types of functions. First there are functions that do simple computations: {`*`, `-`, `*`, `%`, `if<=`} where `(if<= a b c d)` = c if $a \leq b$; d otherwise; and where `(% a b)` = a/b if $b \neq 0$; 1 otherwise.

The remaining functions in the function set allow the programs to look at multiple residues. Although the buriedness problem as stated above is a mapping from a list of amino acid residues to a list of buriednesses, it is easier to recast the problem as a mapping from a single amino acid residue to a single buriedness. A solution to this simpler problem can then be applied to each amino acid residue separately to determine the buriedness of all amino acid residues in a protein. A program to solve this simpler problem, however, must still look at the residues surrounding the residue whose buriedness it is computing. The following "turtle" functions provide this contextual information. The idea is that the program has access to a turtle (or read head) that walks to the left (towards the N-terminal) or to the right (towards the C-terminal) along the primary structure (or Turing tape). The program can look at the current residue (`res` returns a number corresponding to the current residue[2], `hydro` returns the current residue's hydrophilicity (in the range [-1.25...2.06] and `bulk` returns the volume of the current residue (in the range [-2.16...2.08])[3]), move the turtle to the left by various amounts (`(left-1)`, `(left-2)` and `(left-3)`), move the turtle right (`(right-1)`, `(right-2)` and `(right-3)`) and rubber-band the turtle back to some initial position (`(home)`).

Finally, I scale the output of each program to be in the range 0% to 100%. That is, to predict the buriedness of a protein with a given program, the program is first run on

[2]There are 23 "residues": "don't know" (X), "D or E" (B), "N or Q" (Z) plus the 20 naturally occurring residues. These 23 residues are randomly assigned the numbers 0...22.

[3]The hydrophilicity and bulk metrics used are from columns 1 and 2 of Kidera et al.'s Table III (Kidera et al. 1985).

each amino acid residue in the primary structure. This results in a list of floating point numbers, one per residue. I then scale these numbers so that the largest is 100.0 and the smallest is 0.0. I then interpret each number as the predicted percentage buriedness of the corresponding residue.

Here's an example:

```
(if<= hydro (left-1 hydro)
    (* 2 (+ (home (left-1 bulk))
            (right-1 bulk)))
    (% bulk (right-1 bulk))).
```

This program does one of two things: if the current residue is less hydrophilic than the one to its left then it returns twice the sum of the bulkinesses of the residues immediately to the left and to the right; otherwise it returns the ratio of the bulkiness of the current residue and the residue to its immediate right. To predict the buriedness of all the residues in a protein, this program will be run separately on each residue in the primary structure.

In summary, the terminal set contains functions that return information about the residue that the turtle is looking at plus some random constants,

$$T = \{ \texttt{res}, \texttt{hydro}, \texttt{bulk}, \Re \}$$

where $\Re$ is replaced in the initial generation of randomly-created computer programs by random floating point constants in the range [-10.0,+10.0].

The function set contains two types of functions: those that do simple computations and those that move the turtle around:

$$F = \{ +, -, *, \%, \texttt{if<=} \} \cup$$

$$\{ \texttt{home}, \texttt{left-1}, \texttt{left-2}, \texttt{left-3}, \texttt{right-1}, \texttt{right-2}, \texttt{right-3} \}.$$

Choosing The Remaining Inputs

The fitness measure should reward programs that perform well at the desired task. Here I want to evolve programs that are good predictors of buriedness. So the fitness of a program is defined as the similarity between its predictions and the known buriednesses (as computed by Kabsch and Sander (Kabsch & Sander 1983) from the 3-D coordinates in the PDB). I quantified similarity as the correlation coefficient between the two real-valued curves (predicted and observed buriednesses as a function of residue number).

Two questions remain: First, how many individuals will there be in each population? This is the *population size*, M. Second, for how many generations should the evolution run? This is the *maximum number of generations*, G.

The population size should be large enough that there is enough genetic diversity to solve the problem. It should be small enough that useful results can be obtained in a reasonable amount of time on whatever computers are available. For the results shown here I used a population size, M, of 4,000.

Similarly, the maximum number of generations should be chosen to optimize computer time. I chose G = 76 (1 initial random population + 75 subsequent evolved populations) for the runs reported here.

