Tutorial Topic: Evolutionary Computation

NCSEC'98 Tutorial Session
October 19, 1998
Kasetsart University

Prabhas Chongstitvatana,
Department of Computer Engineering,
Chulalongkorn University.
URL www.cp.eng.chula.ac.th/faculty/pjw
Email prabhas@chula.ac.th

Abstract:
Evolutionary Computation is an approach to computation that emphasize on a general-purpose search algorithm that use principles inspired by population genetics to evolve solutions to problems. Two most well-known methods are Genetic Algorithm (GA) and Genetic Programming (GP). Genetic programming is a machine learning technique derives from genetic algorithms. GA and GP has become increasingly popular in recent years as a method for solving complex search problems in a large number of disciplines. This tutorial will illustrate the basic concept of GA/GP and their current applications. Examples of the actual running system will be given from the research done by the speaker.

Biography of presenter
Prabhas Chongstitvatana earned his first degree in Electrical Engineering from Kasetsart University in 1979. He got PhD from Edinburgh University, UK, in 1992 from the department of Artificial Intelligence. His current research involved in Evolutionary computation where he applied Genetic search method to robot learning problems and logic synthesis. He is interested in hardware evolvable systems.

Most of the text are excerpt from FAQ of comp.ai.genetic

HIERARCHY OF THE FIELD

Natural Computation
- Artificial Life
- Fractal Geometry
- other Complex Systems Sciences

- Computational Intelligence
  - Fuzzy Systems
  - Artificial Neural Networks

  - **Evolutionary Computation**

GLOBAL OPTIMIZATION algorithms

OPTIMIZATION methods: Simulated Annealing (SA), Artificial Neural Networks (ANNs) and the field of Evolutionary Computation (EC).

EC may currently be characterized by the following pathways: Genetic Algorithms (GA), Evolutionary Programming (EP), Evolution Strategies (ES), Classifier Systems (CFS), Genetic Programming (GP), and several other problem solving strategies, that are based upon biological observations, that date back to Charles Darwin's discoveries in the 19th century: the means of natural selection and the survival of the fittest, and theories of evolution. The inspired algorithms are thus termed Evolutionary Algorithms (EA).

What are Evolutionary Algorithms (EAs)?

Evolutionary algorithm is an umbrella term used to describe computer-based problem solving systems which use computational models of some of the known mechanisms of EVOLUTION as key elements in their design and implementation. A variety of evolutionary algorithms have been proposed. The major ones are: GENETIC ALGORITHMs, EVOLUTIONARY PROGRAMMING , EVOLUTION STRATEGIEs , CLASSIFIER SYSTEMs , and GENETIC PROGRAMMING.
They all share a common conceptual base of simulating the evolution of INDIVIDUAL structures via processes of SELECTION, MUTATION, and REPRODUCTION. The processes depend on the perceived PERFORMANCE of the individual structures as defined by an ENVIRONMENT.
More precisely, EAs maintain a POPULATION of structures, that evolve according to rules of selection and other operators, that are referred to as "search operators", (or GENETIC OPERATORs), such as RECOMBINATION and mutation. Each individual in the population receives a measure of it's FITNESS in the environment. Reproduction focuses attention on high fitness individuals, thus exploiting (cf. EXPLOITATION) the available fitness information. Recombination and mutation perturb those individuals, providing general heuristics for EXPLORATION. Although simplistic from a biologist's viewpoint, these algorithms are sufficiently complex to provide robust and powerful adaptive search mechanisms.

--- "An Overview of Evolutionary Computation" [ECML93], 442-459.

It cannot be stressed too strongly that an evolutionary algorithm (as a SIMULATION of a genetic process) is not a random search for a solution to a problem (highly fit individual). EAs use stochastic processes, but the result is distinctly non-random (better than random).

