Towards a New Generation of Program Synthesis Approaches

Justinian P. Rosca
Computer Science Department
University of Rochester
Rochester NY 14627

Position Paper

Proceedings of 1995 Conference on
Software Engineering & Knowledge Engineering
June 22-24, 1995

Towards a New Generation of Program Synthesis Approaches

Justinian P. Rosca
Computer Science Department
University of Rochester
Rochester NY 14627

Automatic programming (AP) has departed from the initial unsuccessful AI attempts to the discovery of programs. Traditional AP approaches rely on explicit domain knowledge to guide program transformation and synthesis. This approach is overturned by the evolutionary method called genetic programming. Genetic programming has demonstrated the surprising and counterintuitive result that computers can learn to synthesize programs without being explicitly programmed how to do it [1].

GP is a randomized, adaptive search technique that represents programs by their parse trees. It starts with a population of randomly generated programs and evolves better and better programs by applying three basic steps: programs in the current population are interpreted and are assigned a fitness value using a domain-dependent fitness function; parent programs are selected based on a fitness-proportional method; offspring programs are generated from parent programs by using the crossover or reproduction operations. Crossover swaps code fragments and is implemented by subtree swapping. Reproduction creates an identical copy of a parent program. Before using GP, three major elements have to be defined. First, the user has to specify the domain-dependent primitives to be used for constructing programs. Second, a domain-dependent fitness function ranks the effects of program execution. Third a termination criterion tests if a satisfactory solution is found. Several parameters have to be additionally specified. Population size is the most important one.

GP has been applied to automatically create programs that solve problems of induction, classification, image analysis, system identification, prediction and control. The GP results are surprising because the size of the GP search space is huge even for simple problems. For example, the parity problem on five boolean variables creates a search space with a size bounded below by 10^{82}. As expected, GP results do not scale well, as it gets harder and harder to solve problems of increasing size.

The capabilities of genetic search can be explained by the building block hypothesis. GP extensions have been designed with the goal of automating the discovery of subroutines that are beneficial during the search for solutions. Such subroutines are a kind of GP building blocks. Two such extensions are *automatic definition of functions* (ADF) [1] and *adaptive representation* (AR) [2]. The former allows the evolution of reusable subroutines. The latter is based on the discovery of useful building blocks of code, which are identified by analyzing the process of evolution. Blocks are generalized and transformed into new functions which extend the problem representation in an adaptive manner. As opposed to ADFs, new AR functions rely on heuristically selected code fragments and do not evolve. Although ADF and AR approaches implement the ideas of discovery, modification, and use of new functions in different ways, both actually evolve a hierarchy of functions that greatly improve search efficiency.

The two extensions introduced above are hierarchical GP approaches. They address three important program synthesis issues: reuse, incorporation of domain knowledge and scalability. (1) Reuse of code is the fundamental feature of reproduction operations. Functional composition of diverse program fragments determines a continuous generalization and specialization process which creates more fit programs. Moreover, reuse is supported by the automatic capability to define, evolve and repeatedly invoke subroutines. Therefore, problems whose solutions parameterize and reuse blocks of code are well tackled in GP. (2) Domain knowledge can be taken into account through block fitness functions [2]. Although the primitives of a final solution may not be known in advance, knowledge of potentially useful code fragments greatly enhances GP convergence. (3) Scalability is a consequence of the hierarchical GP approach. A hierarchy of evolved functions generates a complex program behavior. This allows GP to solve problems of increased difficulty.

Current research focuses on the exploitation of opportunities for adaptation and learning in order to make search more efficient for complex problems. Adaptation can take place at several levels: problem representation, operators, and control strategy. Another research direction is the introduction of more powerful constructs such as memory, iteration and recursion. Research also aims for understanding the GP machinery through theoretical analysis or analogy with other techniques. Such research efforts will lay the basis for a new generation of powerful program synthesis techniques that will combine more traditional, specialized AP methods with the general GP technique.

References

[1] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Massachusetts, 1992.

[2] J. P. Rosca and D. H. Ballard. Hierarchical self-organization in genetic programming. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 251–258. Morgan Kaufmann Publishers, Inc, 1994.