Contract no: 20851A2/2/102

Artificial Intelligence: Genetic Programming

Wolfgang Banzhaf

Dept. of Computer Science, University of Dortmund, Dortmund, Germany

1 Introduction

The term Genetic Programming (GP, for short) describes a research area within Artificial Intelligence (AI, for short) that deals with the *evolution* of computer code. The term *evolution* refers to an artificial process analogous to natural evolution of living organisms, but which has been abstracted and stripped off of most of its intricate details. The resultant algorithms then yield approximate solutions to problems in machine learning or induce precise solutions in the form of grammatically correct (language) structures for the automatic programming of computers.

Genetic Programming is part of the growing set of Evolutionary Algorithms which apply search principles analogous to those of natural evolution in a variety of different problem domains, notably parameter optimization. Evolutionary Programming, Evolutionary Strategies and Genetic Algorithms are three other branches of the area of Evolutionary Algorithms which mostly find applications as optimization techniques. The major distinctions between GP and these other areas of Evolutionary Algorithms are that GP handles active components like symbolic expressions or instructions as opposed to simple parameters, and that GP is able to develop its own representation of a problem by allowing variable complexity of its individuals.

All Evolutionary Algorithms follow Darwin's principle of differential natural selection. This principle states that the following preconditions must be fullfilled for evolution to occur via (natural) selection:

- There are entities called individuals which form a population. These entities can reproduce or can be reproduced.
- There is heredity in reproduction, that is to say that individuals produce similar offspring.
- In the course of reproduction there is variety which affects the likelihood of survival and therefore of reproducibility of individuals.
- There are finite resources which cause the individuals to compete. Due to overreproduction of individuals not all can survive the struggle for existence. Differential natural selection exerts a continuous pressure towards improved individuals.

2 The Mechanisms behind GP

GP works with a population of programs that are executed or interpreted in order to judge their behavior. Usually, a scoring operation called fitness measurement is applied to the outcome of the behavior. For instance, the deviation between the quantitative output of a program and its target value (defined through an error function) could be used to judge the behaviour of the program. This is straight-forward if the function of the target program can be clearly defined. Results may also be defined as side-effects of a program, such as consequences of the physical behavior of a robot controlled by a genetically developed program. Sometimes, an explicit fitness measure is missing, for instance in a game situation, and the results of the game (winning or loosing) are taken to be sufficient scoring for the program's strategy. The general approach is to apply a variety of programs to the same problem and to compare their performance relative to each other.

The outcome of fitness measurement are used to select programs. There are a number of different methods for selection, both determinstic and stochastic. Selection determines (i) which programs are allowed to survive (overproduction selection), and (ii) which programs are allowed to reproduce (mating selection). Once a set of programs has been selected for further reproduction, the following operators are applied:

- reproduction

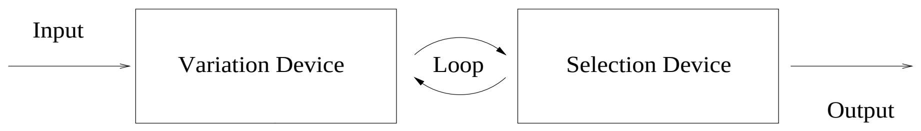

Figure 1

The variation selection loop of GP.

- <u>mutation</u>
- <u>crossover</u>

Reproduction simply copies an individual, *mutation* varies the structure of an individual under control of a random number generator, and *crossover* mixes the structure of two (or more) programs to generate one or more new programs. Additional variation operators are applied in different applications. Most of these contain problem knowledge in the form of heuristic search recipes which are adapted to the problem domain.

In this way, fitness advantages of individual programs are exploited in a population to lead to better solutions. A key effort in Genetic Programming is the definition of the fitness measure. Sometimes the fitness measure has to be iteratively improved in order for the evolved solutions to actually perform the function they were intended for. The entire process can be seen in close analogy to <u>breeding</u> animals. The breeder has to select those individuals from the population which carry the targeted traits to a higher degree than others.

