Artificial Immune System Programming for Symbolic Regression

Colin G. Johnson

Computing Laboratory
University of Kent
Canterbury, Kent
CT2 7NF, England
C.G.Johnson@ukc.ac.uk

Abstract. *Artificial Immune Systems* are computational algorithms which take their inspiration from the way in which natural immune systems learn to respond to attacks on an organism. This paper discusses how such a system can be used as an alternative to genetic algorithms as a way of exploring program-space in a system similar to genetic programming. Some experimental results are given for a symbolic regression problem. The paper ends with a discussion of future directions for the use of artificial immune systems in program induction.

1 Introduction

One way to view genetic programming is that it is the combination of a *representation* of potential solutions and a *search algorithm* which looks through that solution-space in search of an effective solution to the problem at hand.

The early GP work of Koza [15, 16], Cramer [4] and others uses a tree-based representation. In recent years several other representations have been created, for example *grammatical evolution* which uses a bitstring representation which is converted into a program by means of a grammar [18], and *cartesian GP* [17] where a graph of functions and connections is described in a linear chromosome. Several other representations have also been used.

However less interest has been payed to the use of alternative *search* strategies within a GP-like system. A small amount of work has been carried out on the use of simulated annealing rather than genetic algorithms as a search strategy [3, 19, 20], but there appears to be little work beyond this.

Biological systems have proven to be a powerful source of inspiration for the creation of new search algorithms [21]. In recent years algorithms inspired by the adaptive immune system have proven successful on a number of problems. This paper investigates the application of such learning algorithms to systems similar to genetic programming.

2 Artificial Immune Systems

Organisms such as humans need to resist *antigens* (harmful bacteria, viruses et cetera) in order to survive in the world. This resistance is achieved via a multi-layered *immune system*. Initially physical barriers prevent antigens from getting inside the body, then an *innate* immune system destroys antigens which present certain molecular certain patterns to the body, and finally an *adaptive* immune system destroys antigens which are recognized from previous attacks on the body and through a process of maturation where the immune system learns to distinguish between self and non-self.

The learning systems by which organisms learn to identify antigens and distinguish between harmful, non-self cells and those cells which are part of the body or which live commensally within it can be abstracted and implemented as computational learning algorithms. Systems based on such techniques are known as *Artificial Immune Systems* (AIS) [7].

The natural application for such systems is clearly in computer security, and indeed such applications were amongst the earliest applications of AIS [9, 13]. However the learning algorithms at the heart of AIS have proven to be applicable to a wide variety of problems, e.g. data mining, robot control, optimization, control, and scheduling. Surveys are given in [5, 7].

At the core of any immune system is some kind of *pattern recognition*. In the natural immune system this is carried out by a molecular "lock and key" system. Molecules on the surface of antigens (*epitopes*) match up with molecules known as *antibodies* which are found on the surface of *lymphocytes* which are the main cells in the immune system. This recognition then triggers a stream of chemical signals which trigger other components of the immune system to destroy the antigen.

A number of learning and memory mechanisms are important within the immune system. One mechanism is *clonal selection*, which combines the matching system above with a hypermutation mechanism to discover antibodies which accurately match antigens. This both allows rapid learning at the infection site and provides some basic material to feed into immune memories. This mechanism is applied below (section 3) and is described in more detail there.

Another learning mechanism which is important in the immune system is distinguishing between *self* (i.e. the cells in the body) and *non-self* (i.e. cells which are invading the body). This distinction is important for avoiding auto-immune reactions. One important mechanism for this is *negative selection* [2]. Once lymphocytes have been generated they go through a process of *maturation* in the protected environment of the thymus gland. Within the thymus lymphocytes are presented to a variety of self-cells, and those which match are eliminated. As a result the mature cells which are finally released into the main immune system from the thymus are capable of identifying non-self.

Another computationally important aspect of the immune system is *immunological memory*, where the immune system "remembers" previous infections and can therefore attack reinfections or related infections quicker than unknown infections.

There are two main reasons why we may be interested in the application of AIS in GP-like systems. Firstly the search algorithm may work more efficiently than the GA-like algorithms traditionally used, or find more accurate solutions.

Secondly there may be additional features of the immune-inspired algorithms which expand the scope of the GP system.

