Submitted on June 22, 1995 for *Evolutionary Computation: Theory and Applications* edited by Xin Yao.

Automatic Discovery of Protein Motifs Using Genetic Programming

John R. Koza
Computer Science Department
Stanford University
Stanford, California 94305-2140 USA
E-MAIL: Koza@CS.Stanford.Edu
PHONE: 415-941-0336
FAX: 415-941-9430
WWW: http://www-cs-faculty.stanford.edu/~koza/

David Andre
Visiting Scholar
Computer Science Department
Stanford University
Stanford, California 94305-2140 USA
E-MAIL: Andre@flamingo.stanford.edu
PHONE: 415-326-5113

ABSTRACT

Automated methods of machine learning may prove to be useful in discovering biologically meaningful information hidden in the rapidly growing databases of DNA sequences and protein sequences.

Genetic programming is an extension of the genetic algorithm in which a population of computer programs is bred, over a series of generations, in order to solve a problem. Genetic programming is capable of evolving complicated problem-solving expressions of unspecified size and shape. Moreover, when automatically defined functions are added to genetic programming, genetic programming becomes capable of efficiently capturing and exploiting recurring sub-patterns.

This chapter describes how genetic programming with automatically defined functions successfully evolved motifs for detecting the D-E-A-D box family of proteins and for

detecting the manganese superoxide dismutase family. Both motifs were evolved without prespecifying their length. Both evolved motifs employed automatically defined functions to capture the repeated use of common subexpressions. When tested against the SWISS-PROT database of proteins, the two genetically evolved consensus motifs detect the two families either as well, or slightly better than, the comparable human-written motifs found in the PROSITE database.

1. Introduction

The structure and functions of living organisms are primarily determined by proteins (Stryer 1995). Proteins are large polypeptide molecules composed of sequences of up to several thousand amino acid residues. All proteins are composed from the same repertoire of 20 amino acid residues (conventionally denoted by the letters A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, and Y). Subject to only a few minor qualifications, the three-dimensional location of every atom of a protein in a living organism is fully determined by its sequence (its *primary structure*) of amino acid residues (Anfinsen 1973). The protein's three-dimensional structure (its *tertiary structure* or *conformation*), in turn, determines the biological function and activity of the protein within a living organism. Thus, effectively all of the information about the biological function and activity of a protein is contained (albeit deeply hidden) in its primary sequence (i.e., the linear sequence of letters over an alphabet of size 20).

SWISS-PROT is a massive, systematically-collected, periodically-reviewed, annotated database of protein sequences that is maintained by the University of Geneva and the European Molecular Biology Laboratory (Bairoch and Boeckmann 1991). Release 30 (October 1994) of SWISS-PROT, for example, contains 14,147,368 amino acid residues from 40,292 sequences from hundreds of different species. The Human Genome Project and other research efforts in molecular biology are rapidly increasing the number of entries in SWISS-PROT and other databases of protein sequences and genomic DNA sequences. Automated techniques (such as those of machine learning) may prove useful or necessary for analyzing this accumulating data.

Most proteins appear in many different species; however, the primary sequences of the "same" protein in two different species are often not identical. For one thing, the primary sequences of the "same" protein in two different species may differ slightly in length. Moreover, even after using an alignment algorithm (e.g., Smith and Waterman 1981) to align the related sequences of somewhat different lengths, the residues found at a particular aligned position often will still differ. The reason is that only relatively small subsequences of the overall sequence are responsible for the biological function, activity, and structure of the protein. Over millions of years, evolution has substituted dissimilar residues at non-critical positions. Even when one locates the relatively small subsequence of the protein that is responsible for the protein's biological activity, only a few of the residues of the subsequence will prove to be identical (that is, *conserved*) because evolution has also substituted chemically similar residues at these critical positions.

Sometimes, amidst all the differences, it is possible to identify certain high specificity, high sensitivity patterns (called *motifs*, *sites*, *signatures*, or *fingerprints*) in a set of sequences for biologically similar proteins. If a motif is defined well, it will detect a biologically-important common property. The residues in such motifs often prove to be directly responsible for the essential function and activity of the protein.

PROSITE is a database of biologically meaningful patterns found in protein sequences (Bairoch and Bucher 1994). Release 12 (June 1994) of PROSITE, for example, contains 1,029 different motifs. Motifs are entered in the PROSITE database after careful consideration by Amos Bairoch at the University of Geneva and his colleagues. Since the intended primary purpose of PROSITE is to detect families of proteins in computerized databases, a motif is included in PROSITE if it detects most (preferably all) sequences that have a particular biological property (i.e., has few false negatives), while detecting few (preferably zero) unrelated sequences (i.e., has few false positives).

Automated methods of machine learning may be useful in discovering biologically meaningful patterns that are hidden in the rapidly growing databases of genomic and protein

sequences. Unfortunately, almost all existing methods of automated discovery require that the user specify, in advance, the size and shape of the pattern that is to be discovered. However, in practice, the discovery of the size and shape of the pattern may, in fact, be *the problem* (or at least a major part of the problem). Moreover, none of the existing methods of automated discovery have a workable analog of the idea of a reusable, parameterized subroutine or subprogram to capture and exploit repeated occurrences of regularities or sub-patterns of the problem environment.

The problem of discovering biologically meaningful patterns in databases can be rephrased as a search for an unknown-sized task-performing computer program (i.e., a composition of primitive functions and terminals). When the motif discovery problem is so rephrased, genetic programming becomes a candidate for solving this problem. Moreover, if it is also desired to reuse regularities in the problem environment, then genetic programming with automatically defined functions becomes a candidate.

Section 2 of this chapter provides background on protein databases, motifs, the D-E-A-D box family of proteins, and the manganese superoxide dismutase family. Section 3 of this chapter provides background on genetic programming. Section 4 identifies the preparatory steps required to apply genetic programming to the D-E-A-D box problem. Section 5 describes the implementation of genetic programming on a parallel computer. Section 6 presents a genetically consensus evolved motif that is slightly better than the human-written motif found in the PROSITE database for detecting the D-E-A-D box family of proteins. Section 7 presents a genetically evolved consensus motif for detecting the manganese superoxide dismutase family that is as good as the human-written motif found in the PROSITE database. Section 8 states the conclusion.