Researchers [15,3,12] suggested ideas similar to GP already in the early days of AI, but did not get very far. So it was only after other techniques in Evolu-

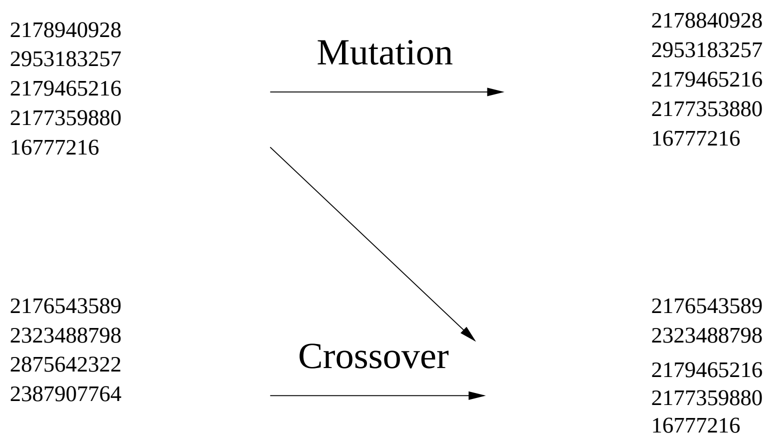

Figure 2

The primary operations of GP, mutation and crossover, are applied here to programs represented by sequences of instructions. The instructions are coded as integer numbers which allows easy manipulation by access to these numbers.

tionary algorithms had been successfully developed that GP emerged. Earlier work concerned *genetic algorithms* [6], *evolution strategies* [13], and *evolutionary programming* [2]. These methods have been applied successfully to a wide spectrum of problem domains, especially in optimization. However, it was unclear for a long time whether the principles of evolution could be applied to computer code, with all its dependencies and structural brittleness. Negative results from early experiments seemed to indicate that evolution of computer code was not possible. Successes were all in the area of contraint optimization [10], where methods were made available for dealing with structural brittleness. These methods found their way into programming and gave rise to the new field of GP [7].

3 Progress and State-of-the-Art

In his seminal work of 1992, Koza established the field of GP by arguing convincingly that manipulation of symbolic tree structures is possible with evolutionary algorithms and that the resulting technique would have a wide variety of applications. In subsequent years, the field experienced both broadening and deepening [1]. Many different representations for GP were studied, among them other generic data structures such as sequences of instructions or directed graphs, as well as more exotic data structures such as stacks or neural networks. Today, different approaches are considered as GP, from the evolution of parse trees to the evolution of arbitrary structures. The overarching principle is to subject structures with variable complexity to forces of evolution by applying mutation, crossover and fitness-based selection. The results are not necessarily programs.

An ever present difficulty with GP is that the evolution of structures of variable complexity leads often to large structures with considerable redundancy. Notably, variable complexity often leads to inefficient and space consuming code. It was subsequently recognized that the evolutionary forces exerted a pressure toward more complex solutions, most of which could be removed after evolution without doing any harm to the evolved solution. By drawing an analogy from biological evolution of genomes, this phenomenon was called "intron growth", or growth of ineffective code. Though the explanation for this phenomenon is not fully understood yet, it was found that at least two different influences were at work promoting the growth of complexity during evolution. The most important one has to do with the protection effect of redundant code if subjected to the action of crossover or mutation. Redundant code was resistant to crossover and mutation and allowed its carrier solution to survive better, compared to other individuals which did not possess the redundancy.

Currently, many researchers are working to transfer results from research in Genetic Algorithms to GP. To achieve results in GP is generally more difficult since GP works with variable complexity and multiple fitness cases for fitness scoring. The schema theory of Genetic Algorithms [5,16] has been a primary target of knowledge transfer. In the meantime, a number of different schema theorems have been formulated for GP.

When analyzing search spaces of programs it was realized that their size is many orders of magnitude larger than search spaces of combinatorial optimization problems. A typical size for a program search space might be $10^{100,000}$, as opposed to a typical search space for a combinatorial optimization problem of the order of 10^{100}. Although this might be interpreted as discouraging for search mechanisms, it was also realized that the solution density in program spaces is, above a certain threshold, constant with changing complexity [9]. In other words, there are proportionally many more valid solutions in programs spaces than in the spaces of combinatiorial optimization problems.

4 Applications

The main application areas of GP are (from narrow to wide) [1]:

- Computer Science
- Science
- Engineering
- Art and Entertainment.

In Computer Science, the development of algorithms has been a focus of attention. By being able to manipulate symbolic structures, GP is one of the few heuristic search methods for algorithms. Sorting algorithms, caching algorithms, random number generators and algorithms for automatic parallelization of code [11], to name a few, have been studied. The spectrum of applications in Computer Science spans from the generation of proofs for predicate calculus to the evolution of machine code for accelerating function evaluation. The general tendency is to try to automate the design process for algorithms of different kinds.