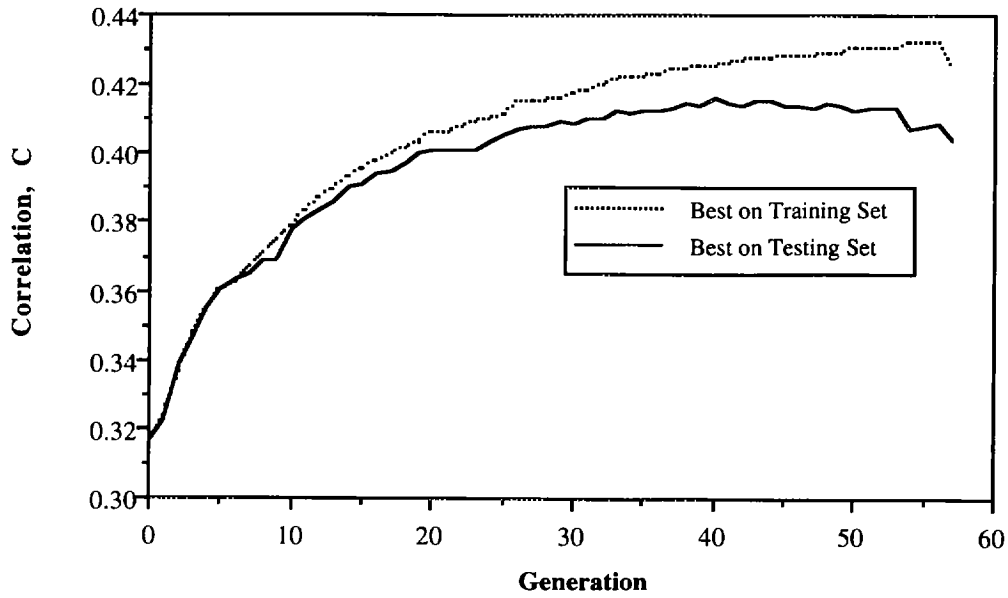


Figure 1. Correlation between the observed and predicted buriedness for best (on the training set) individual of each generation, tested on both the training set and the testing set. Averaged over the 15 runs. Note that the values for generation 58 are based on only one run.

The Data

I used 122 canonical proteins from the PDB that Tod Klingler (personal communication) selected for me. These 122 proteins are canonical in the sense that there is minimal sequence identity and evolutionary homology between them. Table 1 shows the Brookhaven identifiers of these proteins.

To cross-validate the predictive power of the programs that evolved, I divided the 122 proteins into a training set and a testing set. The 122 proteins were partitioned so that one third of the proteins were in the testing set and the remaining two thirds made up the training set. The possible testing sets were: proteins 0–39, proteins 40–79, and proteins 80–119; one of these three testing sets was chosen randomly for each run.

Results

I did 15 runs with M = 4,000 and G = 76. An average of 33.8 generations were evolved for each run. The evolutionary process was driven by the proteins in the training set (two-thirds of the 122 proteins). That is, I defined fitness as the similarity between a program's predictions and the observed reality for the proteins in the training set. Separately, I reran the best individual of each generation of each run on the testing set (the remaining one-third of the 122 proteins). The difference between the predictive performance on the training and testing sets gives a measure of how much each program is overfitting the training data; that is, the degree to which a program does better on the training set than the testing set shows to what degree the program is looking at irrelevant aspects of the problem (i.e., aspects that aren't shared by the proteins in the testing set).

Figure 1 plots the correlation between the observed and predicted buriedness curves on both the training and testing sets, averaged over the 15 runs. The best performance on the testing set of all the generation 0 individuals (i.e., the best of the best of generation 0s, *not* the average of the best of generation 0s) is C = 0.359. This indicates the ability of random search to solve this problem.

Table 2 shows two best-of-generation ("best" here meaning best on the training set) individuals that had high predictive power on the testing set.

Conclusions

I have shown that it is possible to evolve programs that predict the degree to which an amino acid residue is exposed to the solvent. The best program produced by the evolutionary process had a correlation of 0.434 between the observed and predicted buriednesses on the testing set of 40 proteins.

Acknowledgments

I'd like to thank: John Koza and James Rice for their support and genetic programming expertise; Tod Klingler for selecting the proteins; and the anonymous reviewers for their comments.

References

Anfinsen, C. B. Principles that govern the folding of protein chains. *Science* 81 (1973): 223–30.

Bernstein, F. C.; Koetzle, T. F.; Williams, G. J. B.; Meyer, J., E. J.; Brice, M. D.; Rodgers, J. R.; Kennard, O.; Shimamouchi, T.; and Tasumi, M. The protein data bank: A computer based archival file for macromolecular structures. *Journal of Molecular Biology* 112 (1977): 535–42.

Holbrook, S. R.; Muskal, S. M.; and Kim, S.-H. Predicting surface exposure of amino acids from protein sequence. *Protein Engineering* 3 (8 1990): 659–65.