2. Background on Motifs and Proteins

The D-E-A-D Box Family of Proteins and the Manganese Superoxide Dismutase Family of Proteins will be used to illustrate how genetic programming may be applied to the problem of discovering motifs in protein sequences.

2.1 The D-E-A-D Box Family of Proteins

In the "Birth of the D-E-A-D box," Linder et. al. (1989) described a family of proteins (called *helicases*) involved in the unwinding of the double helix of the DNA molecule during the replication of DNA (Chang, Arenas, and Abelson 1990; and Dorer, Christensen, and Johnson 1990; Hodgman 1988). This family of proteins gets its name from the fact that the amino acid residues D (aspartic acid), E (glutamic acid), A (alanine), and D appear, in that order, at the core of one of its biologically critical subsequences. There are 34 proteins from this family among the 40,292 proteins appearing in Release 30 of SWISS-PROT. Proteins of this family can be detected effectively (but not perfectly) by the following motif of length nine (called ATP_HELICASE_1) that was included by Amos Bairoch at the University of Geneva in the PROSITE database:

`[LIVM]-[LIVM]-D-E-A-D-X-[LIVM]-[LIVM].`

In interpreting this expression, the first pair of square brackets indicates that the first residue of the nine is to be chosen from the set consisting of the amino acid residues L, I, V, and M. The second pair of square brackets indicates that the second residue is chosen (independently from the first) from the same set of four possibilities. Then, the third, fourth, fifth, and sixth residues must be D, E, A, and D, respectively. The X in the motif indicates that the seventh residue can be any of the 20 possible amino acid residues. The eighth and ninth residues are chosen from the same set of four, namely L, I, V, and M.

D and E are negatively charged and hence hydrophilic (water-loving) at normal pH values. A is small, uncharged, hydrophobic (water-hating). L (leucine), I (isoleucine), V (valine), or M (methionine) are moderately-sized, uncharged, and hydrophobic. Thus, ignoring the X, this motif calls for three hydrophilic residues accompanied, on each side, by two moderately-sized hydrophobic residues.

The above PROSITE expression detects any of $4^4 \infty 20 = 5{,}120$ different possible sequences of length nine (out of approximately $5 \infty 10^{11}$ possible sequences of length nine). When SWISS-PROT is searched using the above PROSITE motif, there are 34 true positives, 14,147,333 true

negatives (among the 40,292 proteins), 1 false positive, and 0 false negatives. This corresponds to a correlation coefficient (Matthews 1975), *C*, of 0.99.

Table 1 shows six of the 34 proteins containing the D-E-A-D box motif in the SWISS-PROT database. The table shows the position of the start of the D-E-A-D box motif in its second column. The third column shows the three amino acid residues in the primary sequence before the onset of the motif, the nine residues (in boldface) of the D-E-A-D box itself, and the five residues following the D-E-A-D box.

Table 1 Six examples of D-E-A-D box motif.

Protein	Start	Subsequence
Human Putative ATP Dependent RNA Helicase P54	244	QMI**VLDEADKLL**SQDFV
Rabbit Eukaryotic Initiation Factor 4A	168	KMF**VLDEADEML**SRGFK
Fruit Fly Vasa Protein	397	RFV**VLDEADRML**DMGFS
C. Elegans Putative ATP-Dependent RNA Helicase	192	KFL**IMDEADRIL**NMDFE
E. Coli ATP-Dependent RNA Helicase	155	ETL**ILDEADRML**DMGFA
Fruit Fly Putative ATP-Dependent RNA Helicase	303	KFL**VIDEADRIM**DAVFQ

The number of PROSITE expressions (composed of disjunctions such as shown above) covering exactly nine positions is $(2^{20})^9 \sim 10^{54}$. Since the length of an expression that is capable of detecting a particular family of proteins is, in actual practice, not known in advance, the search space of the motif discovery problem is considerably larger than 10^{54}.

The question arises as to whether it is possible to use an automated machine learning technique to examine a large set of protein sequences and extract biologically meaningful motifs. Such a technique should, of course, not require advance specification of the length of the motif. When this problem is rephrased as a search for an unknown-sized task-performing computer program (i.e., a composition of primitive functions and terminals), genetic programming becomes a candidate for solving this problem. Moreover, if it is also desired to capture

regularities in the problem environment (e.g., the repeated use of a subexpression such as [LIVM]), then genetic programming with automatically defined functions becomes relevant.

2.2 The Manganese Superoxide Dismutase Family of Proteins

The oxygen radicals that are normally produced in living cells have been implicated in many degenerative processes, including cancer and aging. Proteins belonging to the manganese superoxide dismutase family prevent oxidative damage to DNA and other molecules by catalyzing the conversion of these toxic superoxide radicals to oxygen and hydrogen peroxide (Ludwig et al. 1991; Stoddard, Ringe, and Petsko 1990; Bannister, Bannister, and Rotilio 1987). The four ligands of the manganese atom are conserved in all the known sequences of the manganese superoxide dismutase family. Amos Bairoch selected a short conserved region that includes two of the four ligands, namely one D (aspartic acid) and one nearby H (histidine), to create the following motif of length eight (called SOD_MN) for detecting proteins belonging to this family:

```
D-X-W-E-H-[STA]-[FY][FY].
```

For example, in human manganese superoxide dismutase (whose length is 198), the above motif correctly identifies the protein as belonging to this family because residues 159 to 166 of this protein are

```
DVWEHAYY.
```

When it is tested against all of SWISS-PROT, the above motif scored 40 true positives, 14,147,328 true negatives, and 0 false positive and 0 false negatives (for a correlation of 1.00).

The Protein Data Bank (PDB), maintained by the Brookhaven National Laboratory in Upton, New York (Bernstein et al. 1977), is the worldwide computerized repository of the three-dimensional coordinates of the atomic structure of proteins. Proteins from the PDB can be interactively displayed by making a three-dimensional *kinemage* of the protein using the PREKIN software and viewing the kinemage with the MAGE software (Richardson and Richardson 1992). Figure 1 shows residues 159 to 166 of human manganese superoxide

dismutase (1ABM in the Protein Data Bank) as well as the histidines at positions 26 and 74 of the protein sequence. The manganese is ligated by Asp 159, His 163, His 26, and His 74.

