Computational Complexity Analysis of Genetic Programming

Frank Neumann

School of Computer Science

University of Adelaide

Joint work with Greg Durrett (UC Berkeley), Una-May O'Reilly (MIT), Markus Wagner (Uni Adelaide), Timo Kötzing (MPI), Reto Spöhel (MPI)

Introduction

There are many

- successful application
- experimental studies

of Genetic Programming.

We want to

- argue in a rigorous way about GP algorithms and
- contribute to their theoretical understanding

This is also important for the acceptance of our algorithms outside our community.

Classical Optimization

Classical algorithm analysis has a large focus on runtime and approximation behavior of algorithms.

Our key questions:

- Which optimization problems can provably be solved by (simple) EAs in polynomial time?
- Which functions can provably be learned by (simple) GP systems in polynomial time?

Theory of Evolutionary Computing

Solid Foundation of Evolutionary Computing:

- Understand how and why such algorithms work.
- Algorithms make use of random decisions.
- Treat them as randomized algorithms.
- Consider their expected runtime and/or success probability.

Computational Complexity Analysis

Black Box Scenario:

- Measure the runtime T by the number of fitness evaluations.
- Consider time to reach
  - An optimal solution.
  - A good approximation.

Analyze:

- Expected number of fitness evaluation.
- Success probability after a fixed number of t steps.

Current Status

Computational Complexity Analysis of evolutionary computing

- EAs for discrete combinatorial optimization (lots of results)
- Evolutionary Multi-Objective Optimization (many results)
- Ant Colony Optimization (some results)
- EAs for continuous optimization (initial results)
- Particle Swarm Optimization (initial results)

Goal: Rigorous insights into the working principles of GP using this approach!

Methods

Huge set of methods for the analysis is available:

- Fitness-based partitions
- Expected distance decrease
- Coupon Collector's Theorem
- Markov, Chebyshev, Chernoff, Hoeffding bounds
- Markov chain theory: waiting times, first hitting times
- Rapidly Mixing Markov Chains
- Random Walks: Gambler's Ruin, drift analysis, martingale theory
- Identifying typical events and failure events
- Potential functions

Genetic Programming (GP):

- High complex GP variants address challenging problems for example in symbolic regression.
- Currently, seems to be impossible to analyze these complex variants on complex problems.

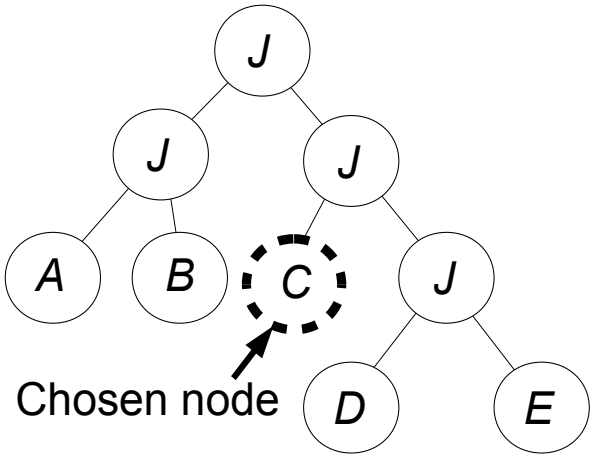
This talk:

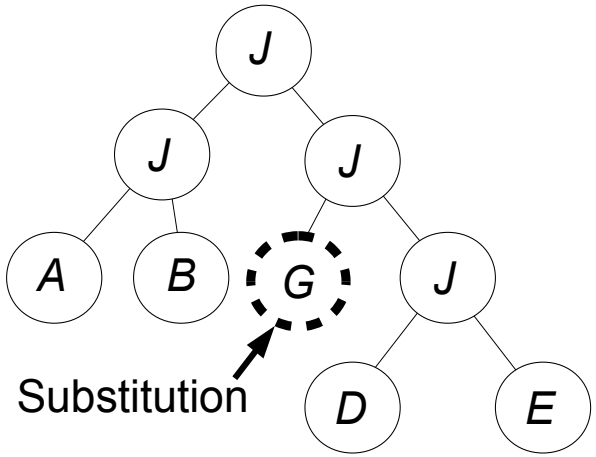
- Analyze (still relevant) versions of GP.
- Give some initial computational complexity results
- Discuss topics for future work