3 A clonal selection algorithm

This section describes the application of a simple AIS-based system to a GP-like problem. This is based on the *clonal selection* concept. The AIS algorithm used is similar to the CLONALG algorithm by DeCastro et al. [6, 8]. The basic idea of clonal selection is summarized in figure 1.

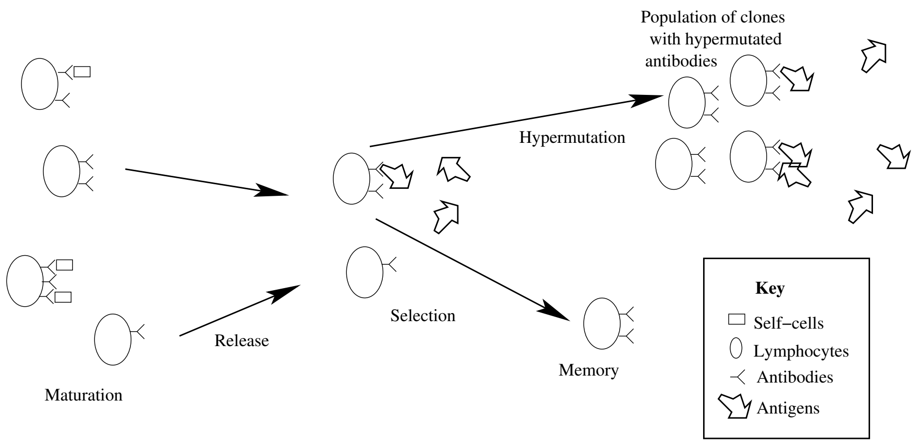

Fig. 1. A summary of clonal selection (after [7])

.

The core function of the clonal selection process is to create lymphocytes which are able to recognize antigens whilst not attacking self cells. There are a number of key stages in this.

Generation The antibodies on the lymphocyte surface are generated via a process of gene assortment which assembles a random fragment of gene segments drawn from a *gene library* and expresses the protein defined by the resultant DNA sequence.

Maturation The lymphocytes undergo a process such as *negative selection* which removes those which react against self-cells.

Release The lymphocytes are released into the body (via the lymphatic system) and diffuse around the body.

Selection If an antigen invades the body the antibodies on the surface of the lymphocytes will attempt to match against the epitopes on the surface of the antigens. Those which match (within a certain tolerance) will continue to the next two stages (in parallel).

Hypermutation The lymphocytes which match will undergo a hypermutation process whereby many copies are generated with random changes in the lymphocytes. Some of these may match the antigens better than the originals. There is some degree of cycling between this stage and the previous stage as successful products of one hypermutation stage are themselves copied and hypermutated.

Memorization Some copies of the successful lymphocytes are stored in immunological memory so that future attacks by the same or similar antigens can be responded to quickly.

This can be abstracted into a *clonal selection algorithm*. In the context of a GP-like system the various components of the problem are translated into the immune metaphor using the scheme given in table 1.

Lymphocytes and antibodies	Programs
Antigens	Test data
Generation of lymphocytes	Random generation of programs
Maturation of lymphocytes	*Not used in this system*
Matching between antibody and antigens	Effectiveness of program on test cases
Selection for hypermutation	Selection of the best-matching programs
Hypermutation of lymphocytes	Hypermutation of programs

Table 1. How clonal selection is used as a metaphor in a GP-like system.

Here is the pseudocode for the algorithm:

```
Generate a set of test data
Generate a random population of lymphocytes
LOOP: Loop a fixed number of timesteps
   Calculate a match between each lymphocyte in the population
      and the test data
   Select the strongest nth of the lymphocytes for hypermutation
   LOOP: through the selected lymphocytes
      Make popSize/n copies of the lymphocyte
      First copy: Replace in the population
      Remaining copies: Apply hypermutation then place in the population
   END OF LOOP
END OF LOOP
```

where n is a parameter which controls the proportion of lymphocytes which are selected to form the basis of the next population.

There are similarities between this algorithm and a standard genetic algorithm. Perhaps the most important reason for placing this work in this framework is that it suggests new directions for future work. Asserting that something "is" a genetic algorithm suggests certain ways in which it might be developed; suggesting that it "is" an AIS provokes different avenues for development. The process of reframing existing algorithms in terms of alternative forms of inspiration could be argued to be a valuable activity because of this.

4 Applying the algorithm to symbolic regression

The above algorithm has been applied to a simple symbolic regression problem (after the problems in [15]).