3. Background on Genetic Programming

John Holland's pioneering *Adaptation in Natural and Artificial Systems* described how the evolutionary process in nature can be applied to solving problems using what is now called the *genetic algorithm* (Holland 1975). Additional information on recent work in the field of genetic algorithms can be found in Goldberg (1989); Davis (1987, 1991); Michalewicz (1992); Eshelman (1995); Whitley (1992); Maenner and Manderick (1992); Schaffer and Whitley (1992); Albrecht, Reeves, and Steele (1993); Stender (1993); Buckles and Petry (1992); Stender, Hillebrand, and Kingdon (1994); Forrest (1991); Bauer (1994); and Davidor (1990).

Genetic programming is an extension of the genetic algorithm in which the genetic population consists of computer programs. Genetic programming starts with a primordial ooze of randomly generated computer programs composed of available primitive programmatic ingredients. The population of programs is then bred over many generations in a domain-independent way using the Darwinian principle of survival of the fittest and an analog of the naturally-occurring genetic operation of crossover (sexual recombination). The crossover operation is designed to create syntactically valid offspring programs from parents that are probabilistically selected based on their fitness at solving the problem at hand. Genetic programming combines the expressive high-level symbolic representations of computer programs with the near-optimal efficiency of learning of the genetic algorithm.

Genetic Programming: On the Programming of Computers by Means of Natural Selection (Koza 1992) provides evidence that genetic programming can solve, or approximately solve, a variety of problems from a variety of fields. These problems include benchmark problems from machine learning and artificial intelligence involving control, robotics, optimization, game playing, symbolic regression, system identification, empirical discovery, and concept learning. Recent work on genetic programming is found in Kinnear 1994 and Angeline and Kinnear 1996.

A videotape visualization of a number of applications of genetic programming can be found in Koza and Rice (1992) and Koza (1994).

The sequence of work-performing steps of the to-be-evolved programs are not specified in advance by the user. Instead, the sequence of steps is evolved as a result of the competitive and selective pressures of the evolutionary process and the recombinative role of crossover. Specifically, genetic programming is a domain-independent method that genetically breeds populations of computer programs to solve problems by executing the following three steps:

(1) Generate an initial population of random computer programs composed of the primitive functions and terminals of the problem.
(2) Iteratively perform generations consisting of the following sub-steps until the termination criterion of the problem is satisfied:
    (a) Execute each program in the population and determine how fit it is at solving the problem.
    (b) Create a new generation of the population of programs by applying the following two primary operations to program(s) that are selected from the population with a probability based on fitness (i.e., the fitter the program, the more likely it is to be selected).
        (i) *Reproduction*: Copy a selected program to the new population.
        (ii) *Crossover*: Create two new offspring programs for the new population by genetically recombining randomly chosen parts of two selected programs.
(3) The single best computer program produced anytime during the run is typically designated as the result of the run. This result may be a solution (or approximate solution) to the problem.

3.1 Automatically Defined Functions

When humans write programs, they use subroutines to exploit, *by reuse*, the regularities, symmetries, homogeneities, similarities, patterns, and modularities of problem environments. *Genetic Programming II: Automatic Discovery of Reusable Programs* (Koza 1994a) extends genetic programming to evolve multi-part programs consisting of a main program and one or more reusable hierarchically-callable subprograms.

An *automatically defined function* (*ADF*) is a function (i.e., subroutine, `DEFUN`, procedure, module) that is dynamically evolved during a run of genetic programming and that may be called by a calling program (or subprogram) that is simultaneously being evolved.

When automatically defined functions are being used, a program in the population consists of one (or more) *reusable* function-defining branches along with a main result-producing branch.

As a run progresses, genetic programming evolves different main programs in the result-producing branch, different subprograms (i.e., automatically defined functions) in the function-defining branches, and different hierarchical references among the branches. The initial random generation of the population is created so that every individual program in the population has a constrained syntactic structure consisting of a particular architectural arrangement of result-producing branches and function-defining branches. Crossover is then performed in a structure-preserving way so as to preserve the syntactic validity of all offspring by assigning *types* to either entire branches (*branch typing* and *like-branch typing*) or individual points of the overall program (*point typing*).

Genetic programming with automatically defined functions has been shown to be capable of solving numerous problems. More importantly, the evidence so far indicates that, for many problems, genetic programming requires less computational effort (i.e., fewer fitness evaluations to yield a solution with a satisfactorily high probability) with automatically defined functions than without them (provided the difficulty of the problem is above a certain relatively low break-even point). Also, genetic programming usually yields solutions with smaller average overall size with automatically defined functions than without them (again provided that the problem is not too simple). That is, parsimony is an emergent property of automatically defined functions.

Moreover, there is also evidence that genetic programming with automatically defined functions is scalable. For several problems for which a progression of scaled-up versions was studied, the computational effort increased as a function of problem size at a *slower rate* with automatically defined functions than without them. Also, the average size of solutions similarly increased as a function of problem size at a *slower rate* with automatically defined functions than without them. Scalability results from the profitable reuse of hierarchically-callable subprograms within an overall program.

The PROSITE language has no facility for defining and using subroutines; however, the D-E-A-D box motif found in PROSITE database contains a repeatedly used subexpression consisting of four moderately-sized, uncharged, hydrophobic residues (L, I, V, and M). Similarly, the

manganese superoxide dismutase motif contains a repeatedly used subexpression consisting of Y and F. Because of this manifest modularity, genetic programming with automatically defined functions may be appropriate for evolving a motif-detecting program for the D-E-A-D box family of proteins.

When genetic programming with automatically defined functions was applied to the problem of identifying transmembrane domains in proteins, the results were slightly better than previous human-written algorithms (Koza 1994c). Genetic programming has also been used to identify omega loops in proteins (Koza 1994c), to predict whether a residue in a protein sequence is in an α-helix (Handley 1993a, 1994a), to predict the degree to which a protein sequence is exposed to solvent (Handley 1994b), to predict whether or not a nucleic acid sequence is an *E. coli* promoter region (Handley 1995a); to predict whether or not a 60-base DNA sequence contains a centrally-located splice site (Handley 1995b); and to classify a nucleic acid subsequence as being an intron or exon (Handley 1995c).

4. Preparatory Steps for the D-E-A-D Box Family of Proteins

In applying genetic programming with automatically defined functions to a problem, there are six major preparatory steps, namely determining

(1) the set of terminals for each branch,
(2) the set of primitive functions for each branch,
(3) the fitness measure for evaluating how well a program does at solving the problem,
(4) the parameters and variables for controlling the run,
(5) the criterion for terminating a run and designating the result, and
(6) the architecture of the overall multi-part program.

