PREDICTION OF PROTEIN SECONDARY STRUCTURE USING GENETIC PROGRAMMING

Summer Internship Project Report
During June-July 2003

Under: Dr. Bob MacCallum
Stockholm Bioinformatics Center
Stockholm University, Sweden

Varun Aggarwal
Electronics And Communication Engg.
Netaji Subhas Institute of Technology
New Delhi- 110048

Contents

Certificate

This is to certify that, **Varun Aggarwal**, (104/ECE/2000) a student of NSIT, Delhi, India did his summer training under me at Stockholm Bioinformatics Center for the months of June-July 2003. He worked on two projects documented in this report.

Robert M. MacCallum
Assistant Proffessor

Stockholm Bioinformatics Center
SCFAB
Stockholm University
S-106 91 Stockholm
Sweden

phone: +46 (0)8 5537 8567
fax: +46 (0)8 5537 8214
email: maccallr@sbc.su.se

Acknowledgement

I will like to thanks **Dr. Bob MacCallum** for giving me this opportunity to work with his group. I hugely benefited and wish to thank profusely for spending time with me explaining bioinformatics, genetic programming and perl programming concepts. I also enjoyed the stimulating discussions we had on various concepts and ideas. It was a pleasure working under his guidance.

Varun Aggarwal
Electronics and Communication Engg.,
Netaji Subhas Institute of Technology,
New Delhi, India

The Group

During my internship, I worked in the group of Dr. Bob Maccallum, Assistant Professor, Stockholm Bioinformatics Center, (**www.sbc.su.se/~maccallr**).The group comprises of six members including myself, three master students and one post-doc.

The group's major interest is to develop machine learning and evolutionary computation techniques to understand protein folding and structure. A protein starts life inside the cell as a simple linear chain of amino acids, but quickly and efficiently folds into a specific three-dimensional shape which dictates how it interacts with other molecules in the cell or organism. This interplay of proteins and other biomolecules is the very essence of life.

An organism's DNA specifies the exact sequence of amino acids for every protein. In turn, the sequence of amino acids specifies the three-dimensional shape of the folded protein. The folding process is difficult to observe experimentally and currently impossible to simulate accurately in a computer. Therefore our limited understanding of protein folding has not yet resulted in a computational prediction method, which can accurately produce a 3D structure given a sequence as input.

Living organisms have overcome many difficult challenges, such as surviving in extreme climates or flying through the air, by the process of evolution, driven by chance mutation events, selection and reproduction. Evolutionary computation is an area of computer science, which takes inspiration from Nature as it tries to evolve solutions to difficult problems. We hold the view that it may be possible to evolve simplified but accurate models of protein folding that no human has so far been able to come up with. We are particularly interested in genetic programming, which is the automatic evolution of computer programs, as the means to achieve this. We pay particular attention to developing evolutionary algorithms which follow the biological lead. One example is our exploration of meta-evolution - where not only the 'organisms' evolve but also the evolutionary mechanisms.

At the same time as working on algorithms, we are also looking at aspects of sequence-structure relationships with other techniques which we hope will give valuable insight into protein structure and function.

The ultimate goal is to understand complex biological systems with exciting new biologically inspired machine learning algorithms.

*(From: **www.sbc.su.se/~maccallr**)*

Using SOM and Genetic Programming to predict Protein Secondary structure

Introduction

Proteins are polymers of amino acids. A protein consists of amino acids (there are 20 amino acids) bonded together in various ways. Each protein folds in a unique way giving rise to a 3-D secondary structure. The 3-D structure basically contains three structural elements, i.e. helix, strands and coils. Figure 1 illustrates the 3-D primary secondary structure of a protein. (For further reading, one may consult, “Introduction to Protein Structure”, by Branden and Tooze.)

The 3-D structure of protein has been of keen interest to researchers (eg. biologists) for long. The structure of the protein is useful for understanding its functionality and is therefore useful in development of drugs in the pharmaceutical industry. Moreover, they are of great academic interest since they give essential hints about evolution of different species.

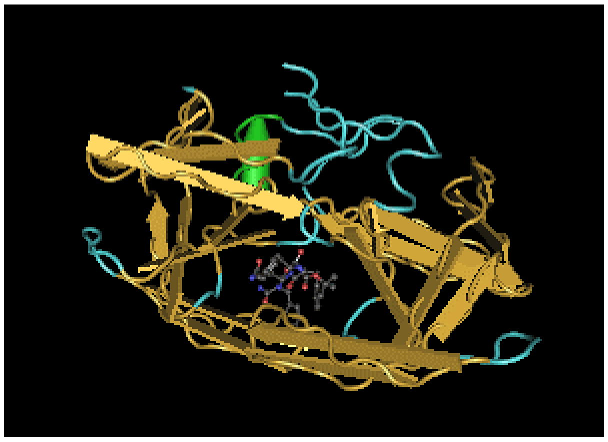


Figure 1: Depiction of 3-D structure of a protein

There are experimental ways of finding out the structure of protein using X-RAY crystallography, which involves human effort and lot of time. However, since a particular amino acid sequence maps to a unique structure, (even if the same protein exists in different organisms), it is logical to believe that the amino acid structure contains full information about the protein structure. Therefore prediction of protein sequence should be possible, if a one-to-one mapping (function) between the amino acid sequence and the protein structure can be identified.