Holbrook, S. R.; Muskal, S. M.; and Kim, S.-H. "Predicting protein structural features with artificial neural networks." In *Artificial Intelligence and Molecular Biology*, ed. Hunter, L. 161–94. AAAI Press, 1993.

Kabsch, W. and Sander, C. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers* 22 (1983): 2577–637.

Kidera, A.; Konishi, Y.; Oka, M.; Ooi, T.; and Scheraga, H. A. Statistical analysis of the physical properties of the 20 naturally occurring amino acids. *Journal of Protein Chemistry* 4 (1985): 23–55.

Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press, 1992.

Koza, J. R. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press, 1994.

Number of points: 141. Occurred at generation: 51. Performance on training set: **C = 0.429**, two-state Q_3 = 68.5%, three-state Q_3 = 66.9%. Performance on testing set: **C = 0.434**, two-state Q_3 = 70.0%, three-state Q_3 = 68.4%.

```
(if<= (* hydro -6.02425) (right-2 (left-2 (* hydro -4.27684))) (if<= hydro (% hydro (+ (+ (* hydro -8.22096) 6.5959) hydro)) (if<= hydro (* hydro -4.27684) (left-1 (home res)) (if<= hydro (if<= hydro (% hydro -3.20838) (left-1 7.08283) hydro) (if<= hydro (% (+ (- bulk hydro) 6.5959) (* res (* 6.5959 res))) -3.20838 (home hydro)) -7.89316)) (if<= hydro -8.22096 (if<= hydro (* hydro -6.02425) (left-2 hydro) (left-1 7.08283)) -7.89316)) (if<= hydro (- (- bulk hydro) hydro) (if<= (home res) (if<= res bulk hydro (home (if<= hydro (* hydro -3.20838) (left-1 7.08283) hydro))) (% hydro (+ (+ bulk bulk) hydro)) (- (+ bulk bulk) (- -4.89054 (+ hydro -1.88183)))) (if<= hydro (* hydro -3.20838) (home (- (+ (+ bulk bulk) bulk) hydro)) (if<= hydro (* hydro -3.20838) (if<= hydro (* hydro -3.20838) (home (- (- bulk hydro) hydro)) (home hydro)) (home res)))))
```

if $H_0 \le 0.77$ **then**
  if $H_{-1} \le H_{-2}$ **then**
    if $H_{-2} \le \dfrac{B_{-2} - H_{-2} + 6.6}{6.6 R_{-2}^2}$ **then**
      -3.21
    else
      H_{-2}
    endif
  else
    -7.89
  endif
else
  -7.89
endif

Number of points: 79. Occurred at generation: 37. Performance on training set: **C = 0.428**, two-state Q_3 = 69.6%, three-state Q_3 = 68.0%. Performance on testing set: **C = 0.432**, two-state Q_3 = 71.1%, three-state Q_3 = 69.5%.

```
(if<= bulk (- (% (+ -1.74019 hydro) hydro) (+ -1.74019 hydro)) (+ (* (home (- res 0.507344)) (% bulk 5.08997)) (- (left-2 8.6028) (home (home (right-3 hydro))))) (if<= (left-1 (+ -1.74019 hydro)) (right-2 res) (if<= (left-1 hydro) (- hydro hydro) (+ (left-2 8.6028) (- (- (left-2 8.6028) (home (right-3 hydro))) (home (home (right-3 hydro))))) (if<= bulk (- res 0.507344) (home -4.11196) (- -4.86839 (if<= res (home (- res 0.507344)) (home (right-3 hydro)) (+ -1.74019 hydro))))) (% bulk 5.08997)))
```

if $B_0 \le (H_0 - 1.74)/H_0 - H_0 - 1.74$ **then**
  $5.09 \dfrac{R_0 - 0.51}{B_0} + 8.6 - H_1$
else
  if $H_{-1} - 1.74 \le R_{+1}$ **then**
    if $H_0 \le 0$ **then**
      $17.2 - H_{-1} - H_{+2}$
    else
      if $B_0 \le R_0 - 0.51$ **then** -4.11 **else** $-3.12 - H_0$ **endif**
    endif
  else
    $B_0/5.1$
  endif
endif

Table 2. Two of the five best (on the testing set) best-(on the training set)of-generation individuals. (The other three were minor variations of these two.) The table shows both the actual function that evolved and a simplified version of the function. The simplifications were done by hand and assumed that turtle movements were always successful (this means that `(left-2 (right-2 x))` simplifies to x which is not true if the turtle is within 1 residue of the N-terminal: the attempt to move left two residues would fail but the attempt to move right two residues would succeed. (Notation: H_δ is the hydrophilicity of the residue δ residues away from the current residue, B_δ and R_δ are defined similarly.)