PSEUDO CODE

```
Algorithm EA is
     // start with an initial time
     t := 0;
     // initialize a usually random population of individuals
     initpopulation P (t);
     // evaluate fitness of all initial individuals in population
     evaluate P (t);
     // test for termination criterion (time, fitness, etc.)
     while not done do
          // increase the time counter
          t := t + 1;
          // select sub-population for offspring production
          P' := selectparents P (t);
          // recombine the "genes" of selected parents
          recombine P' (t);
          // perturb the mated population stochastically
          mutate P' (t);
          // evaluate it's new fitness
          evaluate P' (t);
          // select the survivors from actual fitness
```

```
            P := survive P,P' (t);
        od
    end EA.
```

What's a Genetic Algorithm (GA)?

The GENETIC ALGORITHM is a model of machine learning which derives its behavior from a metaphor of some of the mechanisms of EVOLUTION in nature. This is done by the creation within a machine of a POPULATION of INDIVIDUALs represented by CHROMOSOMEs, in essence a set of character strings that are analogous to the base-4 chromosomes that we see in our own DNA. The individuals in the population then go through a process of simulated "evolution".

Genetic algorithms are used for a number of different application areas. An example of this would be multidimensional OPTIMIZATION problems in which the character string of the chromosome can be used to encode the values for the different parameters being optimized. In practice, therefore, we can implement this genetic model of computation by having arrays of bits or characters to represent the chromosomes. Simple bit manipulation operations allow the implementation of CROSSOVER, MUTATION and other operations. Although a substantial amount of research has been performed on variable-length strings and other structures, the majority of work with genetic algorithms is focussed on fixed-length character strings. We should focus on both this aspect of fixed-lengthness and the need to encode the representation of the solution being sought as a character string, since these are crucial aspects that distinguish GENETIC PROGRAMMING, which does not have a fixed length representation and there is typically no encoding of the problem.

PSEUDO CODE

```
    Algorithm GA is
        // start with an initial time
        t := 0;
        // initialize a usually random population of individuals
        initpopulation P (t);
        // evaluate fitness of all initial individuals of population
        evaluate P (t);
        // test for termination criterion (time, fitness, etc.)
        while not done do
            // increase the time counter
            t := t + 1;
            // select a sub-population for offspring production
            P' := selectparents P (t);
            // recombine the "genes" of selected parents
            recombine P' (t);
            // perturb the mated population stochastically
            mutate P' (t);
            // evaluate it's new fitness
            evaluate P' (t);
            // select the survivors from actual fitness
            P := survive P,P' (t);
        od
    end GA.
```

What's Evolutionary Programming (EP)?

Introduction

EVOLUTIONARY PROGRAMMING, originally conceived by Lawrence J. Fogel in 1960, is a stochastic OPTIMIZATION strategy similar to GENETIC ALGORITHMs, but instead places emphasis on the behavioral linkage between PARENTs and their OFFSPRING, rather than seeking to emulate specific GENETIC OPERATORS as observed in nature. Evolutionary programming is similar to EVOLUTION STRATEGIES, although the two approaches developed independently.

The basic EP method involves 3 steps (Repeat until a threshold for iteration is exceeded or an adequate solution is obtained):

(1) Choose an initial POPULATION of trial solutions at random. The number of solutions in a population is highly relevant to the speed of optimization, but no definite answers are available as to how many solutions are appropriate (other than >1) and how many solutions are just wasteful.

(2) Each solution is replicated into a new population. Each of these offspring solutions are mutated according to a distribution of MUTATION types, ranging from minor to extreme with a continuum of mutation types between. The severity of MUTATION is judged on the basis of the functional change imposed on the parents.

(3) Each offspring solution is assessed by computing it's fitness. Typically, a stochastic tournament is held to determine N solutions to be retained for the population of solutions, although this is occasionally performed deterministically. There is no requirement that the population size be held constant, however, nor that only a single offspring be generated from each parent.

It should be pointed out that EP typically does not use any CROSSOVER as a GENETIC OPERATOR.