Since the 3-D structure is quite complicated and difficult to predict, researchers around the world made an effort towards predicting the secondary structure of the protein. The secondary structure simply tells whether a given amino acid in a given protein is a helix, strand or coil. Therefore an amino acid sequence can be mapped into a sequence of helices (H), extended strands (E) and coils (C). This sequence of structural elements is called the secondary structure of the protein. Figure. 2 clarifies this.

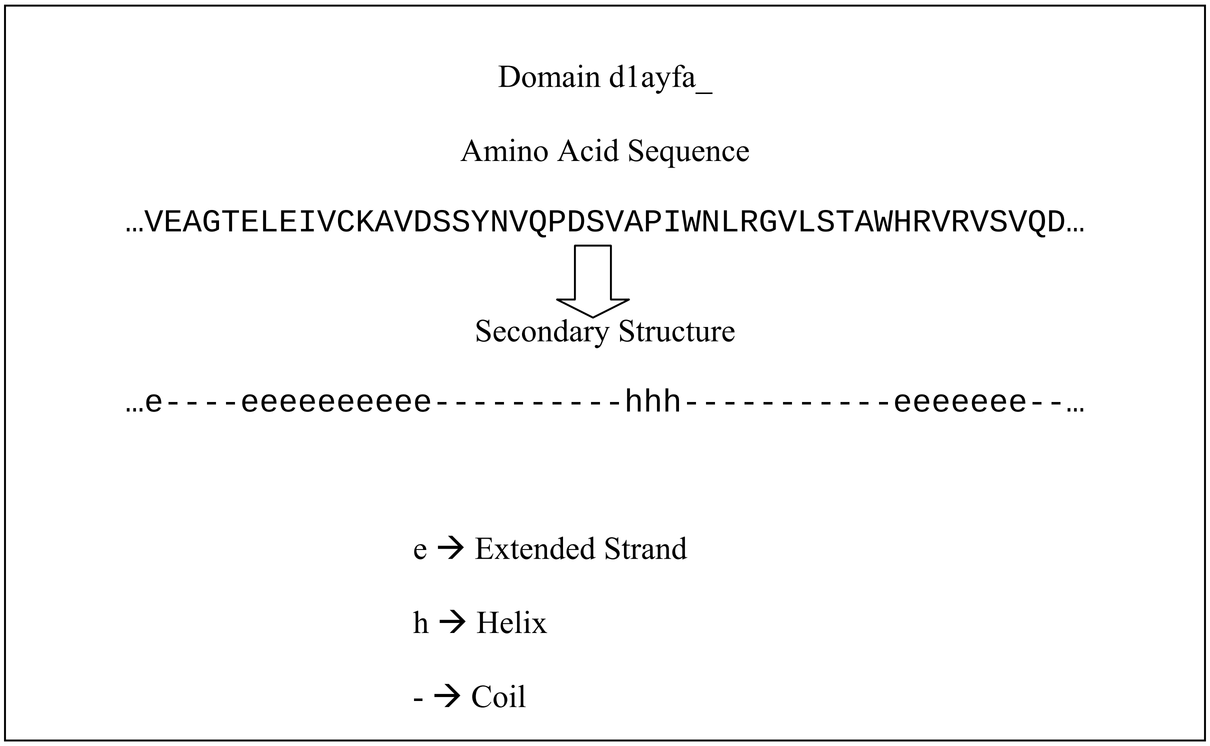

Domain d1ayfa_
Amino Acid Sequence
…VEAGTELEIVCKAVDSSYNVQPDSVAPIWNLRGVLSTAWHRVRVSVQD…
Secondary Structure
…e----eeeeeeeeee----------hhh-----------eeeeeee--…
e → Extended Strand
h → Helix
- → Coil

Figure 2. Depiction of Protein Secondary Structure for domain d1ayfa_i

The problem of predicting the 3-D structure of proteins was thus broken into two parts: 1. Prediction of secondary structure from amino acid sequence, 2. Prediction/Mapping of teritiary 3-D structure from secondary structure. In the present project, we deal with the first part, i.e. prediction of protein secondary structure.

The prediction of protein structure is being explored since 1960, however the most ground breaking and interesting studies came through use of Neural of neural networks for prediction, which gave a protein prediction accuracy of 76% [3]. A complete survey of the different methods of prediction of protein secondary structure in given in [1].

In this project we attempted to use Genetic Programming (GP) to predict the secondary structure of protein. Genetic Programming is basically a search/optimization algorithm, which attracted global attention through the book "Genetic Programming: On the Programming of Computers by Means of Natural Selection", by Prof. John Koza [2]. GP is a class of algorithms that try to 'evolve' programs to solve a given problem. It primarily requires the ideal input and output dataset to calculate the performance of the evolving program. A typical GP would begin with a random population of programs and try to search for a solution using principles of natural evolution, such as 'survival of the fittest', crossovers, mutations, etc. GP is generally sensitive to parameter variation and there is no guarantee that GP would be able to solve a given problem. (However, there have been numerous publications, where GP has been able to solve NP-Complete problem, regression problems, circuit design problems, etc.)

