Genetic Programming

William H. Hsu
Kansas State University, USA

INTRODUCTION

Genetic programming (**GP**) is a sub-area of evolutionary computation first explored by John Koza (1992) and independently developed by Nichael Lynn Cramer (1985). It is a method for producing computer programs through adaptation according to a user-defined fitness criterion, or objective function.

Like genetic algorithms, GP uses a representation related to some computational model, but in GP, fitness is tied to task performance by specific program semantics. Instead of strings or permutations, genetic programs are most commonly represented as variable-sized expression trees in imperative or functional programming languages, as grammars (O'Neill & Ryan, 2001), or as circuits (Koza et al., 1999). GP uses patterns from biological evolution to evolve programs:

- **Crossover:** Exchange of genetic material such as program subtrees or grammatical rules
- **Selection:** The application of the fitness criterion to choose which individuals from a population will go on to reproduce
- **Replication:** The propagation of individuals from one generation to the next
- **Mutation:** The structural modification of individuals

To work effectively, GP requires an appropriate set of program operators, variables, and constants. Fitness in GP is typically evaluated over fitness cases. In data mining, this usually means training and validation data, but cases can also be generated dynamically using a simulator or directly sampled from a real-world problem solving environment. GP uses evaluation over these cases to measure performance over the required task, according to the given fitness criterion.

BACKGROUND

Although Cramer (1985) first described the use of crossover, selection, and mutation and tree representations for using genetic algorithms to generate programs, Koza is indisputably the field's most prolific and persuasive author. (Wikipedia, 2007) In four books since 1992, Koza et al. have described GP-based solutions to numerous toy problems and several important real-world problems.

> **State of the field:** To date, GPs have been successfully applied to a few significant problems in machine learning and data mining, most notably symbolic regression and feature construction. The method is very computationally intensive, however, and it is still an open question in current research whether simpler methods can be used instead. These include supervised inductive learning, deterministic optimization, randomized approximation using non-evolutionary algorithms (such as Markov chain Monte Carlo approaches), or genetic algorithms and evolutionary algorithms. It is postulated by GP researchers that the adaptability of GPs to structural, functional, and structure-generating solutions of unknown form makes them more amenable to solving complex problems. Specifically, Koza et al. demonstrate (1999, 2003) that in many domains, GP is capable of "human-competitive" automated discovery of concepts deemed to be innovative through technical review such as patent evaluation.

MAIN THRUST OF THE CHAPTER

The general strengths of genetic programs lie in their ability to produce solutions of variable functional form, reuse partial solutions, solve multi-criterion optimization problems, and explore a large search space of solutions in parallel. Modern GP systems are also able to produce structured, object-oriented, and functional

programming solutions involving recursion or iteration, subtyping, and higher-order functions.

A more specific advantage of GPs are their ability to represent procedural, generative solutions to pattern recognition and machine learning problems. Examples of this include image compression and reconstruction (Koza, 1992) and several of the recent applications surveyed below.

GP Methodology

GP in pattern classification departs from traditional supervised inductive learning in that it evolves solutions whose functional form is not determined in advance, and in some cases can be theoretically arbitrary. Koza (1992, 1994) developed GPs for several pattern reproduction problems such as the multiplexer and symbolic regression problems.

Since then, there has been continuing work on inductive GP for pattern classification (Kishore et al., 2000), prediction (Brameier & Banzhaf, 2001), and numerical curve-fitting (Nikolaev & Iba, 2001, IEEE Trans. Evol. Comp.). GP has been used to boost performance in learning polynomial functions (Nikolaev & Iba, 2001, GP & Evol. Machines). More recent work on tree-based multi-crossover schemes has produced positive results in GP-based design of classification functions (Muni et al., 2004).

While early work in GP for data mining and knowledge discovery in databases (KDD) focused on specific fitness measures related to classification and prediction (Eggermont *et al.*, 1999), more recent work has sought to use GP to implement search behaviors and procedural solutions. Among the methodologies related to GP are swarm intelligence approaches such as ant colony optimization (ACO) and particle swarm optimization (PSO), which seek to evolve solutions through fine-grained simulation of many simple agents. (Azzag *et al.*, 2007; Holden & Freitas, 2007; Tsang & Kwong, 2007)

Applications in Data Mining and Warehousing

The domains within data mining and warehousing where GP has been most successfully applied in recent research include classification (Raymer et al., 1996; Connolly, 2004a; Langdon & Buxton, 2004; Langdon & Barrett, 2005; Holden & Freitas, 2007), prediction (Kaboudan, 2000), and search (Burke & Kendall, 2005). Higher-level tasks such as decision support are often reduced to classification or prediction, while the symbolic representation (S-expressions) used by GP admits query optimization.

GP for Control of Inductive Bias, Feature Construction, and Feature Extraction

GP approaches to inductive learning face the general problem of optimizing inductive bias: the preference for groups of hypotheses over others on bases other than pure consistency with training data or other fitness cases. Krawiec (2002) approaches this problem by using GP to preserve useful components of representation (features) during an evolutionary run, validating them using the classification data, and reusing them in subsequent generations. This technique is related to the wrapper approach to KDD, where validation data is held out and used to select examples for supervised learning, or to construct or select variables given as input to the learning system. Because GP is a generative problem solving approach, feature construction in GP tends to involve production of new variable definitions rather than merely selecting a subset.

Evolving dimensionally-correct equations on the basis of data is another area where GP has been applied. Keijzer & Babovic (2002) provide a study of how GP formulates its declarative bias and preferential (search-based) bias. In this and related work, it is shown that proper units of measurement (strong typing) approach can capture declarative bias towards correct equations, whereas type coercion can implement even better preferential bias.

Grammar-Based GP for Data Mining

Not all GP-based approaches use expression tree-based representations, nor functional program interpretation as the computational model. Wong and Leung (2000) survey data mining using grammars and formal languages. This general approach has been shown effective for some natural language learning problems, and extension of the approach to procedural informa-