Graph-based Genetic Programming Workshop

Program

The workshop will start with the welcome and opening by the workshop organizers, followed by an invited talk and by the oral presentations of accepted workshop articles. More details on the program will follow.

Invited talks

“Evolutionary Optimization of Model Merging Recipes”

With the increased diffusion of large and diverse models, Evolutionary Model Merge is proposed as a general method that uses evolutionary techniques to efficiently discover the best ways to combine different models with diverse capabilities, using ideas similar to graph-based neural architecture search.

“Evolutionary Robustness”

W. B. Langdon

University College London, UK

Genetic programming (GP) and other types of Evolutionary Algorithms (EC) have long been demonstrated to be creative. A recently raised question was how much are they used? Data from the genetic programming bibliography for last year suggests 38 ±5% of published papers are primarily on applications which just happen to use GP. Many applications relate to health, civil engineering or solid state materials, e.g. batteries.

Lenski et al’s studies of long term evolution in biology show evolution can retain its ability for continued change even after 75,000 generations. Instead of 36 years, with performance effectively exceeding a trillion GP operations per second, GP experiments can be run to a million generations in weeks on a single computer. Information theory explains why in small populations, GP populations converge and the rate of fitness improvement falls as huge GP trees become more robust to crossover.

Mutation testing on C and C++ programs show that real software can also be robust to many source code changes. As with lisp functional language in tree GP, there is a tendency for deeply nested imperative code to be more robust.

There are already examples of human written software systems that exceed a billion

lines of (imperative) source code. Information theory's failed disruption propagation (FDP) helps to explain why maintaining, testing and debugging such deeply nested code repositories is hard and why software companies prefer unit testing of modules (each of which is typically only shallowly nested) rather than system testing of complete functional hierarchies. There is already SBSE work on automatically optimizing test oracles. FDP suggests systems should be built with many densely packed test agents so that disruption caused by bugs has little distance to travel before being discovered by an oracle.

For evolutionary computing and artificial life experiments aiming for sustained innovation, we propose the use of "mangrove" architectures composed of many small trees which are intimate with their environment. For continuous innovative evolution the fitness function needs to be able to measure on average if genetic changes are good or not, or at least have made a difference. This means we must overcome robustness, without introducing chaos. We suggest this might be met by systems where the bulk of the code remains close to the fitness environment and the disruption caused by most mutations and crossovers has only a short depth to propagate in order to have a measurable fitness impact.

Oral presentations (8 minutes each)

Byron: A Fuzzer for Turing-complete Test Programs

Marco Sacchet, Dimitri Masetta, Giovanni Squillero, Alberto Tonda

Directed Acyclic Program Graph Applied to Supervised Classification

Thibaut Bellanger, Matthieu Le Berre, Manuel Clergue, Jin-Kao Hao

On Search Trajectory Networks for Graph Genetic Programming

Camilo De La Torre, Sylvain Cussat-Blanc, Dennis Wilson, Yuri Lavinas

Minimizing the EXA-GP Graph-Based Genetic Programming Algorithm for Interpretable Time Series Forecasting

Jared Murphy, Travis Desell