Version 2 - June 25, 1996 for *Handbook of Evolutionary Computation*.

Classifying Protein Segments as Transmembrane Domains Using Genetic Programming and Architecture-Altering Operations

John R. Koza

Computer Science Department

Stanford University

258 Gates Building

Stanford, California 94305 USA

E-MAIL: Koza@CS.Stanford.Edu

PHONE: 415-941-0336

FAX: 415-941-9430

http://www-cs-faculty.stanford.edu/~koza/

ABSTRACT

The goal of automatic programming is to create, in an automated way, a computer program that enables a computer to solve a problem. Ideally, an automatic programming system should require that the user pre-specify as little as possible about the problem. In particular, it is desirable that the user not be required to specify the size and shape (i.e., the architecture) of the ultimate solution to the problem before applying the technique.

This paper describes how the biological theory of gene duplication described in Susumu Ohno's provocative book, *Evolution by Means of Gene Duplication*, was brought to bear on a vexatious problem from the domain of automated machine learning in the computer science field. The resulting biologically-motivated approach using six new architecture-altering operations enables genetic programming to automatically discover the size and shape of the solution at the same time as it is evolving a solution to the problem.

Genetic programming with the architecture-altering operations was used to evolve a computer program to classify a given protein segment as being a transmembrane domain or non-transmembrane area of the protein (without biochemical knowledge, such as hydrophobicity values). The best genetically-evolved program achieved an out-of-sample error rate that was better than that reported for other previously reported human-constructed algorithms. This is an instance of an automated machine learning algorithm that is competitive with human performance on a non-trivial problem.

1. Background on Genetic Programming

The goal of automatic programming is to create, in an automated way, a computer program that enables a computer to solve a problem. Ideally, an automatic programming system should require that the user pre-specify as little as possible about the problem. In particular, it is desirable that the user not be required to specify the size and shape (i.e., the architecture) of the ultimate solution to the problem before applying the technique. One of the banes of automated machine learning from the earliest times has been the requirement that the human user predetermine the size and shape of the ultimate solution to his problem (Samuel 1959). I believe that the size and shape of the solution should be part of the *answer* provided by an automated machine learning technique, rather than part of the *question* supplied by the investigator.

John Holland's pioneering *Adaptation in Natural and Artificial Systems* (Holland 1975) described how an analog of the naturally-occurring evolutionary process can be applied to solving problems using what is now called the *genetic algorithm* (described in section B 1.2 of this volume).

The book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Koza 1992) describes an extension of the genetic algorithm in which the genetic population consists of computer programs (that is, compositions of primitive functions, terminals, and possibly automatically defined functions). (See section B 1.5.1 of this volume). In a run of genetic programming in its most basic form, the size and shape of the result-producing

program as well as the sequence of work-performing steps are evolved. A videotape description of genetic programming can be found in Koza and Rice (1992). Recent research activity in genetic programming is described in Kinnear (1994), Angeline and Kinnear (1996), and Koza, Goldberg, Fogel, and Riolo (1996).

I believe that no approach to automated programming is likely to be successful on non-trivial problems unless it provides some hierarchical mechanism to exploit, *by reuse* and parametrization, the regularities, symmetries, homogeneities, similarities, patterns, and modularities inherent in problem environments. Subroutines do this in ordinary computer programs. Accordingly, *Genetic Programming II: Automatic Discovery of Reusable Programs* (Koza 1994a) describes how to evolve multi-part programs consisting of a main program and one or more reusable, parametrized, hierarchically-called subprograms. An *automatically defined function* is a function (i.e., subroutine, procedure, `DEFUN` module) that is dynamically evolved during a run of genetic programming in association with a particular individual program in the population and which may be invoked by a calling program (e.g., a main program) that is simultaneously being evolved. A videotape description of automatically defined functions can be found in Koza (1994b).

