The joy of genetic programming

Lizzy the virtual lizard was genetically programmed to enjoy those staples of life, food and sex. Maybe other computers could be too. Nick Beard explores the possibilities.

Computers are almost everywhere. Few areas of human endeavour have not been touched by them. From the start, the problem of programming them has been obvious.

Computers are hard to program, especially if you want to quickly generate working software. So, one of the interesting and most difficult problems in computer science becomes: how can computers be made to solve problems without explicitly being programmed? In other words (in fact, words attributed to Arnold Samuel in the 1950s), how can computers be made to do exactly what is needed to be done, without being told exactly how to do it?

It is an attractive notion, quite in keeping with smarter approaches to people management (like management by objective, or, telling someone what is to be done, not the detail of how to do it). This is the goal of genetic programming. It is an attempt to program computers — with explicit programs — without having to sit and write code.

One for the pot

The bulk of approaches to date have been to start with a lump of computing capability which is not expected to serve as a solution in the style of a conventional program, such as a neural net, a Boltzmann machine, or a small package of squid cerebellum. This is then embedded in a pot of standard code [C, machine, or whatever type] to handle the links to the rest of the world: i.e. you with the fingers on the keyboard or eyes on the cash machine. (Actually, not much work has been done with squid cerebellum as an add-on to Excel, but if Bill would like to fund a research project up there in Seattle, I could be open to offers...)

Problem: humans do not generally regard these 'non-standard lumps' of computing capability (neural net, squid brain, hologram, whatever) as having the flexibility or wide applicability for general problem solving. Nor do people find such constructs especially useful as a 'reasoning substrate', a way of thinking about how they might solve a problem. Existing methods of machine learning, artificial intelligence, self organising systems, neural networks and the like do not seek solutions in the form of computer programs. In fact, they tend to deal in structures which bear little or no resemblance to programs, such as a neural network's weight vectors, a genetic algorithm's chromosome string, or some sort of formal grammar, production rule, frame, concept cluster or whatever.

These things are all useful — in the right application domain. Human programmers, however, do not usually find them to be sufficiently accessible or usable structures for generic programming, as evidenced by the fact that computers are rarely programmed in the language of weight vectors, formal grammars, schemata, polynomial coefficients or chromosome strings...

The tough conclusion is that if we are keen to get computers to do useful stuff without having to be explicitly programmed, we need automated techniques which produce structures known as... computer programs! Moreover, this is not the place for quibbles about the definition of a computer program. The fact that Turing and Co showed that all computers are effectively equivalent is not relevant here: they are only equivalent in terms of the results of the computations they perform. They are not equivalent in terms of usability by C++ code-merchants.

Sex, lies and codes

Genetic programming is an attempt to harness the obvious power of evolution, to help design and build programs. In crude terms, take a bunch of programs, test them for performance, inter-breed them, allow them to mutate a bit and generally fight among themselves, test them again, re-mutate them, re-breed them. Keep testing the results and cycling through this sequence of breed, mutate and test until one program emerges — survival of the fittest and all that — as clearly superior to the others. This computational core has been used before, but not usually to produce programs which are recognisable and understandable as programs. Genetic algorithms, first discussed on these pages in the first ever Frontiers back in July 1990, are clearly related, ancestrally, to genetic programming.

Computational sex

To solve any problem using genetic algorithms, a 'population' of possible solutions is generated randomly. These 'guesses' are coded as strings of symbols; letters and numbers strung together to code different guesses. These are analogous to chromosomes in natural selection, and are 'processed' by a bit of computational sex: crossover and inversion (terms which refer to chromosomal operations, rather than any gender-change/flexibility or other proclivities). Genetic algorithms need mutation as well as sex, because some of the best 'genes' — portions of the chromosome representing best bits of a potential solution — can be lost during mating and reselection. Like the anecdote about George Bernard Shaw: Mrs Campbell coyly hinted that they should consider having a child together, since with his brains and her looks, what a wonderful child they might have. Shaw replied: the child might have her brains and his looks. Sex does not always bring out the best in people (that is purely a computational statement...).

In Frontiers #1, genetic algorithms

were discussed there as an approach to solving non-linear optimisation problems, such as designing neural networks. Here, with genetic programming, the aim is to produce good code. John Koza, author of a recently published volume on the topic from the MIT Press, succinctly states the attraction of the evolutionary paradigm. Fitness begets structure. Evolution, driven by natural selection of the fittest, generally begets living, physical structure. Programs of course are computational structures, the most complex artifacts of humankind, potentially as complex as many living things. As computational structures, programs are made up of data structures and problem/solution structures (or algorithms). Why not evolve them? This is Koza's approach.