Genetic Programming

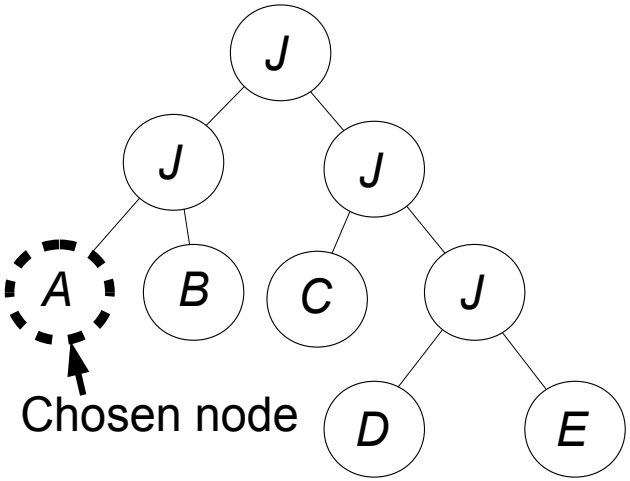
- Type of evolutionary algorithm
- Evolves tree structures for a given problem
- Often used to learn a function
- Consider simple mutation-based genetic programming algorithms

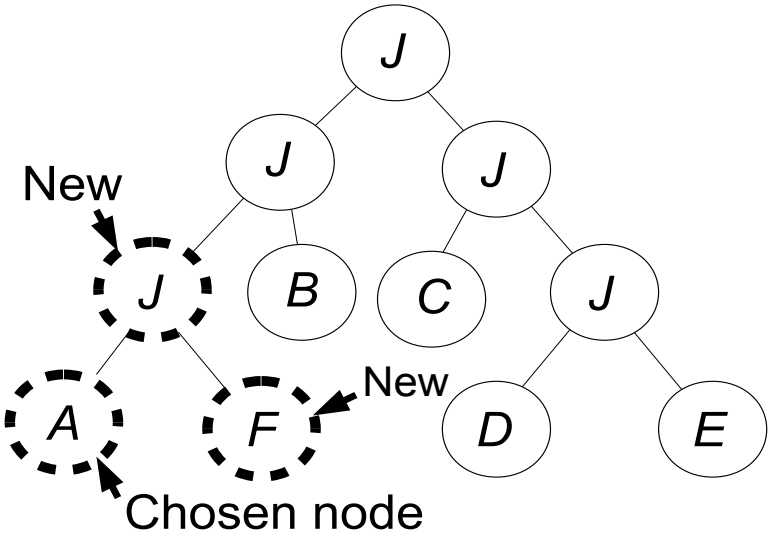
Substitution



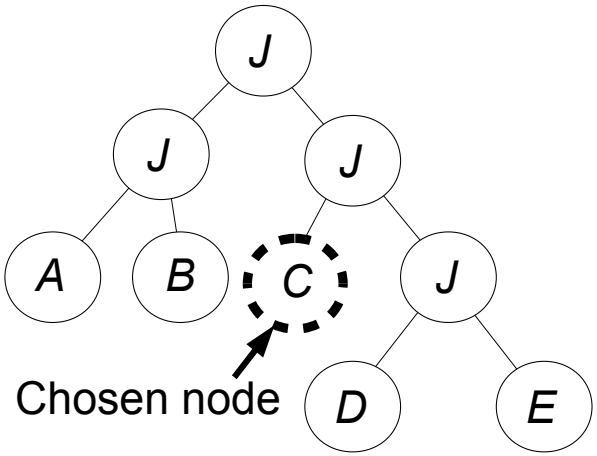


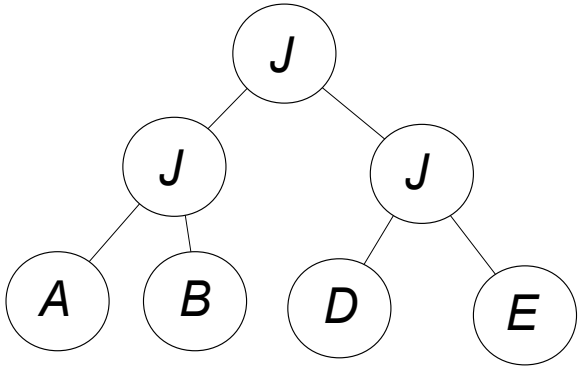
Insertion





Deletion





Simple GP Algorithm

Algorithm 1 ((1+1) GP)**.**

1. *Choose an initial solution* X*.*
2. *Set* $X' := X$*.*
3. *Mutate* X' *by applying HVL-Mutate*$'$ k *times. For each application, randomly choose to either substitute, insert, or delete.*