Typical applications in Science are to modeling and pattern recognition. Modeling certain processes in Physics and Chemistry with the unconventional help of evolutionary creativity supports research and understanding of the systems under study. Pattern recognition is a key ability in molecular biology and other branches of biology, as well as in Science in general. Here, GP has delivered first results that are competitive if not better than human-generated results.

In Engineering, GP is used in competition or cooperation with other heuristic methods such as Neural Networks or Fuzzy Systems. The general goal is again to model processes such as production plants, or to classify results of production. Control of man-made apparatus is another area where GP has been used successfully, with process control and robot control the primary applications.

In Art and Entertainment, GP is used to evolve realistic animation scenes and appealing visual graphics. It also has been used to extract structural information from musical composition in order to model the process so that automatic composition of music pieces becomes possible.

Many of these problems require a huge amount of computational power on the part of the GP systems. Parallel evolution has hence been a key engineering aspect of developments in GP. As a paradigm, GP is very well suited for a natural way of parallelization. With the advent of inexpensive parallel hardware and software [14], a considerable proliferation of results is expected from GP systems.

5 Methodological Issues and Future Directions

In recent years, some researchers have claimed human-competitive results in the application of GP to certain problems [8]. These claims are based on a comparison between the presently-best known human solutions to a problem and its GP counterpart. Usually, a large amount of computational power had to be invested in order to gain human-competitive results from Genetic Programming runs. Due to the ability of the human mind to quickly grasp the recipes of problem solution an artificial system has applied, the question remains open whether GP solutions will stay better than human solutions.

Theory of GP is presently greatly underdeveloped and will need to progress quickly in order to catch up with other evolutionary algorithm paradigms. Most of the obstacles stem from the fact of variable complexity of solutions evolved in GP. Implementation of GP will benefit in the coming years from new approaches which include research from developmental biology. Also, it will be necessary to learn to handle the redundancy forming pressures in the evolution of code.

Application of GP will continue to broaden. Many applications focus on controlling behavior of real or virtual agents. In this role, Genetic Programming may contribute considerably to the growing field of social and behavioral simulations. Genetic Algorithms have already been found beneficial in optimizing strategies of social agents [4]. With its ability to adjust the complexity of a strategy to the environment and to allow competition between agents, GP is well positioned to play an important role in the study and simulation of societies and their evolution.

Bibliography

[1] Banzhaf W, Nordin P, Keller, R and Francone F 1998 *Genetic Programming - An Introduction.* Morgan Kaufmann, San Francisco, CA

[2] Fogel L, Owens A and Walsh M 1966 *Artificial Intelligence through Simulated Evolution.* Wiley, New York

[3] Friedberg R M 1958 A Learning Machine: Part I. *IBM Journal of Research and Development.* 2: 2–13

[4] Gilbert N and Troitzsch K 1999 *Simulation for the Social Scientist.* Open University Press, Buckingham, UK

[5] Goldberg D 1989 *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison Wesley, Reading, MA

[6] Holland J 1975 *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, MI

[7] Koza J 1992 *Genetic Programming.* MIT Press, Cambridge, MA

[8] Koza J, Andre D, Bennett F and Keane M 1999 *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, San Francisco, CA

[9] Langdon W 1999 Boolean function fitness spaces. In: Poli R, Nordin P Langdon, W and Fogarty T (eds.) *Proceedings EuroGP'99.* Springer, Berlin

[10] Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, Berlin

[11] Ryan C 2000 *Automatic Re-engineering of Software Using Genetic Programming.* Kluwer Academic, Boston, MA

[12] Samuel A 1963 Some studies in machine learning using the game of checkers. In: Feigenbaum E and Feldman J (eds.) *Computers and Thought.* McGraw-Hill, New York

[13] Schwefel H P 1981 *Numerical Optimization of Computer Models.* Wiley, Chichester

[14] Sterling T, Salmon J, Becker D and Savarese D 1999 *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters.* MIT Press, Cambridge, MA

[15] Turing A 1950 Computing Machinery and Intelligence. *Mind.* 59: 433–460

[16] Vose M 1999 *The simple Genetic Algorithm: Foundations and Theory.* MIT Press, Cambridge, MA