PSEUDO CODE

```
Algorithm EP is
     // start with an initial time
     t := 0;
     // initialize a usually random population of individuals
     initpopulation P (t);
     // evaluate fitness of all initial individuals of population
     evaluate P (t);
     // test for termination criterion (time, fitness, etc.)
     while not done do
          // perturb the whole population stochastically
          P'(t) := mutate P (t);
          // evaluate it's new fitness
          evaluate P' (t);
          // stochastically select the survivors from actual fitness
          P(t+1) := survive P(t),P'(t);
          // increase the time counter
          t := t + 1;
     od
end EP.
```

What's an Evolution Strategy (ES)?

EVOLUTION STRATEGIEs were invented to solve technical

OPTIMIZATION problems (TOPs) like e.g. constructing an optimal flashing nozzle, and until recently ES were only known to civil engineering folks, as an alternative to standard solutions. Usually no closed form analytical objective function is available for TOPs and hence, no applicable optimization method exists, but the engineer's intuition.

A single INDIVIDUAL of the ES' population consists of the following GENOTYPE representing a point in the SEARCH SPACE:

OBJECT VARIABLES

Real-valued x_i have to be tuned by recombination and mutation such that an objective function reaches its global optimum.

STRATEGY VARIABLEs

Real-valued s_i (usually denoted by a lowercase sigma) or mean stepsizes determine the mutability of the x_i. They represent the STANDARD DEVIATION of a $(0, s_i)$ GAUSSIAN DISTRIBUTION (GD) being added to each x_i as an undirected mutation. With an "expectancy value" of 0 the parents will produce offspring similar to themselves on average. In order to make a doubling and a halving of a stepsize equally probable, the s_i mutate log-normally, distributed, i.e. exp(GD), from generation to generation. These stepsizes hide the internal model the population has made of its ENVIRONMENT, i.e. a SELF-ADAPTATION of the stepsizes has replaced the exogenous control of the (1+1) ES.

This concept works because selection sooner or later prefers those individuals having built a good model of the objective function, thus producing better offspring. Hence, learning takes place on two levels: (1) at the genotypic, i.e. the object and strategy variable level and (2) at the phenotypic level, i.e. the FITNESS level.

Depending on an individual's x_i, the resulting objective function value $f(x)$, where x denotes the vector of objective variables, serves as the PHENOTYPE (fitness) in the selection step. In a plus strategy, the m best of all $(m+l)$ individuals survive to become the parents of the next generation. Using the comma variant, selection takes place only among the l offspring. The second scheme is more realistic and therefore more successful, because no individual may survive forever, which could at least theoretically occur using the plus variant. Untypical for conventional optimization algorithms and lavish at first sight, a comma strategy allowing intermediate deterioration performs better! Only by forgetting highly fit individuals can a permanent adaptation of the stepsizes take place and avoid long stagnation phases due to misadapted s_i's. This means that these individuals have built an internal model that is no longer appropriate for further progress, and thus should better be discarded.

By choosing a certain ratio m/l, one can determine the convergence property of the evolution strategy: If one wants a fast, but local convergence, one should choose a small HARD SELECTION, ratio, e.g. (5,100), but looking for the global optimum, one should favour a softer selection (15,100).

What's a Classifier System (CFS)?

Holland envisioned a cognitive system capable of classifying the goings on in its environment, and then reacting to these goings on appropriately. So what is needed to build such a system? Obviously, we need (1) an environment; (2) receptors that tell our system about the goings on; (3) effectors, that let our system manipulate its environment; and (4) the system itself, conveniently a "black box" in this first approach, that has (2) and (3) attached to it, and "lives" in (1).