Figure 2 summarizes these six user-supplied inputs to the genetic programming process with automatically defined functions. A run of genetic programming breeds, over a series of generations, a population of computer programs that generally exhibit increasing fitness in grappling with the problem environment. The result of a run of genetic programming is a computer program that may solve, or approximately solve, the user's given problem.

4.1 Terminal Set, Function Set, and Architecture

After analyzing the problem, it seems reasonable that the ingredients of the to-be-evolved computer programs should include 20 zero-argument logical functions capable of interrogating the current position of a protein sequence. For example, (A?) is the zero-argument residue-detecting function that returns 1 if the current residue in the sequence is alanine (A) but otherwise returning 0.

The square brackets used in PROSITE expressions are used to form sets of residues. The two-argument disjunctive function OR can be used to dynamically define disjunctions of the values returned by the 20 residue-detecting functions.

If the overall architecture of the yet-to-be-evolved motif-detecting program uses automatically defined functions to organize amino acid residues into subsets, then the result-producing branch can be used to perform some operations to reach a conclusion. A two-argument conjunctive function AND can then correspond to the dash of the PROSITE language that is used to lengthen the PROSITE expression. This suggests an overall architecture consisting of several (say, two) automatically defined functions and one result-producing branch.

Specifically, the terminal set, $\mathcal{T}_{adf}$, for the two function-defining branches (ADF0 and ADF1) contains the 20 zero-argument residue-detecting functions. That is,

$\mathcal{T}_{adf}$ = {(A?), (C?), (D?), ... , (Y?)}.

The function set, $\mathcal{F}_{adf}$, for the two function-defining branches is

$\mathcal{F}_{adf}$ = {OR}.

The terminal set, $\mathcal{T}_{rpb}$, for the result-producing branch includes the now-defined ADFs, ADF0 and ADF1, and the 20 zero-argument residue-detecting functions.

$\mathcal{T}_{rpb}$ = {ADF0, ADF1, (A?), (C?), (D?),..., (Y?)}.

The function set, $\mathcal{F}_{rpb}$, for the result-producing branch is

$\mathcal{F}_{rpb}$ = {AND}.

Programs consist of two function-defining branches (ADF0 and ADF1) composed of functions from $\mathcal{F}_{adf}$ and terminals from $\mathcal{T}_{adf}$, as well as one result-producing branch composed of functions from $\mathcal{F}_{rpb}$ and terminals from $\mathcal{T}_{rpb}$. Note that we do not prespecify the length of the motif that is

to be evolved. Both the size and content of each branch of the multi-part program is to be evolved by genetic programming.

If the result-producing branch of an overall program returns a logically true value at a particular position in a protein sequence, that position will be identified as the beginning of an occurrence of the motif; otherwise that position will be classified negatively. If a program examines a residue that is beyond the C-terminal (end) of the protein, the position will be classified negatively.

4.2 Fitness Measure

The fitness measure must assign a value as to how well a particular genetically-evolved motif-detecting program predicts whether a particular amino acid residue is the beginning of a D-E-A-D box.

Fitness is measured over a number of trials called *fitness cases*. A set of in-sample fitness cases (i.e., the training set) is used to measure the fitness of programs during the evolutionary process. The fitness cases for this problem are individual amino acid residues of proteins. The single residue indicating the start of the occurrence of the motif is a positive fitness case and all other residues are negative fitness cases.

When a genetically evolved motif-detecting program in the population is tested against a particular fitness case, the outcome can be a true positive, a true negative, a false positive (an overprediction) , or a false negative (and underprediction). The sum of the number of true positives (N_{tp}) the number of true negatives (N_{tn}), the number of false positives (N_{fp}), and the number of false negatives (N_{fn}) equals the total number of fitness cases, N_{fc}:

$$N_{fc} = N_{tp} + N_{tn} + N_{fp} + N_{fn}.$$

The set of positive in-sample fitness cases contained all the residues of 26 of the 34 proteins in SWISS-PROT belonging to the D-E-A-D box family. Because of the rarity of the motif among the 14,147,368 residues in SWISS-PROT and in order to save computer time, we constructed the set of negative in-sample fitness cases by extracting 210 30-residue fragments that did not belong to the D-E-A-D box family, did not contain the D-E-A-D box motif, but did

contain a sizable partial match with the D-E-A-D box motif (such as all X-X-D-E-A-D-X-X-X or V-X-X-EAD-X-X-X that are not in the D-E-A-D box family). When we were done, the in-sample fitness cases consisted of 19,200 amino acid residues from 236 proteins (26 residues being positive instances and l9,174 being negative instances of the D-E-A-D box motif).

Correlation is appropriate as a measure of raw fitness for genetic programming in a two-way classification problem. For a two-way classification problem, *correlation*, *C*, of a genetically evolved motif-detecting program can be computed (Matthews 1975) as

$$C = \frac{N_{tp}N_{tn} - N_{fn}N_{fp}}{\sqrt{(N_{tn} + N_{fn})(N_{tn} + N_{fp})(N_{tp} + N_{fn})(N_{tp} + N_{fp})}}$$

The correlation coefficient indicates how much better a particular predictor is than a random predictor. It may be instructive to view the correlation, *C*, as the cosine of the angle in a space of dimensionality N_{fc} between the zero-mean vector (obtained by subtracting the mean value of all components of the vector from each of its components) of correct answers and the zero-mean vector of predictions made by the predicting program. A correlation *C* of –1.0 indicates that the pair of vectors point in opposite directions in N_{fc}-space (i.e., greatest negative correlation); a correlation of +1.0 indicates coincident vectors (i.e., greatest positive correlation); a correlation *C* of 0.0 indicates orthogonality (i.e., no correlation).

The *in-sample correlation*, *C*, is the correlation computed from the set of in-sample fitness cases. The in-sample correlation *C* lends itself to being the measure of raw fitness for a genetically evolved computer program. Since raw fitness ranges between –1.0 and +1.0 (higher values being better), standardized fitness can then be defined as

$$\frac{1 - C}{2}.$$

Standardized fitness ranges between 0.0 and +1.0, lower values being better and a value of 0 being best. Specifically, a standardized fitness of 0 indicates perfect agreement between the predicting program and the observed reality; a standardized fitness of +1.0 indicates perfect disagreement; a standardized fitness of 0.50 indicates that the predictor is no better than random.