The programs are represented as lisp parse-trees in the standard manner given in [15]. The function set is $+$, $-$, $\times$, $\div$ (protected for divide by zero), EXP, RLOG (defined in [15]), COS, SIN. The only terminal is a single real variable x, defined on $[-1.0, 1.0]$.

The problem being used in the test below is to find a function to match $x^4 + x^3 + x^2 + x$ on the interval $[-1.0, 1.0]$. The test cases are 100 data points generated randomly in this interval. The match between the lymphocyte and test data is defined as

$$\max\left(0.0, 250.0 - \sum_{\text{test cases}} |f_t(x) - f_p(x)|\right)$$

where f_t is the true function defined on the test data, and f_p is the value of the function given by the program at that point. This turns the error into something which can be used by a maximization procedure.

The initial population is calculated via a standard ramped half-and-half procedure. During each round of the algorithm the top 20% of the population is chosen for hypermutation; one copy of the program is retained and four mutated copies are made and returned to the population. The population size is 1000.

The hypermutation stage is carried out as follows. A random node in the program-tree is chosen and replaced by a randomly generated tree. This new tree is generated by the grow method, and the maximum depth is equal to the depth of the current tree at that position $+1$ (the $+1$ is to allow the trees to grow in depth; otherwise the trees are fixed with a depth at most their initial depth). This has similarities with *headless chicken crossover* $[1, 12, 23]$.

A comparison of the effectiveness of the two search algorithms is given in figures 2 and 3. This gives comparisons between the AIS-based algorithm described above with a standard GP system (similar to the system used for the standard experiments in [15], except that elitism is used). The graphs show the mean results over 20 rounds. The AIS algorithm typically converges to a good solution quicker than the standard GP algorithm. Both algorithms find perfect solutions around half the time (AIS 9/20 trials, GP 12/20 trials). Figure 4 shows the size of trees evolved (where size is measured by counting nodes), both as a per-generation average and as an average of the sizes of the best-performing individuals.

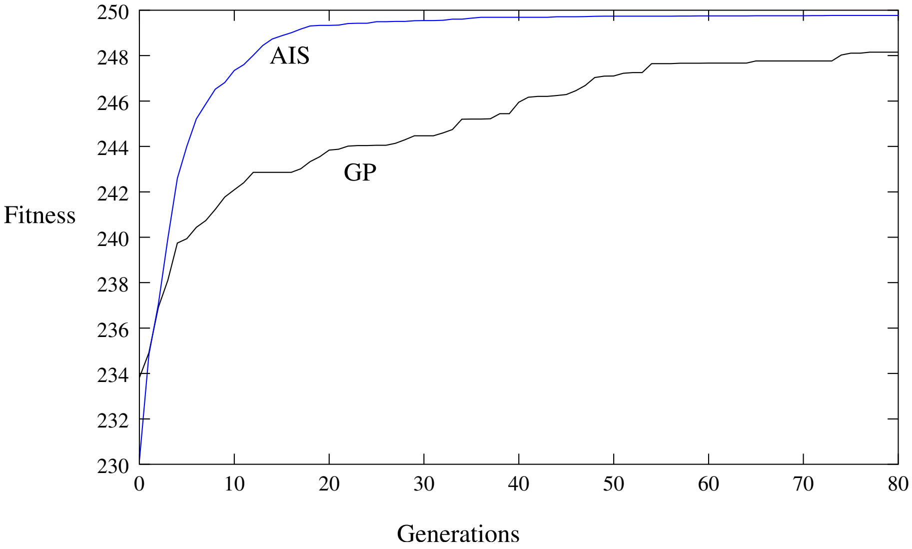

Fig. 2. Comparative fitness curves for AIS and standard GP search algorithms. Best fitness achieved in each generation.