We used the multi-aligned amino acid sequence [4] as the input data to the GP and wanted as output the correct secondary structure of the protein. Since the input data set is highly dimensional, it becomes difficult for the GP to use it effectively. Therefore, we decided to use Self Organizing Maps (SOM) to transform the initial higher dimensional data (the multi-aligned amino acid sequence) to lower dimensional data (positions in the SOM). SOM is a neural network tool, which uses unsupervised (competitive) learning to classify data into positions in a map. One can choose the dimensions of the map (analogous to number of classification parameters) and size of the map (analogous to total number of different classes). In this way, we could map each amino acid in a protein into a position in SOM, equivalently characterized by the co-ordinates of the position in the input to GP. Some nice introductory text on SOM can be found at http://davis.wpi.edu/~matt/courses/soms/#Introduction.

We hoped that this genetic programming approach could give us probability higher than the current 76%.

My Work

Algorithms to train the SOM were already coded by the group. I had to do the following things:

- I had to interpret and try to figure out patterns in the SOM data to judge whether it contains ample information about the secondary structure of the data.
- Vary various parameters of SOM such as the window size, map size, etc. and observe their effect on the data.
- Find innovative ways to further preprocess the SOM data before sending to the genetic algorithm.
- Work upon the Grammar of the evolving genetic program and tune its parameters through intuition, experiments and observation for better results.

Tools Used

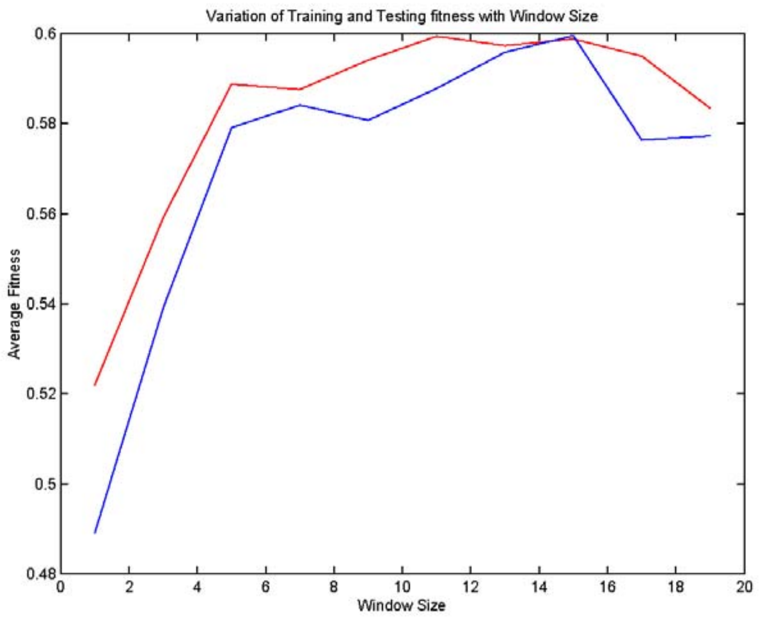
Perl, PDL, PerlGP, Matlab, PDB Resources, etc.

Experiments and Observations

Studying the SOM data

The SOM data mapped each amino acid into a position in the SOM. Therefore, each amino acid is depicted by the coordinates of its positions in the map in the input data for the GP. (Details of how multi-aligned sequences were mapped to SOM positions can be requested from Dr. Bob MacCallum)

It was so hypothesized that each place in the map shall correspond to a single secondary structural element, i.e. coil, strand or helix. We did statistical tests on the map to see whether this hypothesis was correct. We trained the map using 100 proteins, allotting each place in the map the secondary structure, which hit it the maximum time. Then we tested the maps on another independent set of 100 proteins and found the percent correct prediction (secondary structure) to judge whether our hypothesis was correct. Results for a map of 6X6X6 with varying window sizes is included underneath.



🟥 Training % Correct Prediction 🟦 Testing % Correct Prediction

Figure 3: Variation of training and test fitness with window size

It is thus observed that the window size of 15 gives the least difference between the training and testing fitness, thus substantiating our claim. Also, previous studies using Neural Networks stated 15 as the optimum window size.

In the aforesaid manner, prediction through Statistical Evaluation of Kohonen Maps gave a prediction accuracy of 59.94% for a window of 15.

I further studied the errors of mapping each amino acid in the SOM. We believed that we could see some pattern between the secondary structural element and the range of error. Figure. 4 depicts results of experiments over 100 proteins.