When automatically defined functions are being evolved in a run of genetic programming, it becomes necessary to determine the architecture of the overall program to be evolved. The specification of the architecture consists of (a) the number of function-defining branches (automatically defined functions) in the overall program, (b) the number of arguments (if any) possessed by each function-defining branch, and (c) if there is more than one function-defining branch, the nature of the hierarchical references (if any) allowed between the function-defining branches.

The question of how to specify the architecture of the overall program in genetic programming has a parallel in the biological world: how are new structures and behaviors created in living things? This corresponds to the question of how new proteins are created in more complex organisms.

In nature, recombination ordinarily recombines a part of the chromosome of one parent with a corresponding (homologous) part of the second parent's chromosome. A gene duplication is a rare illegitimate recombination event that results in the duplication of a possibly lengthy subsequence of a chromosome. Susumu Ohno's seminal book *Evolution by Gene Duplication* (Ohno 1970) proposed the then-provocative (now accepted) thesis that the creation of new proteins (and hence new structures and behaviors in living things) begins with a gene duplication and that gene duplication is "the major force of evolution." Ohno claimed that simple point mutation and crossover are insufficient to explain major evolutionary changes.

> "...while allelic changes at already existing gene loci suffice for racial differentiation within species as well as for adaptive radiation from an immediate ancestor, they cannot account for large changes in evolution, because large changes are made possible by the acquisition of new gene loci with previously non-existent functions."

The naturally occurring mechanism of gene duplication (and the complementary mechanism of gene deletion) motivated the addition of six new architecture-altering operations to genetic programming (Koza 1994d, 1995). These operations of branch duplication, branch creation, branch deletion, argument duplication, argument creation, and argument deletion enable genetic programming evolve the architecture of a multi-part program containing automatically defined functions (ADFs) during a run of genetic programming. The operations enable the analog of what Ohno described as "the acquisition of new gene loci with previously non-existent functions."

2. Classifying Protein Segments as Transmembrane Domains

This paper considers the problem of deciding whether a given protein segment is a transmembrane domain or non-transmembrane area of the protein.

Proteins are responsible for such a wide variety of biological structures and functions that it can be said that the structure and functions of living organisms are primarily determined by proteins (Stryer 1995). Proteins are polypeptide molecules composed of sequences of amino

acids. There are 20 amino acids (also called residues) in the alphabet of proteins (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). 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.

Membranes play many important roles in living things. A *transmembrane protein* (Yeagle 1993) is embedded in a membrane in such a way that part of the protein is located on one side of the membrane, part is within the membrane, and part is on the opposite side of the membrane. Transmembrane proteins often cross back and forth through the membrane several times and have short loops immersed in the different milieu on each side of the membrane. Understanding the behavior of transmembrane proteins requires identification of the portion(s) of the protein that are actually embedded within the membrane, such portion(s) being called the *transmembrane domain(s)* of the protein. The lengths of the transmembrane domains of a protein are usually different from one another and the lengths of the non-transmembrane area are also usually different from one another.

Algorithms written by biologists for the problem of classifying transmembrane domains in protein sequences are based on biochemical knowledge about hydrophobicity and other properties of membrane-spanning areas of the protein sequence (Kyte-Doolittle 1982, von Heijne 1992, Engelman, Steitz, and Goldman 1986).

This problem provides an opportunity to illustrate automatic discovery of reusable feature detectors, the evolution of the architecture of a multi-part computer program using the architecture-altering operations, the use of state (memory), and the use of iteration-performing steps (in conjunction with information stored in memory) in genetically evolved computer programs. In this section, genetic programming will be given a set of differently-sized protein segments and asked to give the correct classification for each segment.

Genetic programming has previously demonstrated the ability to evolve a classifying program for this task without using any biochemical knowledge (Koza 1994c) when the user specified the architecture of the program to be evolved. The genetically evolved program achieved a better

error rate than the three human-written algorithms that were compared as well as the algorithm developed by Weiss, Cohen, and Indurkhya (1993) using human knowledge along with an element of machine learning.