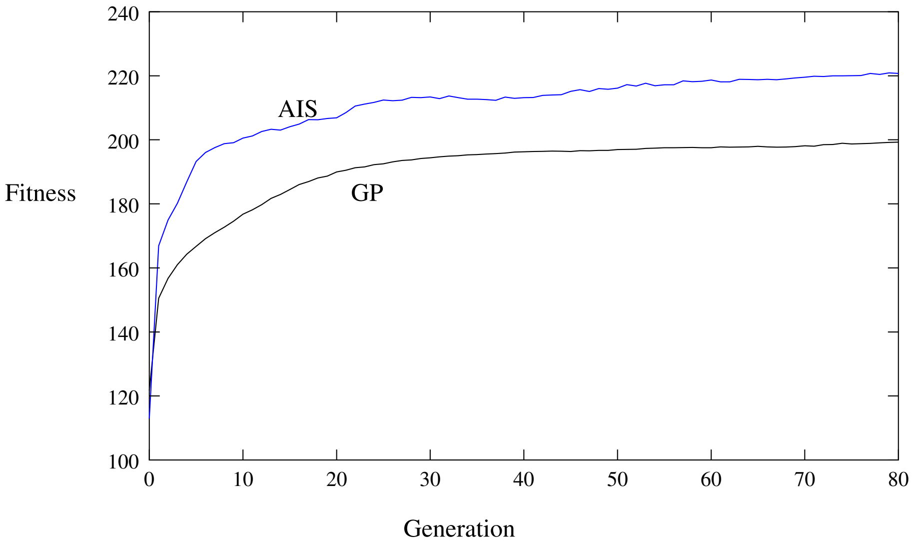

Fig. 3. Comparative fitness curves for AIS and standard GP search algorithms. Average fitness achieved in each generation. Mean over 20 runs.

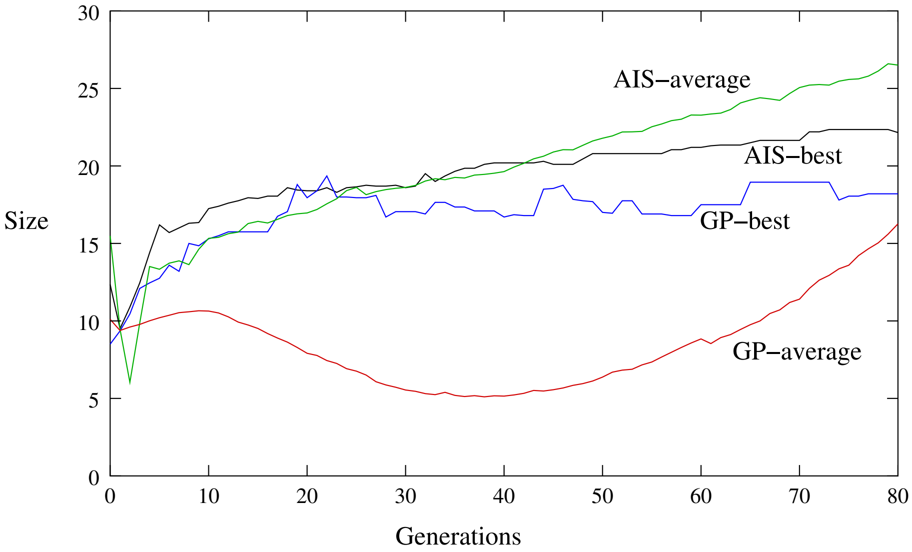

Fig. 4. Comparative tree size for AIS and GP search algorithms.

5 Future work

Clearly the application of these ideas to other test problems is the most immediate piece of future work.

The *immune memory* concept has potential for application in this area. In the natural immune system a proportion of the lymphocytes generated by the hypermutation/selection stages of the algorithm are not used to match with antigens, but are instead added to an immune memory structure. One of the most interesting areas of immunology is concerned with this memory structure, because the immune system can maintain a memory of which lymphocytes have been effective for many years, many times the lifespan of a single lymphocyte. A number of theories exist to explain this, one of which is the *idiotypic network theory* of Jerne [11], which suggests that immune memory is maintained by chains of mutually-stimulating lymphocytes, which select matching replacements for gaps in the network as lymphocytes decay. This theory has proved to be an powerful model in AIS applications [14].

Such memory concepts have potential for being applied to program induction by the accumulation of knowledge about a related set of problems. Instead of each new problem being tackled by a randomly-generated set of lymphocytes, some of the lymphocytes are taken from a memory-network which contains lymphocytes from successful related problems.

Another important mechanism is the evolution of the *gene libraries* from which lymphocytes are created in the first place. In recent work on the application of AIS in scheduling systems, Hart and Ross [10] discuss the learning of libraries of antibody patterns which match sub-schedules in a number of different scheduling problems. This is used to build up a repertoire of "useful patterns" for solving a certain class of scheduling problems.