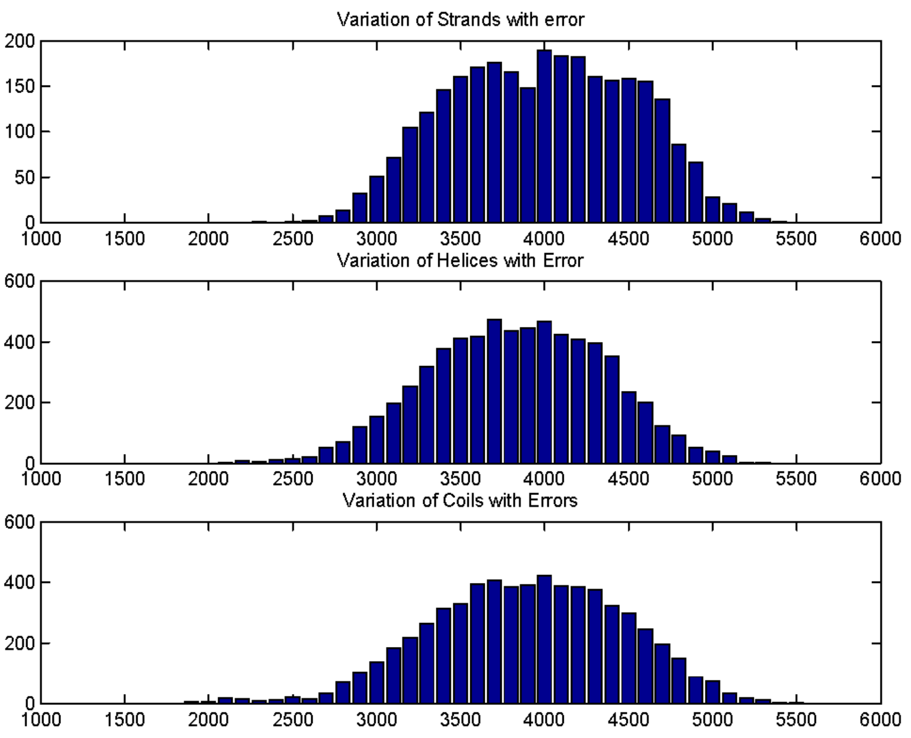


Figure 4: Variation of number of strand, helices and coils with value of error. X axis depicts error value, while the Y axis shows the number of secondary structural element

One could observe that the graph for strands was noisy. Though, no direct inference could be made from this data. We felt(intuitively) that wrongly predicted secondary structure would have higher values of error, however statistical tests proved the hypothesis to be wrong.

I believed, maybe, the length of the amino acid sequence contained some information about the wrongly predicted secondary structure. Following is the plotted data.

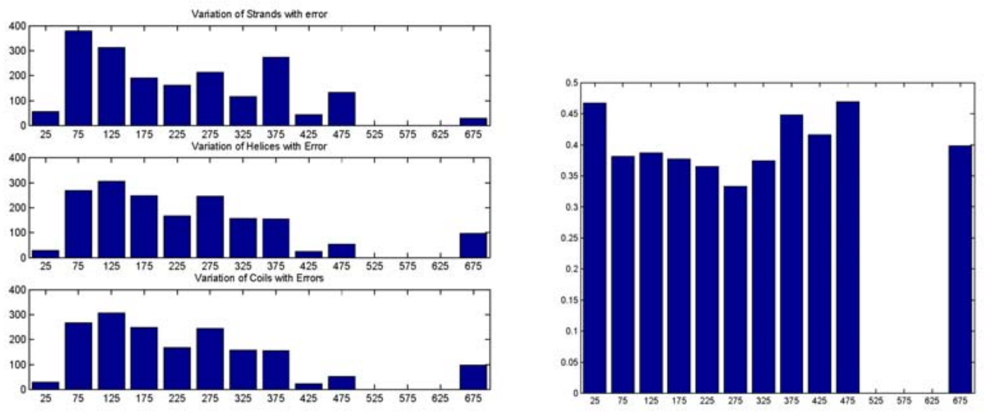


Figure 5: Number of wrongly predicted structures vs. length of protein.

The first graph shows the number of wrongly predicted secondary structure (individually for strand, coil and helix) for different lengths. The second graph shows the total number of wrongly predicted secondary structure for each length. Clearly, one cannot see any bias towards predicting wrong secondary structure for particular lengths of proteins. Hence this information was also regarded useless as far as this project was concerned.

The next approach to process the SOM data was using smoothing. In carrying out statistical tests on the SOM, one could calculate the probability of occurrence of each kind of secondary structure for each place in the SOM. When the SOM was applied on the test set, a record of this probability was made for each position on the map. On plotting this probability on a sequence of amino acids (protein), we observed that the curve was quite noisy. However the PSIPRED (gives 76% prediction accuracy, Rost 11) probabilities vary smoothly over the protein sequence. Hence, we were made to believe that smoothening the data might improve the prediction accuracy. The underlying figure shows some examples of how the probabilities look before and after smoothening.