Substitute
- *If substitute, replace a randomly chosen leaf of* X' *with a new leaf* $u \in L$ *selected uniformly at random.*

Insert
- *If insert, randomly choose a node* v *in* X' *and select* $u \in L$ *uniformly at random. Replace* v *with a join node whose children are* u *and* v*, with the order of the children chosen randomly.*

Delete
- *If delete, randomly choose a leaf node* v *of* X'*, with parent* p *and sibling* u*. Replace* p *with* u *and delete* p *and* v*.*

4. *If* $f(X') \geq f(X)$*, set* $X := X'$*.*
5. *Go to 2.*

Algorithms:

k=1 called (1+1) GP-single

k-1 according to Pois(1) called (1+1) GP-multi

Rejecting Neutral Moves

Algorithm 2 (Acceptance for (1+1) GP*).

4'. If $f(X') > f(X)$, *set* $X := X'$.

ORDER and MAJORITY

Frank Neumann

Simple Functions

ORDER and MAJORITY (Goldberg/O'Reilly 1998)

- $F := \{J\}$, J has arity 2.
- $L := \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$

Properties of functions

- Separable
- Admit multiple solutions

ORDER

- Imitate semantics of a conditional execution path
- Output depends on the order of leaves in an in-order parse of a program tree

1. *Derive conditional execution path P of X:*

   Init: *l an empty leaf list, P an empty conditional execution path*

   1.1 *Parse X inorder and insert each leaf at the rear of l as it is visited.*

   1.2 *Generate P by parsing l front to rear and adding ("expressing") a leaf to P only if it or its complement are not yet in P (i.e. have not yet been expressed).*

2. $f(X) = |\{x_i \in P\}|$.

Example: X $l = (x_1, \bar{x}_4, x_2, \bar{x}_1, x_3, \bar{x}_6)$, $P = (x_1, \bar{x}_4, x_2, x_3, \bar{x}_6)$ and $f(X) = 3$

Analysis for ORDER:

- Use fitness-based partitions
- Let current fitness be k and current tree size T then probability for an improvement is

$$p_k = \Omega\left(\frac{(n-k)^2}{n \max\{T, n\}}\right)$$

- Summing up waiting times, the expected optimization time is upper bounded by $O(nT_{max})$

MAJORITY

- Imitate semantics of multiple statements
- Output depends on the number of leaves in an in-order parse of a program tree

1. *Derive the combined execution statements S of X:*

   Init: *l an empty leaf list, S is an empty statement list.*

   1.1 *Parse X inorder and insert each leaf at the rear of l as it is visited.*

   1.2 *For $i \leq n$: if $count(x_i \in l) \geq count(\bar{x}_i \in l)$ and $count(x_i \in l) \geq 1$, add x_i to S*

2. $f(X) = |S|$.

Example: X $l = (x_1, \bar{x}_4, x_2, \bar{x}_1, \bar{x}_3, \bar{x}_6, x_1, x_4), S = (x_1, x_2, x_4)$ and $f(X) = 3$.

Analysis for MAJORITY:

Observation:

- Plateaus in the search space.
- Inserts are uniform.
- Probability for deletion of a certain type of variable depends on its fraction in the current tree.
- Enables to use random walk arguments.

Can not move on plateau:

- (1+1) GP*-single expected optimization time is infinite
- (1+1) GP*-multi expected optimization time is exponential.

(1+1) GP-single

- Worst case bound: $O(n^2 T_{\max} \log n)$
- Average case for uniform initialization: $O(n T_{\max} \log n)$

Results for ORDER and MAJORITY

	ORDER	
	(1+1) GP	(1+1) GP*
single	$O(nT_{\max})$ w.c. †	$O(n^2)$ w.c.
multi	$O(nT_{\max})$ w.c. †	$O(nT_{\max})$ w.c. †

	MAJORITY	
	(1+1) GP	(1+1) GP*
single	$O(n^2 T_{\max} \log n)$ w.c. † $O(nT_{\max} \log n)$ a.c.	$\Omega(\infty)$ a.c.
multi	?	$\Omega\left(\left(\frac{n}{2e}\right)^{\frac{n}{2}}\right)$ w.c.