One of Koza's first points is that there are many problems which can be valuably represented as program induction. Programs can often be seen as black boxes. You cannot — need not — see the insides; you simply see a number of input channels (labelled funnels on the top of the box) and some output channels. The box takes a particular input and produces an output. You do not — need not — know how.

Programming is setting up the black box's innards. Induction is an approach to doing this from examples of input-output relationships. There are many examples: establishing a mathematical function which returns a particular value in response to particular inputs; a signal processing device, a planning system, a forecasting program.

Crucial here is the idea of search space. Imagine the induction process to be one of looking for the best way to 'fill in the blanks' in a general-purpose formula. Given A as a starting value (input), multiply A by B then subtract C, divide by D, add E and take the Fth root of the result. The answer is the output. The induction problem is finding the right values for B, C, D, E and F. The 'formula' thus derived can also be described as a program, and the space of possible programs is a five-dimensional space defined by the variables B to F. The genetic programming paradigm involves using an evolutionary mechanism to search such a hypothetical 'space' of possible programs to find the one(s) which solve(s) the problem best, or at least well enough.

Find the singer

Searching a multidimensional space is a common problem in computing. Imagine trying to find someone in a multi-storey house, when you can hear them singing. Each room is a separate location, and the 'fitness' of each room is simply measured by the volume of the voice of the person you are looking for. Here, the space being searched is real. At each point, in each room, that is, you make a judgement about the loudness of the singing, and then move in what appears to be the right direction. (Sometimes you will set off in the wrong direction, such as when the noise seems to be coming from the next room when it is in fact coming from upstairs. To get to the next room, you go away from the stairwell and thus find a false maximum: an apparently good location, but not the best that it could be.)

Genetic algorithms (or genetic programming methods) are effective in part because they involve a highly parallel search over many points (rooms in the house) simultaneously. (Precisely how this happens is beyond the scope of this piece, but see discussion of schemata in Koza's book or in Goldberg's brilliant *Genetic Algorithms*, published by Addison Wesley.)

Representation of the problem is important. The 'formula' above (multiply A by B then subtract C, divide... etc) is not a generally, arbitrarily applicable representation. There are many problems (in fact, most) which cannot be represented that way. No combination of values for B to F would solve the problem of how to obtain the maximum value in a string of digits, for example. It only has one input value and no way of comparing numbers. Program induction, then, needs a more sophisticated representation scheme.

Naming names: Darwin machines

Koza describes an approach which involves evolving hierarchically structured programs made up of two classes of object: functions and terminals. Functions are such things as mathematical functions (cos, tan, log etc) or Boolean operators (AND, NOT etc). Consider the hierarchy as an upside-down tree. The functions are the branching points, and the terminals are the leaves: where the branches terminate. Terminals are typically variables (such as, perhaps, inputs, the state of some sensor) or arithmetical or other constants. Programs are selected from the space defined by the selection of functions and terminals, and tested for fitness. This is done by testing their response to some inputs, and the cycle of reproductive-genetics, described above, is begun.

Clearly this is computationally costly. Not only is there the processing involved in handling all the potential programs, but they each have to be run against dummy data and the scores retained as a basis for selection and monitoring of overall best performance. If the approach is to be used more widely, special purpose machines are likely to be needed, with hardware optimised to handling the data structures and processing involved. Such computers have been dubbed Darwin machines. The techniques can be used to grow and evolve specialised electronic circuits.

The story of Lizzy

One example is the Lizzy Circuit, which Hugo de Garis describes in the Springer volume mentioned in the panel below. This field has been called embryological electronics.

Lizzy is a virtual lizard, which does three things: flees predators, eats prey, and mates. Lizzy senses the world as including three types of objects: those which make high, middle and low frequency noises. The rules are simple: predators (low frequency) cause Lizzy to turn and flee, mates (middle frequency) or prey (high frequency) cause Lizzy to turn and move towards the source, until either feeding or mating takes place (a typical West End Saturday night, really). Importantly, Lizzy was not specified in much more detail than that: instead, the circuits were simply evolved. (Much as, over longer periods and to greater complexity, real lizards' 'circuits' were!)

Once again, Nature proves a useful engineering inspiration.

Resource Guide

Genetic Programming
John R Koza, MIT Press
A large and detailed volume which introduces the notion of genetic programming, discusses its wide applicability, presents lots of background material. The style is accessible, though there is a great deal of advanced material here too. An accompanying one hour VHS video tape is also available.

Parallel Problem Solving from Nature
HP Schwefel and R Manner, Springer Verlag
Proceedings of a workshop held in Dortmund in 1990, at which genetic algorithms, genetic programming, and other computational approaches to parallel problem solving inspired by natural systems were discussed. Includes material on Darwin machines (by Hugo de Garis) and an exotic array of computational strategies founded on fluid dynamics, immune networks, neural systems and more.