After a motif-detecting program is evolved using the in-sample fitness cases, the question arises as to how well it generalizes to unseen different fitness cases from the same problem environment. A set of out-of-sample fitness cases consisting of 5,605 amino acid residues from 53 proteins (8 residues being positive instances and 5,597 being negative instances) is used to validate the performance of a genetically-evolved motif-detecting program. For reference, when the D-E-A-D box motif found in the PROSITE database is tested against the out-of-sample fitness cases, there are 8 true positives, 5,596 true negatives, 1 false positive, and 0 false negatives (for a correlation of 0.94). When it is tested against all of SWISS-PROT, it scored 34 true positives, 14,147,333 true negatives, and 1 false positive and 0 false negatives (for a correlation of 0.99).

4.3 Parameters

A population size, *M*, of 256,000 was used. The maximum number of generations, *G*, was set at 201 (although every run we made yielded a solution long before generation 200). The maximum size (i.e., number of functions and terminals in the work-performing parts of each branch) was 50 points per branch. Minor parameters were chosen as in Koza 1994a.

4.4 Termination Criterion and Result Designation

The termination criterion for any one run of this problem is emergence of an evolved program with an in-sample correlation of 1.00 on the in-sample fitness cases. That program is designated as the result of the one run.

4.5 Jury Method for Creating a Consensus Motif

Because of the intended purpose of PROSITE (which is purposely oriented toward overfitted descriptions) and because the 1,029 PROSITE motifs partition the existing databases into relatively small subsets, there is a poverty of instances of any given motif in the database. The difficulties of evolving a motif from such an impoverished database can be compensated for by using a jury (Rost and Sander 1993) of at least two evolved results having an in-sample correlation of 1.00. A unanimous jury decision was required in order to classify a position of a

protein sequence as the beginning of the motif. Otherwise, the position was classified negatively.

5. Implementation of Parallel Genetic Programming

The problem (written in ANSI C) was run on a home-built medium-grained parallel computer system consisting of 64 INMOS transputers (housed on Transtech 4 megabyte TRAMs) arranged in a toroidal mesh with a host PC 486 type computer (running Windows). The so-called *distributed genetic algorithm* or *island model* for parallelization (Tanese 1989, Goldberg l989) was used. That is, subpopulations (called *demes* after Sewell Wright 1943) were situated at the processing nodes of the system. Population size was Q = 4,000 at each of the D = 64 demes. The initial random subpopulations were created locally at each processing node. Generations were run asynchronously on each node. After a generation of genetic operations was performed locally on each node, four boatloads, each consisting of B = 8% (the migration rate) of the subpopulation (selected on the basis of fitness) were dispatched to each of the four toroidally adjacent nodes. Details of the parallel implementation of genetic programming can be found in Koza and Andre 1995.

6. Results for the D-E-A-D Box Family of Proteins

In one run, the best motif-detecting program (shown below) among the 256,000 random programs of generation 0 scored 20 true positives, 19,152 true negatives, 22 false positives, and 6 false negatives (for an in-sample correlation of 0.60):

```
(PROGN (DEFUN ADF0 ()
          (VALUES (OR (L?) (N?))))
       (DEFUN ADF1 ()
          (VALUES (OR (R?) (V?))))
       (VALUES (AND (V?) (AND (L?) (D?)))))
```

Note that although we program genetic programming in C, the LISP programming language is used to present the genetically evolved programs since LISP highlights the program's tree structure. This best-of-generation program defines the motif, `V-L-D`, of length three. This

motif admits no alternatives in any of its three positions and ignores the subsets defined by its two automatically defined functions, ADF0 and ADF1.

In subsequent generations, the programs in the population became more complex and their fitness improved apace. The length of the motifs started to increase (sometimes beyond nine). The result-producing branches started to refer to one or both of their automatically defined functions. The automatically defined functions started to be used two or more times.

On generation 42 of one run, the best-of-generation program (shown below) scored 26 true positives, 19,174 true negatives, and 0 false positives and 0 false negatives (for an in-sample correlation of 1.00):

```
(PROGN (DEFUN ADF0 ()
          (VALUES (OR (OR (OR (OR (W?)(M?))(OR (C?)(A?)))(M?))(OR
          (OR (OR (OR (W?)(M?))(OR (M?)(I?)))(L?))(I?))))))
       (DEFUN ADF1 ()
          (VALUES (OR (OR (OR (A?) (E?))(OR (OR (OR (OR
          (A?)(K?))(OR (V?)(N?)))(OR (OR (E?)(R?))(OR (OR
          (A?)(W?))(OR (K?)(Q?))))))(OR (V?)(K?))))(OR (OR
          (I?)(C?))(A?)))))
       (VALUES (AND (AND (AND (AND (AND (AND (AND (ADF1) (ADF0))
       ( AND (D? ) (E? ))) (ADF0)) (D? )) (ADF1)) (ADF0))
       (ADF0)))
```

This best-of-generation program defines the following motif of length nine:

```
[VIAEKNRWQC]-[LIMCAW]-D-E-[LIMCAW]-D-
[RNEKVIAWQC]-[LIMCAW]-[LIMCAW]
```

Note that the common sub-expression [LIM CAW] defined in ADF0 is used a total of four times in the above overall expression. When tested on the 5,605 out-of-sample fitness cases, this expression scored 8 true positives, 5,597 true negatives, and 0 false positives and 0 false negatives (for a correlation of 1.00). When tested against SWISS-PROT, this program scored 34

true positives, 14,147,328 true negatives, and 6 false positives, and 0 false negatives (for a correlation of 0.92).

In another run, the best-of-generation program from generation 64 had an in-sample correlation of 1.00 and defined the following motif of length ten:

```
[FVIAC]-[LIM EQDNRSK]-D-E-[AFVIC]-D-
  [LIMEQDNRSK]-[LIMEQDNRSK] -[LIMEQDNRSK]-[LIMEQDNRSK]
```

Note that the common sub-expression `[LIMEQDNRSK]` defined in `ADF1` is used a total of five times in the above overall expression. When tested on the out-of-sample fitness cases, this program has a correlation of 0.94. When tested against SWISS-PROT, this program scored 34 true positives, 14,147,272 true negatives, and 62 false positives, and 0 false negatives (for a correlation of 0.60).