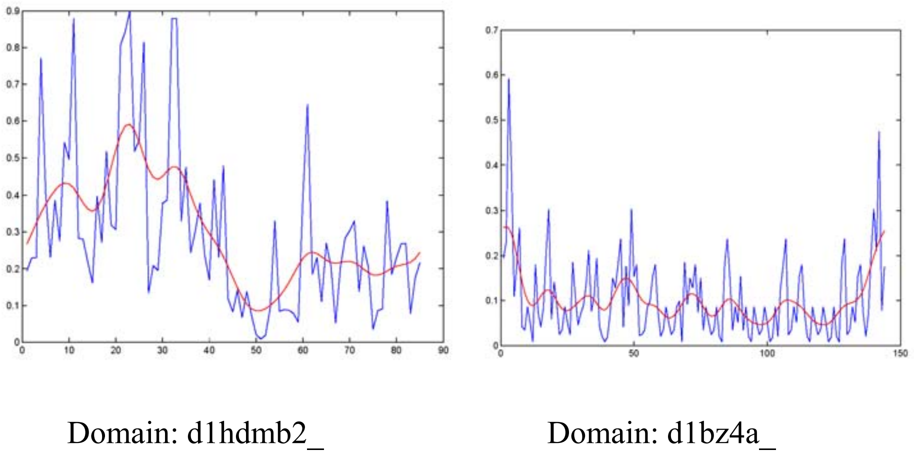


Figure 6: Variation of probability of occurrence of strand with residue number (amino acid number). The blue line depicts the SOM probabilities, while the red line shows the SOM probability smoothened using a window of three and repeated five times.

We carried out experiments varying the window and the ‘number of repeats’ parameter. The test percent correct prediction was calculated for each case. The results are given in Appendix I. The best result obtained was for a window of 1 repeated 3 times. It improved the test fitness structure (prediction accuracy) to 63.525%., which meant an improvement of 3.58% upon the initial SOM data.

Further Work

- There is scope for further studying the effect of changing dimensionality and size of the SOM on the prediction accuracy.
- We believe that there is scope for further preprocessing of SOM data before it is sent to GP. Novel ideas in this direction may be developed. The problem is to somehow classify the wrongly predicted residues. Then use innovative methods to repair these errors, e.g., maybe a way to combine the SOM data from two maps of different dimensionality (or other parameters) for better prediction. (a very vague example, just a direction of thought)
- We did some initial experiments and tried to correlate pairs of windows (of size 5) containing SOM positions over the amino acid sequence of a given domain and extract information. We believed that the maximum correlation a window would have in a given protein would give some information about its secondary structure. Due to weak preliminary results and lack of time, this approach couldn’t be fully explored.

Final Result: The SOM data gives a prediction accuracy of approx. 59.9% and can be improved to 63.5% using smoothing. We have three sets of available data, SOM places, SOM probabilities and Smoothened SOM probabilities.

Using GP to process the SOM data

We used the PerlGP system (http://www.perlgp.org, [5]) to evolve perl programs. It was decided to use PDL in the evolved code to process the SOM data. PDL [website reference] is a library for perl, which has similar syntax and goals to that of MATLAB. It allows easy and flexible array/matrix operations. Since our input data (SOM location for each residue in the protein sequence) to the GP was a matrix of dimensions (length(protein) X dimensions of map), and we required the same operation to be carried out on each residue (row) of the protein sequence (matrix), PDL was the appropriate choice.

We wanted to evolve a code, which could map the input SOM location matrix into probabilities of occurrence of three secondary structural elements (the three columns of the output matrix). Using this information, we would predict the secondary structure for each residue using the highest valued column. Hence, the input and output data for the evolved code is a matrix of same size i.e. (length(protein) X dimensions of map).

The GP used two sets of data to evaluate the prediction accuracy. One set was used to guide the evolution, while the second test was a validation test. It saw the effect on the evolved program on a fresh (independent, not used for evolution) set of data. It was the test set, which gave the right idea about both the prediction accuracy and GP efficiency. The test and training set contained 200 domains each for this experiment.

Approach 1: Since we believed that each SOM position mapped to a unique secondary structure, we allowed only horizontal operations on the input matrix, so that the same position in the SOM always maps to the same structural element. We used addition, multiplication, division, equality, and etc. operators on the matrix, individual columns of matrix, numbers, etc. as the transformation operators for the GP. A copy of the Grammar used for evolving the code is included in Appendix II. Apart from this approach, we used two other approaches to grammar and varied parameters to obtain the best performance. The highest prediction percentage we could obtain was 53.4% when we ran the GP for 5 hours. We didn't pursue this approach further, since statistical tests already gave prediction accuracy as high as 59.5%. However, there is always a chance that GP could do better in more hours.

Approach 2: In the second approach, we allowed vertical operations on the input matrix, enabling GP to avail information from the SOM locations of neighboring amino acids to predict the secondary structure of the given amino acid. The same grammar could be used; only a few additions were made to include vertical operations. The best prediction accuracy 59.652% after running the GP for 120hrs. One can see that the GP was able to reach the statistical percentage correctness, however the statistical tests used no vertical information.