A final idea is to exploit the way in which the immune system works in a distributed fashion. Real lymphocytes do not always recognize whole antigens at once; different antibodies will recognize different epitopes on the surface of a single antigen, so several lymphocytes may attack a single antigen. Can this be exploited to produce a system where several coordinated mini-programs tackle different aspects of a large problem?

Acknowledgements

Many thanks to Jon Timmis for many discussions on Artificial Immune Systems.

References

1. P.J. Angeline. Subtree crossover: Building block engine or macromutation? In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 9–17. Morgan Kaufman, 1997.
2. M. Ayara, J. Timmis, R. de Lemos, L.N. de Castro, and R. Duncan. Negative selection: How to generate detectors. In J. Timmis and P.J. Bentley, editors, *Proceedings of the First International Conference on Artificial Immune Systems*, pages 89–98. University of Kent, 2002.
3. John A. Clark and Jeremy L. Jacob. Protocols are programs too: the meta-heuristic search for security protocols. *Information and Software Technology*, 43(14):891–904, 2001.
4. Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In John J. Greffenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 183–187. Erlbaum, 1985.
5. Dipankar Dasgupta and Nii Attoh-Okine. Immunity-based systems: a survey. In *Proceedings of the 1997 IEEE International Conference on Systems, Man and Cybernetics*. IEEE Press, 1997.
6. Leandro N. de Castro and Jon Timmis. An artificial immune network for multimodal function optimization. In *Proceedings of the 2002 Congress on Evolutionary Computation*, pages 699–704. IEEE Press, 2002.
7. Leandro N. de Castro and Jon Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, 2002.
8. L.N. de Castro and F.J. Von Zuben. Learning and optimization using the clonal selection principle. *IEEE Transactions on Evolutionary Comptuation*, 6(3):239–251, 2002.
9. Patrik D'haeseleer, Stephanie Forrest, and Paul Helman. An immunological approach to change detection: Algorithms, analysis and implications. In *IEEE Symposium on Security and Privacy*. IEEE Press, 1996.
10. Emma Hart and Peter Ross. The evolution and analysis of a potential antibody library for use in job-shop scheduling. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, pages 185–202. McGraw-Hill, 1999.

11. Niels K. Jerne. Towards a network theory of the immune system. *Annals of Immunology (Institute Pasteur)*, 125C:373–389, 1974.
12. Terry Jones. Crossover, macromutation and population-based search. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, 1995.
13. J.O. Kephart, G.B. Sorkin, and M. Swimmer. An immune system for cyberspace. In *Proceedings of the 1999 IEEE International Conference on Systems, Man and Cybernetics*, pages 879–884, 1997.
14. T. Knight and J. Timmis. AINE: An immunological approach to data mining. In N. Cencone, T. Lin, and X. Wu, editors, *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 297–304. IEEE Press, 2001.
15. John R. Koza. *Genetic Programming : On the Programming of Computers by means of Natural Selection.* Series in Complex Adaptive Systems. MIT Press, 1992.
16. John R. Koza. *Genetic Programming II.* Series in Complex Adaptive Systems. MIT Press, 1994.
17. Julian F. Miller and Peter Thomson. Cartesian genetic programming. In Poli et al. [22], pages 121–132. LNCS 1802.
18. Michael O'Neill and Conor Ryan. Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358, August 2001.
19. Una-May O'Reilly and Franz Oppacher. Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In *Parallel Problem Solving from Nature III, Jerusalem*, pages 397–406. Springer, 1994.
20. John O'Sullivan and Conor Ryan. An investigation into the use of different search strategies with grammatical evolution. In Poli et al. [22], pages 268–277. LNCS 1802.
21. R.C. Paton, H.S. Nwana, M.J.R. Shave, and T.J.M. Bench-Capon. An examination of some metaphorical contexts for biologically motivated computing. *British Journal for the Philosophy of Science*, 45:505–525, 1994.
22. Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian Miller, Peter Nordin, and Terence C. Fogarty, editors. *Proceedings of the 2000 European Conference on Genetic Programming*. Springer, 2000. LNCS 1802.
23. Riccardo Poli and Nicholas Freitag McPhee. Exact GP schema theory for headless chicken crossover and subtree mutation. In *Proceedings of the 2001 Congress on Evolutionary Computation*. IEEE Press, 2001.