Enhancing Genetic Programming by $-calculus

Eugene Eberbach
Jodrey School of Computer Science, Acadia University ‡
Wolfville, NS, Canada B0P 1X0; eberbach@acadiau.ca

Genetic programming (GP) is a new approach to automatic program synthesis successful on a wide range of problems. As each new approach it poses several open research questions. The paper tries to answer some of them, based on the results derived from the $-calculus, which is a general model of computation with a quantitative aspect (cost) allowing naturally to express optimization and modification in dynamic parallel AI systems. Evolutionary computation has been derived as an attempt to generalize genetic algorithms. The $-calculus is more general than evolutionary algorithms or neural nets. The project on $-calculus aims at investigation, design and implementation of hybrid AI symbolic, connectionist, fuzzy, and evolutionary systems. The approach tries to combine the best of both genetic programming and symbolic AI. As the result, several enhancements of genetic programming are proposed, including introduction of modifications, standardization of functions and fitness function, incorporation of some symbolic AI techniques, and analytic approach to fitness calculation allowing to optimize the length of the program, its correctness, and complexity. The obtained results allowed not only to integrate to some extent both symbolic AI and GP, but additionally, allowed to obtain a new enlight on several GP open questions, including fitness calculation, length of generated trees, choice of terminals/functions.

The $-calculus operates on cost expressions, which can be simple (terminals, functions and modifications operating on functions and terminals). Other cost expressions are built using functions/modifications like cost, suppression, general and nondeterministic choices, sequential and parallel compositions, negation, forward and backward matching, crossover, and definition. Modifications and costs are the two unique and the most important features of the $-calculus. Costs can be explicit or implicit, representing time, money, fitness, error, probability, possibility, complexity, distance, energy, the number of resources, etc. For instance, the costs representing probabilities of cost expressions lead to the *probabilistic $-calculus*, and the costs interpreted as the fuzzy set characteristic function - lead to the *fuzzy-logic $-calculus*. To compute the costs, the most important is the compositionality principle: the cost of the problem consisting of subproblems is the function of costs of subproblems. Cost functions direct problem solving performed by the Inference Engine of the calculus. The Inference Engine can be in a form of a simple modifying algorithm (resembling a basic structure of an evolutionary algorithm or genetic program), or more complex, in particular, leading to a self-modifying algorithm. In such an approach, a Genetic Programming procedure becomes simply a modifying algorithm.

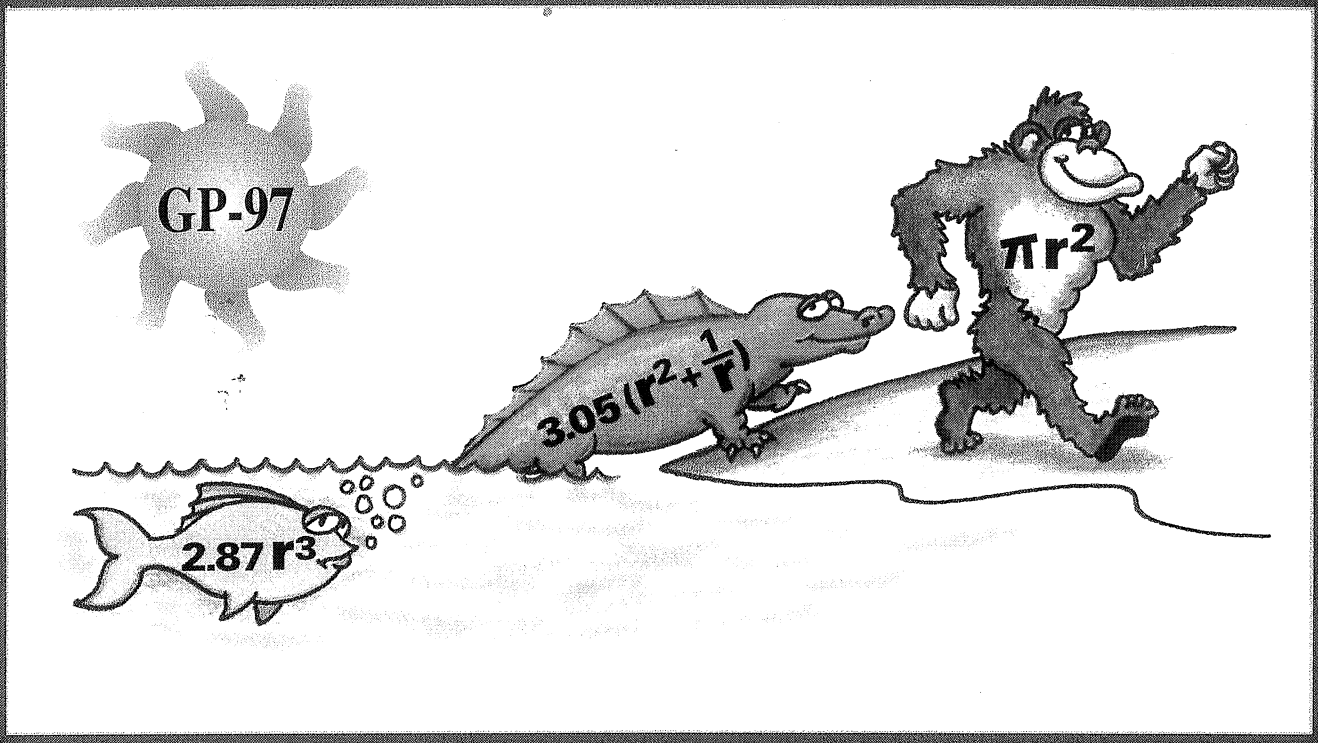
A cost programming paradigm allows to enhance genetic programming by generating nondeterministic and parallel programs. It allows to integrate both symbolic AI inferences and GP operators, and to generate production rules/logic/procedural programs. A majority of programs are built using standard components: compositions, choices, iterations. This is not a case in GP: each program has to have it own terminals and functions. It is proposed that the GP function set included compositions and choices by default, and the user will add his/her own functions, and eventually will remove some defaults (for instance, if the program should not allow parallelism). Genetic programming could also benefit from the library of standard fitness functions. Probably, the most important contribution to genetic programming is to propose to calculate fitness of programs as the function of program components. More precisely, we propose to keep the GP standardized fitness, (better programs have lower costs) estimating how far the program is from the goal, to which we add the cost of component functions used to solve the problem (this resembles classic A* algorithm - so far, GP uses only estimates). It solves the problem of too long GP trees, i.e., longer trees will be more expensive - this means less fit and less likely to be selected for reproduction. It may also allow to treat correctness of generated programs as cost optimization problems (infinite programs will have infinite cost, and programs further apart from the goal will have a higher estimate part of costs).

The cost calculus could be used also to analyze other subareas of evolutionary computation: evolutionary programming, evolution strategies, classifier systems, genetic algorithms, DNA based computing, artificial life, or autonomous agents. In fact, it may lead to an enhancement of genetic programming paradigm to the cost programming paradigm (each statement of the language has cost associated with it), and to incorporate the cost-driven computer architectures ideas into evolvable hardware.

‡Research partially supported by NSERC Grant OGP0046501

CONFERENCE PROCEEDINGS

Genetic Programming 1997



Proceedings of the Second Annual Conference
July 13-16, 1997, Stanford University

Edited by
John R. Koza
Kalyanmoy Deb
Marco Dorigo
David B. Fogel
Max Garzon
Hitoshi Iba
Rick L. Riolo