We now solve this problem again using the architecture-altering operations. The goal is to find a classifying program consisting of an *initially unspecified* number of automatically defined functions, each function possessing an *initially unspecified* number of arguments, consisting of an *initially unspecified* sequence of work-performing operations, an initially *unspecified* sequence of work-performing operations in an iterative calculation, and an initially *unspecified* final result-producing calculation that yields a classification of the protein segment.

The function set for each branch of each program to be evolved consists of four arithmetic operations: (1) a three-argument conditional branching operator, (2) a one-argument setting function, `SETM0`, that sets the settable memory variable, `M0`, to a particular value, and (3) a two-argument numerical-valued disjunctive function.

The terminal set consists of the settable variable, `M0`, floating-point random constants, the length of the protein segment being examined, and 20 zero-argument amino-acid detecting functions that enable the program to examine the protein segment.

Fitness is the correlation between the classification produced by an evolved program and the correct classification. An in-sample (training) set of protein segments is used during the evolutionary process; an out-of-sample (testing) set is used to measure and report the performance of the best program produced by a run.

The population size was 128,000. The problem (written in ANSI C) was run on a medium-grained parallel Parsytec computer system consisting of 64 Power PC 601 processors arranged in a toroidal mesh with a host PC Pentium type computer (running Windows). The Power PC processors communicated by means of one INMOS transputer that was associated with each Power PC processor. The so-called *distributed genetic algorithm* or *island model* for parallelization was used (Goldberg l989). That is, subpopulations (called *demes* after Wright (1943) were situated at the processing nodes of the parallel system. Population size was Q =

2,000 at each of the D = 64 demes for a total population size of 128,000. 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 a given node, four boatloads, each consisting of B = 5% (the migration rate) of the subpopulation (selected on the basis of fitness) were dispatched to each of the four toroidally adjacent nodes. Details of this parallel implementation of genetic programming (and a comparative discussion of migration rates) can be found in Koza and Andre (1995) and Andre and Koza (1996).

On the first run (23 hours) with genetic programming and the architecture-altering operations, a solution was obtained for this problem that exceeded the performance of the three human-written algorithms as well as the algorithm developed by Weiss, Cohen, and Indurkhya (1993).

The best program of generation 28 scores an in-sample correlation of 0.9596, an out-of-sample correlation of 0.9681, an in-sample error rate of 3%, and an out-of-sample error rate of 1.6%. There were 246 fitness cases (half negative and half positive) in the in-sample set of fitness cases and there were 250 fitness cases (again, half negative and half positive) in the out-of-sample set of fitness cases (as described in detail in chapter 18 of Koza 1994).

Figure 1 shows the high-level architecture of the best-of-run program from generation 28. This program has one automatically defined function, `ADF0`, that tests for the amino acid residues phenylalanine (F) and leucine (L), one 36-point iteration-performing branch, `IPB`, and one 169-point result-producing branch, `RPB`.

After genetic programming evolves a solution to a problem, it is often difficult to analyze the program produced by the evolutionary process. However, a number of fortuitous circumstances permitted this particular evolved program to be simplified, by hand, to the following procedure:

(1) Create a sum, S, by adding 4 for each E in the protein segment and 2 for each C, D, G, H, K, N, P, Q, R, S , T, W, or Y (i.e., the 13 residues that are neither E nor A, M, V, I, F or L) in the protein segment.

(2) If

$$\left[\frac{S - 3.1544}{0.9357} \right] < LEN,$$

where LEN is the length of the protein segment, then classify the protein segment as a transmembrane domain; otherwise, classify it as a non-transmembrane area of the protein.

This genetically evolved procedure is simple and works because of the high hydrophobicity of the six amino acid residues A, M, V, I, F and L.

Table 1 shows the out-of-sample error rate for the four previous algorithms for classifying transmembrane domains as well as for three approaches using genetic programming, namely the set-creating version (sections 18.5 through 18.9 of Koza, 1994a), the arithmetic-performing version (sections 18.10 and 18.11 of Koza, 1994a), and the version using the architecture-altering operations as reported herein.

