Evolving Distributed Algorithms with Genetic Programming

Thomas Weise, 2008-12-15, Kassel

http://www.vs.uni-kassel.de/
http://www.it-weise.de/
Contents

- Motivation
- Evolutionary Algorithms and Genetic Programming
- Problems in Genetic Programming
- Fighting Epistasis: RBGP/eRBGP
- Election
- Summary
Motivation – Distribution

- Distributed algorithm
  \[\equiv\] Algorithm involving cooperation of multiple nodes of a distributed system

- Design of distributed algorithms:
  - complexity: time, memory, messages, energy
  - reliability, resiliency/robustness, decentrality
  - cumbersome
  - error-prone

- Can we automatically create distributed algorithms?
Motivation – Algorithm Design

- Mapping of the global behavior of a distributed system to local algorithms running on its single nodes

- An approach which guaranteed success/correctness
Motivation – Natural Design

- Mapping of the global behavior of a distributed system to local algorithms running on its single nodes

- No "absolute" correctness, but tests
Evolutionary Algorithms and Genetic Programming
EAs and GP – The Basic Cycle

- Optimization algorithms based on the idea of evolution
Problems in Genetic Programming
Genetic Programming

- Evolution of real algorithms is hard

- Rugged fitness landscape
- Affinity for overfitting
- Multi-modality
- Weak causality
- High epistasis
- Test data is memorized
- Many local optima
- Small changes in individual
  - Large changes in fitness

Different parts of the solution candidate influence each other.
Problems in Genetic Programming

- Genetic operators like crossover tend to reorder instructions
- *Positional epistasis:* The order of instructions determines their semantics

```
1: i = a;
2: p = 1;
3: while(i > 0) {
4:   p = p * i;
5:   i = i - 1;
6: }
```

vs.

```
1: i = a;
2: p = 1;
3: while(i > 0) {
4:   i = i - 1;
5:   p = p * i;
6: }
```
Fighting Epistasis
Fighting Epistasis
Rule-based Genetic Programming

- Algorithm representation in RBGP

```
1: i = a;
2: p = 1;
3: while(i > 0) {
4:   p = p * i;
5:   i = i - 1;
6: }

1: (start_t=1) ∧ true ⇒ i_{t+1} = a_t
2: (start_t=1) ∧ true ⇒ p_{t+1} = 1
3: (i_t>0) ∧ true ⇒ p_{t+1} = p_t * i_t
4: (i_t>0) ∧ true ⇒ i_{t+1} = i_t-1
```
Rule-based Genetic Programming

start = 1

1. \((\text{start}_t=1) \land \text{true} \Rightarrow i_{t+1} = a_t\)
2. \((\text{start}_t=1) \land \text{true} \Rightarrow p_{t+1} = 1\)
3. \((i_t>0) \land \text{true} \Rightarrow p_{t+1} = p_t \cdot i_t\)
4. \((i_t>0) \land \text{true} \Rightarrow i_{t+1} = i_t - 1\)

start = 0

commit temporary storage: \(t \rightarrow t+1\)
Rule-based Genetic Programming

- Algorithm representation in RBGP

1: \((\text{start}_t=1) \land \text{true} \Rightarrow i_{t+1} = a_t\)

2: \((\text{start}_t=1) \land \text{true} \Rightarrow p_{t+1} = 1\)

3: \((i_t>0) \land \text{true} \Rightarrow p_{t+1} = p_t \ast i_t\)

4: \((i_t>0) \land \text{true} \Rightarrow i_{t+1} = i_t - 1\)
eRBGP

- Indirect memory access
- Arbitrary complex rules
- Tree genome

Sorting algorithm memory layout: \(i_0, i_1, \ldots, i_n, a, b, \text{len}\)

1. \( (a_t < \text{len}_t) \land ([a_t]_t < [b_t]_t) \Rightarrow [a_t]_{t+1} = [b_t]_t \)
2. \( (a_t < \text{len}_t) \land ([a_t]_t < [b_t]_t) \Rightarrow [b_t]_{t+1} = [a_t]_t \)
3. \( (b_t \geq a_t) \land (a_t < \text{len}_t) \Rightarrow a_{t+1} = a_t + 1 \)
4. \( (b_t \geq a_t) \land (a_t < \text{len}_t) \Rightarrow b_{t+1} = 0 \)
5. \( (b_t < a_t) \Rightarrow b_{t+1} = b_t + 1 \)
Evolving Distributed Algorithms
Evolving Distributed Algorithms

- Initial population
- Evaluation
- Fitness assignment
- Reproduction
- Selection
Evolving Distributed Algorithms