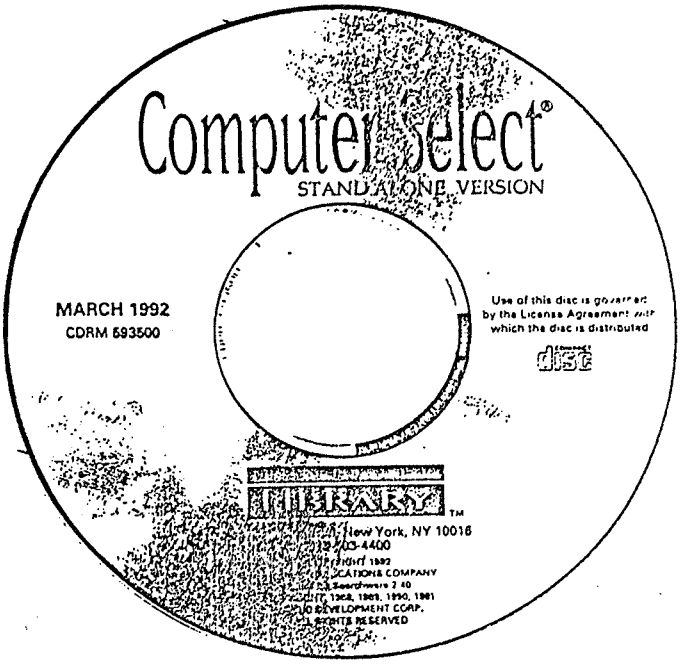
Computer Select – The Answer to Your Search?

Very possibly. Certainly if you are desperately seeking information in computing, Computer Select could help you avoid having to read all the relevant journals and newsletters, should you even succeed in tracking them all down.

Produced on CD-ROM, Computer Select is a full text database containing articles from over 140 computer, technology and business publications from the US and the UK, (obscure titles are usually included!). Subject coverage includes hardware, software, company profiles, and a glossary of terms. Articles from over 40 of the more prominent computer publications such as PC Week and PC Magazine are provided in full, whilst abstracts are provided for most others. Table 1 lists titles meriting full text coverage, and Table 2 lists those for which abstracts only are provided. The database comes with the Lotus BlueFish searchware, which allows you to carry out your own searches on words, phrases, terms or fields. Once identified, documents may be read, copied or printed.

Produced by Internation Software, Computer Select disks are updated on a monthly basis, and require the following environment:

IBM PC/XT, AT, PS/2 or fully compatible computer
640 K RAM
minimum 500K available space on hard disk
DOS version 3.1 or higher
CD-ROM drive card and CD-ROM drive installed,
High Sierra compatible
Microsoft MS-DOS CD-ROM extensions



If all this seems a little too much to hope for on your own desktop, try visiting the UCL Central Computing Service's Library, on the first floor of the Kathleen Lonsdale Building, where Computer Select is available to all members of the Consortium.

Lyn Robinson (User Support, UCL CCS)

Table 1: Full-Text Publications

ACKnowledge, The Window Letter
AI Expert
C Users Journal
Communications of the ACM
The Computer Conference Analysis Newsletter
Computer Language
Computer Letter
Computer Shopper
Computergram International
Computing Canada
Data Based Advisor
Database programming and Design
Datamation
DBMS
Dec Professional
Digital Media
Digital Review
Dr. Dobb's Journal
EDGE, on and about AT & T
EDGE: Work-Group Computing Report
Electronic Computer Glossary
.EXE
Government Computer News
Hewlett Packard Journal
HP Professional
IBM Journal of Research and Development
IBM Systems Journal
Information Industry Bulletin
LAN Computing
LAN Magazine
LAN Technology
LOCALNetter
Lotus
MacUser
MacWeek
Microprocessor Report
Microsoft Systems Journal
MIDRANGE Systems
Multimedia Computing and Presentations
NetWare Advisor and NetWare Advisor Special Report
Newsbytes
The New Science Report on Strategic Computing
OSINetter
Patricia Seybold's Network Monitor
Patricia Seybold's Office Computing Report
Patricia Seybold's Unix in the Office
PC/Computing
P.C. Letter
PC Magazine
PC Sources
PC User
PC Week
Personal Workstation
Release 1.0
The Seybold Report on Desktop Publishing
The Seybold Report on Publishing Systems
Soft*letter
Software Industry Bulletin
Software Magazine
SuperGroup Magazine
Systems Integration
TECH Specialist
Tech Street Journal
Telecom Dictionary
Teleconnect
Token Perspectives
Unique
Unix Review
VAX Professional
Windows Watcher
Wordperfect