We also used the SOM probabilities and smoothened SOM probabilities as input to the GP. Using these data sets, we had already achieved a prediction accuracy of 59% and 63% respectively. We wanted to use the GP to improve the accuracy further.

Using the SOM probabilities for GP (the grammar remaining essentially the same) and running it for 48hrs, we got an improvement of 3.108% over the statistical data. However, the smoothened SOM probabilities already gave us a prediction accuracy improvement of 3.58%.

We are still running GP for programs including both smoothened and normal SOM probabilities. It has already shown an improvement of 4.48% over the statistical prediction of 59.5% (in 45 hours). We hope we will get better results in more hours.

Further Work

- We believe that the variation in the grammar of the evolved code could bring considerable change in the GP performance. One could add for loops, if else commands, or try other novel grammar approaches.

Improving PSIPRED Prediction using Genetic Programming

PSIPRED (3) is one of the most reliable methods for predicting protein secondary structure giving an accuracy of 76%. PSIPRED data is available through the internet (http://bioinf.cs.ucl.ac.uk/psiform.html). The PSIPRED output contains the probability of whether a residue in a protein is a helix, coil or a strand and this data is accessible (downloadable) for all proteins given in The Protein Data Bank (PDM). An example of typical PSIPRED data is shown underneath.

Domain: D153L__

```
  1 R C   0.977  0.005  0.002
  2 T C   0.843  0.045  0.113
  3 D C   0.840  0.018  0.135
  4 C C   0.632  0.020  0.309
  5 Y C   0.557  0.010  0.482
  6 G E   0.249  0.006  0.754
  7 N E   0.312  0.004  0.718
  8 V E   0.313  0.016  0.691
  9 N E   0.209  0.034  0.741
 10 R E   0.137  0.012  0.852
 11 I E   0.211  0.013  0.770
 12 D E   0.180  0.009  0.794
….
```

The file is organized as: Residue Number, Amino Acid Name, Predicted Secondary Structure, and Probabilities of Coil, Helix and Strand.

Figure 7: Typical PSIPRED output

How these probabilities can be improved for better secondary structure prediction is still a challenge. Human-developed (based on intuition and analysis) methods to improve this probability for better results have failed. In the present project, we used Genetic Programming to improve the PSIPRED probabilities to give higher prediction accuracy.

My Work

- Use and adapt the existing PerlGP libraries (created in the group) for the PSIPRED project.
- Discuss and implement innovative methods to improve the GP search.

Tools Used

Perl, PDL, PerlGP, Matlab, PDB Resources, etc.

Experiments and Observations

The PerlGP system was used to evolve code for this problem. The input data, i.e. an array containing the probabilities for the three secondary structural elements was a matrix of (length of domain) by 3. The output matrix also had the same dimensions, since it was just an array of improved probabilities.

The input-output data set of this problem is totally compatible with the one discussed in the project discussed before (Using SOM and Genetic Programming to predict Protein Secondary structure). Therefore the same operations and grammar were used here. However there was more thrust on the vertical operations, since we wanted to extensively use information from neighboring residue for mapping the probability for a given residue to a new improved value.

We also used the smoothened PSIPRED probabilities as a part of the input data. This data effectively (in an unbiased manner) gets the information of neighboring residues in a given residue.

We conducted experiments on the PSIPRED data using the as input data, both the normal PSIPRED probabilities and the smoothened probabilities. The following graph shows the effect of variation of window size (times = 5) on the best prediction accuracy (training and testing) the GP could achieve. The fitness values have been averaged over 6 GP runs for 5 hours evaluating 200 proteins.

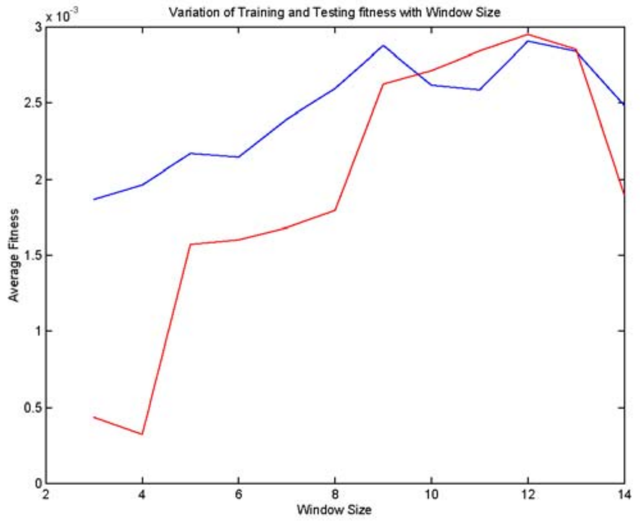


Figure 8: Variation of training and test fitness with Window size

The graph is discrete (bumpy) and needs more time and GP runs to give a clearer picture of the effect of variation of window size. However due to lack of time, it couldn't be done. Still one can clearly see a peak at the window size of 12. For our GP run, we decided to use a low, medium and highly smoothened data for the GP input set.