In other runs, 10 additional evolved programs each had an in-sample correlation of 1.00. These 12 results participated in a jury that created a genetically evolved consensus motif of length 10 (shown below) that scored 26 true positives, 19,174 true negatives, 0 false positives, and 0 false negatives (for an in-sample correlation of 1.00):

```
[IV]-[LIM]-D-E-[AI]-D-[RNEK]-[LIM]-[LIM]-[LIMEQDNRSK]
```

Note that `[LIM]` is used three times in this expression. When tested on the 5,605 out-of-sample fitness cases, this expression scored 8 true positives, 5,597 true negatives, and 0 false positives and 0 false negatives (for a correlation of 1.00). When tested against SWISS-PROT, this program scored 34 true positives, 14,147,334 true negatives, and 0 false positives, and 0 false negatives (for a correlation of 1.00).

Thus, the genetically evolved consensus motif created by the jury scored slightly better than the human-written motif found in the PROSITE database on the problem of detecting the D-E-A-D box family of proteins.

Recalling that the motif found in the PROSITE database for the D-E-A-D box family is

```
[LIVM]-[LIVM]-D-E-A-D-X-[LIVM]-[LIVM],
```

one can see that the genetically evolved consensus motif differs in the following four ways from the motif found in the PROSITE database.

First, position 7 of the motif has a definite character. The X in position 7 of the PROSITE motif is replaced by [RNEK] in the consensus motif. Figure 3 is a scatter diagram relating the Van der Waals volume (Creighton 1993) and the hydrophobicity values (Kyte and Doolittle 1982) of the 20 amino acids. In this figure, R, N, E, and K are located in the same general area (circled in the lower right) indicating that they are all highly hydrophilic and bulky. Second, the [LIVM] in positions 2, 8, and 9 of the PROSITE motif is replaced by the somewhat more precise [LIM] in the consensus motif. As can be seen in the circled area at the top right of the figure, residues I, L, and M have virtually identical volumes whereas V has a different volume.

Third, the [LIVM] in position 1 of the PROSITE motif is replaced by the somewhat more precise [IV] in the consensus motif.

Fourth, the genetically evolved consensus motif specifies that position 10 (beyond the last position specified by the PROSITE motif) contains [LIMRNEKQDS].

7. Results for the Manganese Superoxide Dismutase Family of Proteins

The in-sample fitness cases for the problem of detecting the manganese superoxide dismutase family of proteins consisted of 13,518 amino acid residues from 270 proteins (30 residues being positive instances and 13,488 being negative instances). The out-of-sample fitness cases were constructed using the same approach as described above and consisted of 3,280 residues from 53 proteins (10 residues being positive instances and 3,270 being negative instances).

When the manganese superoxide dismutase motif found in the PROSITE database is tested against the out-of-sample fitness cases, there are 10 true positives, 3,270 true negatives, 0 false positive, and 0 false negatives (for a correlation of 1.00). When it is tested against all of SWISS-PROT, it scores a correlation of 1.00.

Three automatically defined functions were used on this problem.

As before, the genetically evolved motifs participated in a jury that created the following consensus motif of length nine that had an in-sample correlation of 1.00:

D-[VAML]-W-E-H-[SA]-[YFH]-[YFAHS]-[YFADHLIS]

When tested on the 3,280 out-of-sample fitness cases, this expression had an out-of-sample correlation of 1.00. When tested against SWISS-PROT, this program scored 40 true positives, 14,147,328 true negatives, and 0 false positives, and 0 false negatives (for a correlation of 1.00). That is, the genetically evolved consensus motif created by the jury scored as well as the human-written motif found in the PROSITE database on the problem of detecting the manganese superoxide dismutase family of proteins.

The motif found in the PROSITE database for the manganese superoxide dismutase family is D-X-W-E-H-[STA]-[FY][FY].

The genetically evolved consensus motif differs in the following ways from the motif found in the PROSITE database.

First, the X in position 2 of the PROSITE motif is replaced by the set of hydrophobic residues [VAML]. As can be seen from figure 4, position 2 of the motif (i.e., position 160 of the protein) is buried (in contrast to, for example, electrically charged Glu 162 which is exposed to the solvent). Thus, it is reasonable that whatever appears at position 2 should be hydrophobic.

Second, the [STA] in position 6 of the PROSITE motif is replaced by the somewhat more precise [SA] in the consensus motif.

Third, the genetically evolved consensus motif specifies that position 9 (beyond the last position specified by the PROSITE motif) contains [YFADHLIS].

8. Conclusions

Genetic programming was successfully used to create motifs for the D-E-A-D box family of proteins and the manganese superoxide dismutase family. Both motifs were evolved without prespecifying their length. Both evolved motifs employed automatically defined functions to capture the repeated use of a common subexpression. When tested against the SWISS-PROT database of proteins, the two genetically evolved consensus motifs detect the two families either as well, or slightly better than, the comparable human-written motifs found in the PROSITE database.

Bibliography

Albrecht, R. F., Reeves, C. R., and Steele, N. C. 1993. *Artificial Neural Nets and Genetic Algorithms*. Springer-Verlag.

Anfinsen, C. B. 1973. Principles that govern the folding of protein chains. *Science* 81: 223-230.

Angeline, Peter J. and Kinnear, Kenneth E. Jr. (editors). 1996. *Advances in Genetic Programming*. Cambridge, MA: The MIT Press.

Bairoch, A. and Boeckmann, B. 1991. The SWISS-PROT protein sequence data bank: current status. *Nucleic Acids Research* 22(17) 3578–3580.

Bairoch, Amos and Bucher, Philipp. 1994. PROSITE: Recent developments. *Nucleic Acids Research* 22(17) 3583-3589.

Bannister Joe V., Bannister, William.H., and Rotilio Giuseppe. 1987. Aspects of the structure, functions, and applications of superoxide dismutase. *CRC Critical Review of Biochemistry* 22:111-154.

Bauer, R. J., Jr. 1994. *Genetic Algorithms and Investment Strategies*. John Wiley.

Bernstein, F. C., Koetzle, T. F., Williams, G. J. B., Meyer, E.J., Jr., Brice, M. D., Rodgets, J. R., Kennard, O. Shimamouchi, T., and Tasumi, M. 1977. The protein data bank: A computer based archival file for macromolecular structures. *Journal of Molecular Biology*. 112: 535-542.