- create random Algorithms
- reproduction
- - simulation
- - compare with goal
- - objectives
- fitness assignment
- selection

dgpf
Evolving Distributed Algorithms

Network

Objective Function

Functional

Program Size

Aggregation

RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

Functional Program Size

Accesses

Objective Function

Functional

Program Size

Aggregation

RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid

Ring

Linear

(b_i \land a_i) \Rightarrow a_i \land id_i \land a_i

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(start \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{true}) \Rightarrow b_i \land \text{true}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

(b_i \land \text{false}) \Rightarrow b_i \land \text{false}

 RBGPNetVM

LGPNetVM

HLGP

FragletNetVM

Byzantine

Broadcast

Grid
Election
Election

- Election:
  - all nodes in a network elect one and the same node
  - all nodes know the id of this special node

- Known variants: Bully algorithm, Message extinction

- Simulation:
  - initialize all nodes with their id in one memory cell
  - run simulation for some time
  - expect the id of the selected node in another memory cell
Election – Standard Genetic Programming

- SGP: \( \text{mem}[0] \)

\[
\text{while}(-79) \{
  \text{while}(\text{mem}[1] \leq \text{mem}[0]) \{
    \text{SEND\_WORD} \text{ mem}[0]
    \text{SEND\_MESSAGE}
    \text{while}(\text{HAS\_WORD}) \{
      \text{mem}[1]=\text{RECEIVE\_WORD}
    \}
  \}
  \text{mem}[0] = \text{mem}[1]
\}
\]
Election – Linear Genetic Programming

- LGP: $g[1]$

```plaintext
function_0:
  0: $g[1] = g[0] \text{ or } l[0]$
  1: if(262155:!overflow) call 1
  2: if(13:>=0) call 0
  3: if(66054:above) call 0
  4: if(g[l[0]]:equal) call 0
  5: push $g[1]$ xor $l[l[0]]$
  6: if(l[0]:!sign) send
  7: if(l[l[1]]:!overflow) goto -5

function_1:
  3: $g[l[1]] = l[l[1]] \text{ and } g[1]$
  4: $g[1] = 235805633 - l[1]$
  5: $g[1] = g[0] \text{ and } -6439994$
  6: push not $g[0]$
  7: if($g[l[1]]:<0$) call 0
  8: exchange $l[0], g[1]$
```

-comparison
"randomized"
send msg
receive msg
update vote
Election – RBGP

- RBGP: \(a_t\)

\[
\begin{align*}
\text{false} \lor (\text{start}_t = \text{incomingMsg}_t) & \Rightarrow \text{start}_{t+1} = 1 - b_t \\
\text{false} \lor (b_t \geq a_t) & \Rightarrow a_{t+1} = a_t + \text{id}_t \\
(\text{out}_t \leq \text{start}_t) \lor (\text{id}_t \neq b_t) & \Rightarrow \text{send} \\
\text{false} \lor (a_t \neq \text{out}_t) & \Rightarrow \text{out}_{t+1} = a_t \\
(\text{id}_t = 0) \land (\text{out}_t \geq 0) & \Rightarrow \text{id}_{t+1} = \text{id}_t / b_t \\
(0 = \text{id}_t) \lor (\text{id}_t < \text{in}_t) & \Rightarrow a_{t+1} = \text{in}_t
\end{align*}
\]
Election – eRBGP

- eRBGP: $a_t$

\[
\begin{align*}
\text{id}_t \Rightarrow & \text{send} \\
[\text{incomingMsg}_t]_t \Rightarrow & \text{out}_{t+1} = \text{id}_t \\
(\text{out}_t - (\text{incomingMsg}_t \lor [a_t]_t)) \Rightarrow & a_{t+1} = \text{id}_t \\
(\text{in}_t / \text{id}_t) \Rightarrow & \text{id}_{t+1} = \text{in}_t
\end{align*}
\]
Summary
Summary

- Designing distributed algorithms is cumbersome
- Evolving distributed algorithms is also cumbersome but possible
  - Design of simulation
    useful components provided
  - Design of objective functions
    non-functional objectives provided
  - Evolution takes long, often needs to be repeated
    parameter settings thoroughly researched
- Chance of success depends on computational power
Summary

- Some of the problematic aspects of GP can be mitigated (to a certain degree)
- New, efficient forms of GP introduced
- Results of GP are often *adequate* but not necessarily *correct*
- Results have low human readability
- Correctness is hard to verify with common sense
Thanks for your attention!

You may ask questions now.

http://www.vs.uni-kassel.de/
http://www.it-weise.de/
tweise@gmx.de