Presently, we are running a GP for a week which takes as input, a. PSIPRED probabilities b. Smoothened PSIPRED probabilities (Window: 2, Times: 5) c. Smoothened PSIPRED probabilities (Window: 12, Times: 5), d. Smoothened PSIPRED probabilities (Window: 30, Times: 5) e. SOM locations f. Smoothened SOM probabilities (Window: 1, Times: 3) Since the signal (improvement) we are trying to achieve is very weak and needs a large data set to be substantial and recognizable, we use a large set of training and testing data. Our training and testing data set contains 949 and 498 proteins respectively. Results are awaited.

Further Work

- By running GP for longer time and for more number of runs, the effect of window size and number of times smoothing is carried out can be better benchmarked. If some statistical method of benchmarking the effect of window parameter is developed, it will be a break-through for this problem.

- We formed preliminary ideas to implement a difference data in the GP. On observing the PSIPRED probabilities one could see that a sudden change in the probability gave some information. Consider the example: The probability of the residue being helix suddenly drops down, though still remains the largest, PSIPRED predicts it to be a helix, but on observing a handful (we admit!) of domains, we saw that the secondary structure for the specific residue was many-a-time different. Initially we thought of two approaches to bring in this data into

GP, one by just taking the difference of consecutive rows of the PSIPRED data and second by considering the difference percentage, i.e. also dividing by the averaged probability over the two residues. Our initial results with these approaches didn't look good. Secondly, the GP grammar could easily create this data if required. However these ideas might be refined and used in further studies.

- More experiments could be performed with the grammar of the evolved program such as including for loops, if, etc.

- If, by any chance, it is observed that SOM probabilities predict those residues correct, which are predicted wrongly by PSIPRED, a new grammar approach could be to devise a condition to choose one of the two probabilities for the resultant output matrix.

Conclusion

Using SOM and Genetic Programming to predict Protein Secondary structure:

- One can use SOMs to extract information about the secondary structure of the protein. An accuracy upto approx. 63% has been reported in our study.
- GP can do improve the SOM data prediction by around a percent (in our case, running for 60hrs). There is scope that GP could do better in more hours.
- There is scope for further statistical processing of SOM data and we are still working on some ideas related to use of multiple SOM created using different parameters.

Improving PSIPRED Prediction using Genetic Programming

- Our present runs show an improvement of 0.2% on the test set. Also, we see that the GP is still learning. These results are quite exciting. We hope to see an improvement 1% or 2% in more time (maybe a month of running the genetic algorithm).

References

1. Rost, B.: Review: Protein Secondary Structure Prediction Continues to Rise, Journal of Structural Biology, 2001

2. Koza, J: Genetic Programming: On the Programming of Computers by Means of Natural Selection, The MIT Press, 1992

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

4. Dickerson, R. E., Timkovich, R., and Almassy, R. J.: The cytochrome fold and the evolution of bacterial energy metabolism, J. Mol. Biol. 100, 473–491, 1979

5. MacCallum, R. M.: Introducing a Perl Genetic Programming System -- and Can Meta-evolution Solve the Bloat Problem?, (Poster paper) EuroGP 2003

Appendix I

Percentage Correct Probability by smoothening the SOM probabilities.

```
Trained, tested on 200 domains

Smoothening window: 0  Smoothening times: 1 Probability: 5.982596e-001
--
Smoothening window: 0  Smoothening times: 2 Probability: 5.982596e-001
Smoothening window: 0  Smoothening times: 3 Probability: 5.982596e-001
Smoothening window: 0  Smoothening times: 4 Probability: 5.982596e-001
Smoothening window: 0  Smoothening times: 5 Probability: 5.982596e-001
Smoothening window: 1  Smoothening times: 1 Probability: 6.298167e-001
--
Smoothening window: 1  Smoothening times: 2 Probability: 6.349896e-001
Smoothening window: 1  Smoothening times: 3 Probability: 6.352598e-001
Smoothening window: 1  Smoothening times: 4 Probability: 6.335889e-001
Smoothening window: 1  Smoothening times: 5 Probability: 6.329593e-001
Smoothening window: 2  Smoothening times: 1 Probability: 6.320321e-001
--
Smoothening window: 2  Smoothening times: 2 Probability: 6.310021e-001
Smoothening window: 2  Smoothening times: 3 Probability: 6.248429e-001
Smoothening window: 2  Smoothening times: 4 Probability: 6.201926e-001
Smoothening window: 2  Smoothening times: 5 Probability: 6.160115e-001
Smoothening window: 3  Smoothening times: 1 Probability: 6.259857e-001
--
Smoothening window: 3  Smoothening times: 2 Probability: 6.184742e-001
Smoothening window: 3  Smoothening times: 3 Probability: 6.075430e-001
Smoothening window: 3  Smoothening times: 4 Probability: 5.984256e-001
Smoothening window: 3  Smoothening times: 5 Probability: 5.932018e-001
Smoothening window: 4  Smoothening times: 1 Probability: 6.125074e-001
--
Smoothening window: 4  Smoothening times: 2 Probability: 6.037882e-001
Smoothening window: 4  Smoothening times: 3 Probability: 5.913767e-001
Smoothening window: 4  Smoothening times: 4 Probability: 5.825128e-001
Smoothening window: 4  Smoothening times: 5 Probability: 5.759176e-001
Smoothening window: 5  Smoothening times: 1 Probability: 5.926863e-001
--
Smoothening window: 5  Smoothening times: 2 Probability: 5.902588e-001
Smoothening window: 5  Smoothening times: 3 Probability: 5.771551e-001
Smoothening window: 5  Smoothening times: 4 Probability: 5.675426e-001
Smoothening window: 5  Smoothening times: 5 Probability: 5.575844e-001
```