Buckles Bill P. and Petry, Frederick E. *Genetic Algorithms*. Los Alamitos, CA: The IEEE Computer Society Press. 1992.

Chang, Tien-Hsien, Arenas, Jaime, and Abelson, John. 1990. Identification of five putative yeast RNA helicase genes. *Proceedings of the National Academy of Sciences U.S.A.* 87:1571–1575.

Creighton, T. E. 1993. *Proteins: Structures and Molecular Properties*. Second Edition. W. H. Freeman.

Davidor, Yuval. *Genetic Algorithms and Robotics*. Singapore: World Scientific 1990.

Davis, L. (editor). l987. Genetic Algorithms and Simulated Annealing. Pittman.

Davis, L. 1991. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.

Dorer, Douglas R., Christensen, Alan C., and Johnson, Daniel H. 1990. A novel RNA helicase gene tightly linked to the *Triplo-lethal* locus of *Drosophila*. *Nucleic Acids Research* 18(18): 5489–5495.

Eshelman, L. (editor). 1995. *Genetic Algorithms: Proceedings of the Fifth International Conference*. San Francisco, CA: Morgan Kaufmann.

Forrest, Stephanie. *Parallelism and Programming in Classifier Systems*. London: Pittman 1991.

Goldberg, David E. l989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

Handley, Simon. 1993a. Automated learning of a detector for α-helices in protein sequences via genetic programming. In Forrest, Stephanie (editor). *Proceedings of the Fifth International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann Publishers Inc. Pages 271-278.

Handley, Simon. 1994a. Automated learning of a detector for the cores of α-helices in protein sequences via genetic programming. *Proceedings of the First IEEE Conference on Evolutionary Computation.* IEEE Press. Volume I. Pages 474–479.

Handley, Simon. 1994b. The prediction of the degree of exposure to solvent of amino acid residues via genetic programming. In Altman, Russ, Brutlag, Douglas, Karp, Peter, Lathrop, Richard, and Searls, David (editors). *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*. Menlo Park, CA: AAAI Press. 1994. Pages 156–159.

Handley, Simon. 1995a. Predicting whether or not a nucleic acid sequence is an *E. coli* promoter region using genetic programming. *Proceedings of First International Symposium on Intelligence in Neural and Biological Systems*. Los Alamitos, CA: IEEE Computer Society Press. 122-127.

Handley, Simon. 1995b. Predicting whether or not a 60-base DNA sequence contains a centrally-located splice site using genetic programming. **---In Press---ISMB---ADD INFORMATION---**

Handley, Simon, 1995c. Classifying nucleic acid subsequences as introns or exons using genetic programming. **---In Press---ADD INFORMATION---**

Hodgman, T. C. 1988. A new superfamily of replicative proteins. *Nature* 333:2 2–23 and Errata at Nature 333: 578-578.

Holland, John H. 1975. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Ann Arbor, MI: University of Michigan Press. The second edition is currently available from The MIT Press 1992.

Kinnear, Kenneth E. Jr. (editor). 1994. *Advances in Genetic Programming*. Cambridge, MA: The MIT Press.

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

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

Koza, John R. 1994b. *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: The MIT Press.

Koza, John R. 1994c. Evolution of a computer program for classifying protein segments as transmembrane domains using genetic programming. In Altman, Russ, Brutlag, Douglas, Karp, Peter, Lathrop, Richard, and Searls, David (editors). *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*. Menlo Park, CA: AAAI Press. 1994. Pages 244–252.

Koza, John R. and Andre, David. 1995. *Parallel Genetic Programming on a Network of Transputers*. Stanford University Computer Science Department technical report STAN-CS-TR-95-1542. January 30, 1995.

Koza, John R., and Rice, James P. 1992 .*Genetic Programming: The Movie*. Cambridge, MA: The MIT Press.

Kyte, J. and Doolittle, R. 1982. A simple method for displaying the hydropathic character of proteins. *Journal of Molecular Biology*. 157:105-132.

Linder, P., Lasko, P., Ashburner, M., Leroy, P., Nielsen, P. J., Nishi, J., Schneir, J., and Slonimski, P. P. 1989. Birth of the D-E-A-D box. *Nature* 337: 121–122.

Ludwig, M. I., Metzger, A. I., Pattridge, R. A., and Stallings, W. C. 1991. Manganese superoxide dismutase from *Thermus thermophilus*: A structural model refined at 1.8 Å resolution. *Journal of Molecular Biology* 219: 335-358.

Maenner, R., and Manderick, B. (editors). 1992. *Proceedings of the Second International Conference on Parallel Problem Solving from Nature*. North Holland.

Matthews, B. W. 1975. Comparison of the predicted and observed secondary structure of T4 phage lysozyme. *Biochemica et Biophysica Acta*. 405:442-451.

Michalewicz, Z. 1992. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag.

Richardson, D. C. and Richardson, J. S. 1992. The kinemage: A tool for scientific communication. *Protein Science* 1(1) 3–9.

Rost, B. and Sander, C. 1993. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*. 232: 584–599.

Schaffer, J. D. and Whitley, D. (editors). 1992. *Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks 1992*. Los Alamitos, CA: The IEEE Computer Society Press.

Smith, T. F. and Waterman, M. S. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology*. Volume 147. Pages 195-197.

Stender, J. (editor). 1993. *Parallel Genetic Algorithms*. IOS Publishing.

Stender, J., Hillebrand, and Kingdon, J. (editors). 1994. *Genetic Algorithms in Optimization, Simulation, and Modeling*. Amsterdam: IOS Publishing.

Stoddard, B. I., Ringe, D., and Petsko, G. A. 1990. The structure of iron superoxide dismutase from *Pseudomonas ovalis* complexed with the inhibitor azide. *Protein Engineering* 4: 113–199.

Stryer, Lubert. 1995. *Biochemistry*. W. H. Freeman. Fourth Edition.

Tanese, Reiko. 1989. *Distributed Genetic Algorithm for Function Optimization.* PhD. dissertation. Department of Electrical Engineering and Computer Science. University of Michigan.

Whitley, D. (editor). 1992. *Proceedings of Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Vail, Colorado 1992*. San Mateo, CA: Morgan Kaufmann Publishers Inc.

Wright, Sewall. 1943. Isolation by distance. *Genetics* 28. Page 114–138.