PSEUDO CODE (Learning CFS)

```
Algorithm LCS is
     // start with an initial time
     t := 0;
     // an initially empty message list
     initMessageList ML (t);
     // and a randomly generated population of classifiers
     initClassifierPopulation P (t);
     // test for cycle termination criterion (time, fitness, etc.)
     while not done do
          // increase the time counter
          t := t + 1;
          // 1. detectors check whether input messages are present
          ML := readDetectors (t);
          // 2. compare ML to the classifiers and save matches
          ML' := matchClassifiers ML,P (t);
          // 3. highest bidding classifier(s) collected in ML' wins the
          // "race" and post the(ir) message(s)
          ML' := selectMatchingClassifiers ML',P (t);
          // 4. tax bidding classifiers, reduce their strength
          ML' := taxPostingClassifiers ML',P (t);
          // 5. effectors check new message list for output msgs
          ML := sendEffectors ML' (t);
          // 6. receive payoff from environment (REINFORCEMENT)
          C := receivePayoff (t);
          // 7. distribute payoff/credit to classifiers (e.g. BBA)
          P' := distributeCredit C,P (t);
          // 8. Eventually (depending on t), an EA (usually a GA) is
          // applied to the classifier population
          if criterion then
               P := generateNewRules P' (t);
          else
               P := P'
     od
end LCS.
```

What's Genetic Programming (GP)?

GENETIC PROGRAMMING is the extension of the genetic model of learning into the space of programs. That is, the objects that constitute the POPULATION are not fixed-length character strings that encode possible solutions to the problem at hand, they are programs that, when executed, "are" the candidate solutions to the problem. These programs are expressed in genetic programming as parse trees, rather than as lines of code. Thus, for example, the simple program "a + b * c" would be represented as:

```
  +
 / \
a   *
   / \
  b   c
```

or, to be precise, as suitable data structures linked together to achieve this effect. Because this is a very simple thing to do in the programming language Lisp, many GPers tend to use Lisp. However, this is simply an implementation detail. There are straightforward methods to implement GP using a non-Lisp programming environment.

The programs in the population are composed of elements from the FUNCTION SET and the TERMINAL SET, which are typically fixed sets of symbols selected to be appropriate to the solution of problems in the domain of interest.

In GP the CROSSOVER operation is implemented by taking randomly selected subtrees in the INDIVIDUALs (selected according to FITNESS) and exchanging them.

It should be pointed out that GP usually does not use any MUTATION as a GENETIC OPERATOR.

What applications of EAs are there?

But EAs are especially badly suited for problems where efficient ways of solving them are already known, (unless these problems are intended to serve as benchmarks). Special purpose algorithms, i.e. algorithms that have a certain amount of problem domain knowledge hard coded into them, will usually outperform EAs, so there is no black magic in EC. EAs should be used when there is no other known problem solving strategy, and the problem domain is NP-complete. That's where EAs come into play: heuristically finding solutions where all else fails.

BIOCOMPUTING

Biocomputing, or Bioinformatics, is the field of biology dedicated to the automatic analysis of experimental data (mostly sequencing data). Several approaches to specific biocomputing problems have been described that involve the use of GA, GP and simulated annealing.

There are three main domains to which GA have been applied in Bioinformatics: protein folding, RNA folding, sequence alignment.

GAME PLAYING

GAs can be used to evolve behaviors for playing games. Work in evolutionary GAME THEORY typically surrounds the EVOLUTION of a POPULATION of players who meet randomly to play a game in which they each must adopt one of a limited number of moves.

JOB-SHOP SCHEDULING

The Job-Shop Scheduling Problem (JSSP) is a very difficult NP-complete problem which, so far, seems best addressed by sophisticated

branch and bound search techniques. GA researchers, however, are continuing to make progress on it.
Similar to the JSSP is the Open Shop Scheduling Problem (OSSP).
A simpler form of job
shop problem is the Flow-Shop Sequencing problem, recently has been successful on applying GAs to this.

MANAGEMENT SCIENCES

Applications of EA in management science and closely related fields like organizational ecology is a domain that has been covered by some EA researchers - with considerable bias towards scheduling problems.

Nissen, V. (1993) "Evolutionary Algorithms in Management Science: An Overview and List of References", Papers on Economics and Evolution, edited by the European Study Group for Evolutionary Economics. This report is also avail. via anon. FTP from ftp.gwdg.de:/pub/msdos/reports/wi/earef.eps

TIMETABLING

This has been addressed quite successfully with GAs. A very common manifestation of this kind of problem is the timetabling of exams or classes in Universities, etc.