Table 1 Comparison of seven methods.

Method	**Error rate**
von Heijne (1992)	2.8%
Engelman, Steitz, and Goldman (1986)	2.7%
Kyte-Doolittle (1982)	2.5%
Weiss, Cohen, and Indurkhya (1993)	2.5%
GP + Set-creating ADFs in Koza (1994a)	1.6%
GP + Arithmetic-performing ADFs in Koza (1994a)	1.6%
GP + ADFs + Architecture-altering operations (this paper)	1.6%

3. Conclusion

We have shown that it is possible to evolve the architecture of a multi-part program, while concurrently solving the problem, for the problem of classifying protein segments as transmembrane domains or non-transmembrane areas of the protein.

The architecture-altering operations executed during the run of genetic programming determined the existence and eventual number of the automatically defined functions, the number of arguments possessed by each automatically defined function, the size, shape, and sequence of work-performing steps within the automatically defined functions, the size, shape, and sequence of work-performing steps in the iteration-performing branch, and the size, shape, and sequence of work-performing steps in the result-producing branch.

The solution to the problem of classifying transmembrane domains in protein segments is slightly better to the performance of algorithms written by knowledgeable human investigators. This is an instance of an automated machine learning algorithm slightly exceeding human performance on a non-trivial problem.

Acknowledgements

David Andre and Walter Alden Tackett wrote the computer program in ANSI C to implement five of the architecture-altering operations described above.

Bibliography

Andre, David and Koza, John R. 1996. Parallel genetic programming: A scalable implementation using the transputer network architecture. In Angeline, Peter J. and Kinnear, Kenneth E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge, MA: The MIT Press. Chapter 18.

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

Engelman, D., Steitz, T., and Goldman, A. 1986. Identifying nonpolar transbilayer helices in amino acid sequences of membrane proteins. *Annual Review of Biophysics and Biophysiological Chemistry*. Volume 15.

Goldberg, David E. l989a. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley.

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. Pages 244–252.

Koza, John R. 1994d. *Architecture-Altering Operations for Evolving the Architecture of a Multi-Part Program in Genetic Programming*. Stanford University Computer Science Department technical report STAN-CS-TR-94-1528. October 21, 1994.

Koza, John R. 1995. Gene duplication to enable genetic programming to concurrently evolve both the architecture and work-performing steps of a computer program. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. San Francisco, CA: Morgan Kaufmann. Pages 734–740.

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., Goldberg, David E., Fogel, David B., and Riolo, Rick L. (editors). 1996. *Genetic Programming 1996: Proceedings of the First Annual Conference, July 28-31, 1996, Stanford University*. Cambridge, MA: MIT Press.

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

Ohno, Susumu. 1970. *Evolution by Gene Duplication*. New York: Springer-Verlag.

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

Samuel, Arthur L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development.* 3(3): 210–229.

Stryer, Lubert. 1995. *Biochemistry*. New York, NY: W. H. Freeman. Fourth Edition.

von Heijne, G. 1992. Membrane protein structure prediction: Hydrophobicity analysis and the positive-inside rule. *Journal of Molecular Biology*. 225:487–494.

Weiss, S. M., Cohen, D. M., and Indurkhya, N. 1993. Transmembrane segment prediction from protein sequence data. In Hunter, L., Searls, D., and Shavlik, J. (editors). *Proceedings of the First International Conference on Intelligent Systems for Molecular Biology*. Menlo Park, CA: AAAI Press.

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

Yeagle, Philip L. 1993. *The Membranes of Cells*. Second edition. San Diego, CA: Academic Press.

Figure 1 High-level architecture of best-of-run program from generation 28 with one zero-argument automatically defined function, `ADF0`, that tests for certain amino acid residues in the protein segment, one 36-point iteration-performing branch, `IPB0`, and one 169-point result-producing branch, `RPB`.