Techniques: Fitness-based partitions (ORDER),
Random walks and general coupon collector arguments (MAJORITY).

Sorting

Frank Neumann

Dependent variables - Sorting

Order and Majority are in some sense easy as the variables can be optimized independently.

What about dependent variables?

- We considered the sorting problem (first combinatorial opitmization problem analyzed for EAs).
- Many measures of sortedness work provably well for permutation based EAs (Scharnow, Tinnefeld, Wegener (2002))

Measures of Sortedness

- $INV(\pi)$, measuring the number of pairs in correct order,[1] which is the number of pairs (i, j), $1 \leq i < j \leq n$, such that $\pi(i) < \pi(j)$,
- $HAM(\pi)$, measuring the number of elements at correct position, which is the number indices i such that $\pi(i) = i$,
- $RUN(\pi)$, measuring the number of maximal sorted blocks, which is the number of indices i such that $\pi(i+1) < \pi(i)$ plus one,
- $LAS(\pi)$, measuring the length of the longest ascending subsequence, which is the largest k such that $\pi(i_1) < \ldots < \pi(i_k)$ for some $i_1 < \ldots < i_k$,
- $EXC(\pi)$, measuring the minimal number of pairwise exchanges in π, in order to sort the sequence.

Scharnow, Tinnefeld, Wegener (2002): Polynomial upper bounds for all functions except RUN.

GP and Sorting

Algorithms:

- Tree-based approach as for ORDER.
- Inorder Parse leads to (incomplete) permutation.
- Consider the different fitness measures

Results:

- GP approach has much more difficulties as variables can block each other.
- Local optima where many variables have to be changed/deleted.
- Worst case running time increases for most measures from polynomial to exponential.

PAC Learning

Frank Neumann

PAC Learning and GP

Recently some papers appeared that study the learnability of evolutionary algorithms in the probably approximately correct (PAC) learning framework.

(series of papers of Valiant and Feldman)

Papers seem to be interesting for GP as GP is about learning functions.

Problem: They use mutation operators that are very powerful (only restriction that computation runs in polynomial time)

Question: Can we get results for simple (more realistic) mutation operators?

Our task: Learn a function $f_{\mathrm{OPT}}(x) = \sum_{i=1}^{n} w_i x_i$

where $w_i \in \{-1, 1\}$, $1 \leq i \leq n$.

Error of a function f with respect to a sample x is

$$e_x(f, f_{\mathrm{OPT}}) = |f(x) - f_{\mathrm{OPT}}(x)|,$$

Error of a function f with respect to a sample set S is

$$e_S(f, f_{\mathrm{OPT}}) = \sum_{x \in S} e_x(f, f_{\mathrm{OPT}}).$$

Goal: Find a function f that minimizes the error
(in the best case obtain f_{opt})

Algorithm for Learning

Algorithm 1: Linear GP for learning functions.

input Black-box target function f_{OPT} ;
input Sample size z;
initialization Uniformly at random choose an initial function $f = \sum_{i=1}^{n} w_i x_i$;
repeat
    Choose $i \in [n]$ uniformly at random;
    Obtain f' from f by flipping the ith weight;
    Sample a set $S \subseteq \{0,1\}^n$ of size z;
    if $e_S(f', f_{\mathrm{OPT}}) \leq e_S(f, f_{\mathrm{OPT}})$ **then** $f \leftarrow f'$
until *forever*;

Samples are chosen according to uniform distribution

Results

Exact Learning:

THEOREM 3. *If* $|S| \geq c_0 n \log(n)$, c_0 *a large enough constant, the expected learning time of Linear GP is* $O(n^2)$.

Approximative Learning:

THEOREM 4. *If* $|S| \geq c_0 n \log(n)$, c_0 *a large enough constant, the expected number of generations until the best-so-far function found by Linear GP has an expected error* $\leq \delta$ *is* $O(n \log n + n^2/\delta^2)$.

Future work / next steps:

- More complex functions
- Multi-Objective approach.
- Impact of populations and crossover
- Get input from GP practitioners about the next steps.