CELLULAR PROGRAMMING: Evolution of Parallel Cellular Machines

Sipper, M. (1997) "Evolution of Parallel Cellular Machines: The Cellular Programming Approach", Springer-Verlag, Heidelberg.

EVOLVABLE HARDWARE

the term evolware has been used to describe such evolving ware, with current implementations centering on hardware, while raising the possibility of using other forms in the future, such as bioware.

How many EAs exist? Which?

There are currently 3 main paradigms in EA research:
GENETIC ALGORITHMs,
EVOLUTIONARY PROGRAMMING,
and EVOLUTION STRATEGIEs.
CLASSIFIER SYSTEMs and GENETIC PROGRAMMING are OFFSPRING of the GA community.

Besides this leading crop, there are numerous other different approaches, alongside hybrid experiments, i.e. there exist pieces of software residing in some researchers computers, that have been described in papers in conference proceedings, and may someday prove useful on certain tasks.

WWW resources

Newsgroup : comp.ai.genetic
The Santa Fe Institute (USA)
http://alife.santafe.edu/~joke/encore/www/
Purdue University, West Lafayette, IN (USA)
http://www.cs.purdue.edu/coast/archive/clife/FAQ/www/
Heitkoetter, Joerg and Beasley, David, eds. (1997) "The Hitch-

Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions (FAQ)", USENET: comp.ai.genetic. Available via anonymous FTP from rtfm.mit.edu:/pub/usenet/news.answers/ai-faq/genetic/ About 110 pages.

Beasley, D., Bull, D.R., & Martin, R.R. (1993) "An Overview of Genetic Algortihms: Part 1, Fundamentals", University Computing, 15(2) 58-69. Available by ftp from ENCORE in file: GA/papers/over93.ps.gz or from ralph.cs.cf.ac.uk:/pub/papers/GAs/ga_overview1.ps

Beasley, D., Bull, D.R., & Martin, R.R. (1993) "An Overview of Genetic Algortihms: Part 2, Research Topics", University Computing, 15(4) 170-181. Available by ftp from ENCORE in file: GA/papers/over93-2.ps.gz or from ralph.cs.cf.ac.uk:/pub/papers/GAs/ga_overview2.ps

Whitley, D. (1993) "A Genetic Algorithm Tutorial", Colorado State University, Dept. of CS, TR CS-93-103. Available by ftp from ftp.cs.colostate.edu:/pub/public_html/TechReports/1993/tr-103.ps.Z or from http://www.cs.colostate.edu

Jarmo Alander has compiled probably the biggest EC bibliography around. It has 2500 entries, and is available in postscript form by ftp from: garbo.uwasa.fi:/pc/research/2500GArefs.ps.gz

GP mailing list FAQ and from http://www-cs-faculty.stanford.edu/~koza/

Local publications can be found at Intelligent Systems Laboratory, Department of Computer Engineering, Chulalongkorn university : http://orange.cp.eng.chula.ac.th

TEXT

Holland, J, “Adaptation in natural and artificial systems”, MIT Press, 1992, first edition University of Michigan, 1975.

Goldberg, D., “Genetic Algorithms in search, optimization, and machine learning”, Addison-Wesley, 1989.

Mitchell, M., “An introduction to genetic algorithms”, MIT Press, 1996.

Koza, J. “Genetic Programming : On the programming of computers by means of natural selection”, MIT Press, 1992.

Davis, L. (ed), "Handbook of Genetic Algorithms", Van Nostrand Reinhold, New York, NY, 1991.

Michalewicz, Z, Genetic algorithms + Data Structures = Evolution Programs", Springer-Verlag, New York, NY, 1992. Also second, extended edition (1994) with index.

Fogel, D., "Evolutionary Computation: Toward a New Philosophy of Machine Intelligence", Piscataway, NJ: IEEE Press. ISBN 0-7803-1048-0, 1995.

JOURNAL ARTICLES

Holland, J.H. (1992) "Genetic Algorithms", Scientific American, 267(1), 66-72.

Goldberg, D. (1994), "Genetic and Evolutionary Algorithms Come of Age", Communications of the ACM, 37(3), 113--119.