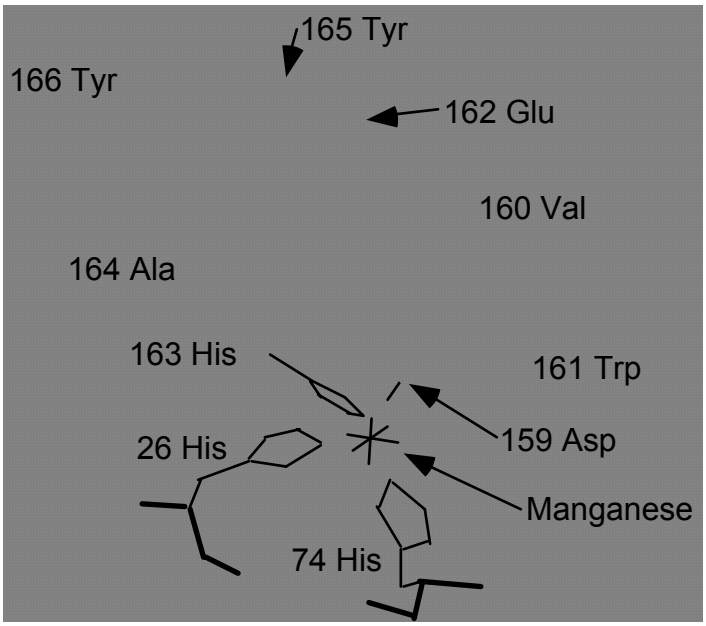

Figure 1 Active site of human manganese superoxide dismutase.

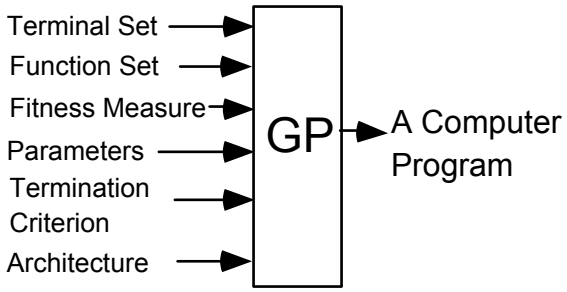

Figure 2 Six major preparatory steps for applying genetic programming with automatically defined functions.

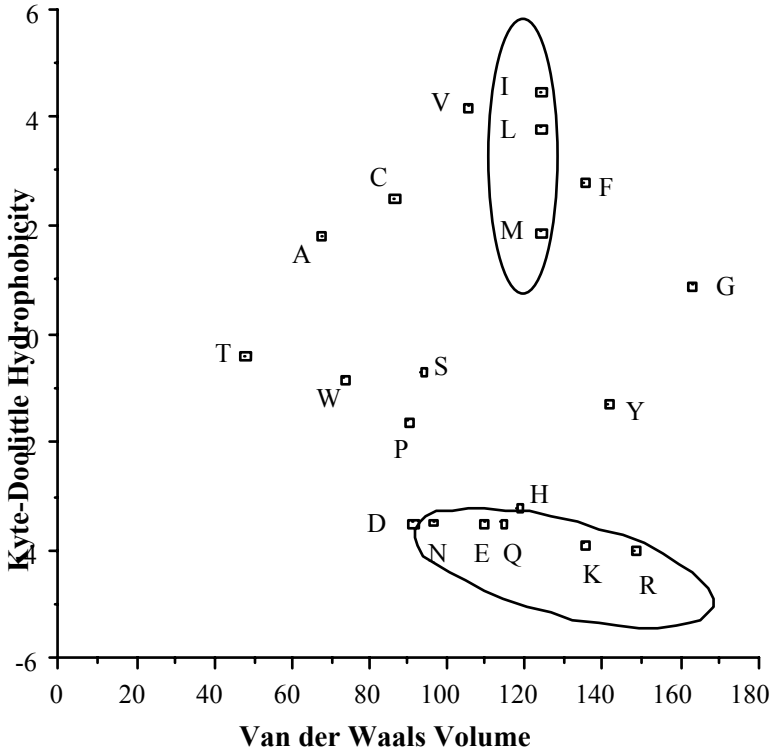

Figure 3 Scatter diagram of hydrophobicity and Van der Waals volume of the 20 amino acid residues.

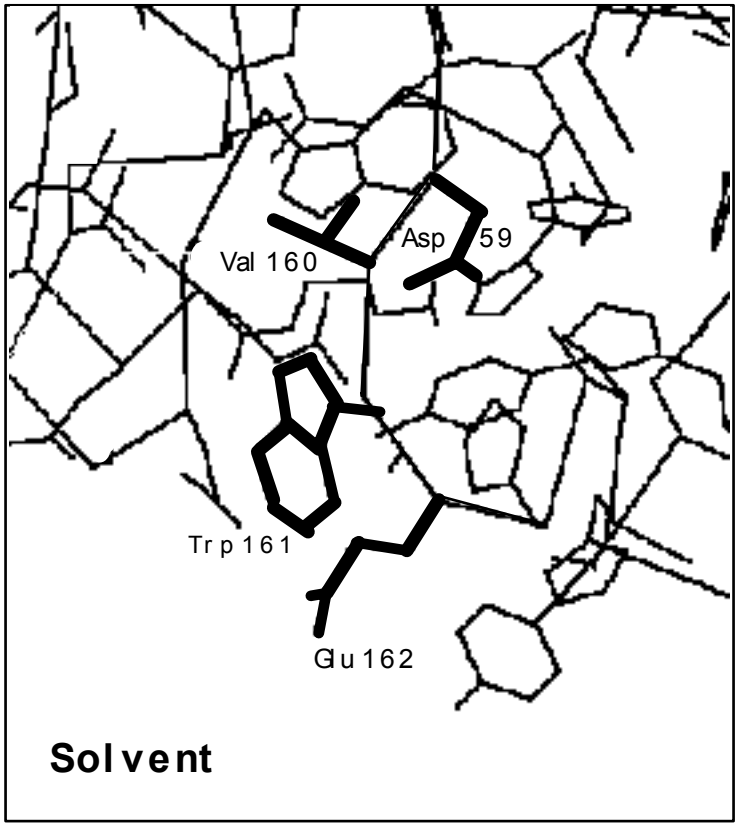

Figure 4 Section of manganese superoxide dismutase showing that position 160 is buried.