Appendix II

A Grammar file illustrating the type of Grammar used to evolve perl (PDL) programs.

```
package Grammar;

# the PerlGP library is distributed under the GNU General Public
License
# all software based on it must also be distributed under the same
terms
# Due to the limitations of the SDBM module, each string you define
# must be less than 1000 characters

# Functions
%F = ();
# Terminals
%T = ();


#dimension of the SOM

my $dims=3;

$F{ROOT} = [ <<'___',
package Individual;

sub transform {
  my $pdl = shift;
#  my $t = null;
#  my $mem = zeroes($pdl);

  {STMNT}

  return $pdl;
}
___
];


$F{STMNT} = [
    copies(4, '{STMNT}
  {STMNT}'),


    # swap 2 colums
    '$t = $pdl->dice([{NUMS},{NUMS}],X); $t .= $t->rotate({NUMS});
#swap',

    # assign/add/multiply to whole PDL
    '$pdl .= {PDL};',
    '$pdl += {PDL};',
    '$pdl *= {PDL};',

    # assign/add/multiply to a vertical slice (one column)
    '$pdl->slice("{NUMS}") .= {VSLICE};',
    '$pdl->slice("{NUMS}") *= {VSLICE};',
    '$pdl->slice("{NUMS}") %= {NUM};',
];


$F{PDL} = [ copies(4, '$pdl->rotate({NUMS})'),
            'pdl('.join(', ', map '{NUM}', (1 .. $dims)).')',
            '{NUM}',
            '{MATH}({PDL})',
            '({PDL} + {PDL})',
            '({PDL} * {PDL})',
            '{NUMD}'
###            '({PDL} > {PDL})', # maybe also with VSLICE
];

$F{VSLICE} = [ '{NUM}',
               '{MATH}({VSLICE})',
               '({VSLICE} + {VSLICE})',
               '({VSLICE} * {VSLICE})',
               copies(4, '$pdl->slice("{NUMS}")'),
               '({VSLICE} {CMP} {VSLICE})',
];

$T{STMNT} = [ '# nothing' ];


$T{CMP} = [ '<', '>', '<=', '>=', '==', ];

#NUMD might be a 1X1 number or a 3X1 number, it can be threaded to be
added to $pdl

$F{NUMD} = [ 'pdl('.join(', ', map '{NUM}', (1 .. $dims)).')',
              '{NUM}' ];

$F{NUM} = [ copies(6, '{NUMX}*{NUM}'),
          copies(6, 'pdiv({NUM},{NUM})'),
          copies(6, '({NUM} + {NUM})'),
          copies(6, '({NUM} - {NUM})'),
            copies(4, 'abs({NUM})'),
#            copies(1, '({NUM} > 0 ? {NUM} : {NUM})'),
            copies(16, '{NUMX}'),
            copies(16, '{RCONST}'),
            copies(4, '($pdl->sum/$pdl->nelem)'),
            copies(4, '$pdl->nelem'),
            copies(2, '$pdl->dim(0)', '$pdl->dim(1)'),

        ];

#NUMPDL is basically a [1 length] matrix where each row element is
#some function of the elements of the the respective row of $pdl

$T{NUMPDL} = [
    '$pdl->sumover()->dummy(0)',
    '$pdl->maximum()->dummy(0)',
    '$pdl->minimum()->dummy(0)',
    '$pdl->average()->dummy(0)',
    '$pdl->medover()->dummy(0)',
    '$pdl->oddmedover()->dummy(0)',
    ];

$F{SLICE} = [
    '$pdl->slice(\'{NUMS}:{NUMS},:\')'
    ];

$T{MATH} = [
    'sin',
    'cos',
    'tan',
    'exp', # could be dangerous...
    'abs',
    ];

$T{ROOT} = [ 'sub transform { }' ];

$T{NUMS} = [0..($dims-1)];
$T{DIMS} = [$dims];

$T{NUM} = $T{NUMX} = $T{NUMD} = $T{PDL} = $T{VSLICE} = [ -5 .. 5 ];

$T{RCONST} = [ map { sprintf "%.4f", rand(1); } (1 .. 10000) ];

# this helper routine will give you multiple copies of
# something, i.e qw(1 1 1 2 3) is equivalent to (copies(3, 1), 2, 3)
sub copies {
  my ($num, @things) = @_;
  my @result;
  while ($num-->0) {
    push @result, @things;
  }
  return @result;
}
```

1;