

Thomas Weise, Michael Zapf, Kurt Geihs
{weise|zapf|geihs}@vs.uni-kassel.de
University of Kassel
Wilhelmshöher Allee 73
D-34121 Kassel
Germany

RBGP

<http://www.sigoa.org/>
<http://dgpforge.sourceforge.net/>

Rule-based Genetic Programming

BIONETICS 2007

2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems

Bio-inspired Models and Systems Track, 2007-12-10, 09.30

Radisson SAS Beke Hotel, 43. Terez krt., Budapest H-1067, Hungary

Contents

- Intro: Genetic Programming
- Problems in Genetic Programming of "real" Algorithms
- Forms of Epistasis in Genetic Programming
- (Learning) Classifier Systems
- Rule-based Genetic Programming
- Examples in Distributed Computing
- Outro: Summary

Genetic Programming

Genetic Programming is the set of evolutionary algorithms that breed programs or algorithms.

- 1958, Friedberg: learning algorithm improves a program
- 1981, Forsyth: trees representing Boolean expressions
- 1985, Cramer: integer strings encoding trees in HLL
- 1990s, Koza: full formalization of Standard (tree-based) GP
- STGP, Grammar-guided: Gads, GE, CGE, TAG³P
- Genotype-Phenotype Mappings: Cramer, BGP, GEP
- Linear Genetic Programming
- Graph-based approaches: PDGP, Cartesian GP

Problems in Genetic Programming

- Evolution of "real algorithms" is hard



rugged fitness landscape

affinity for overfitting

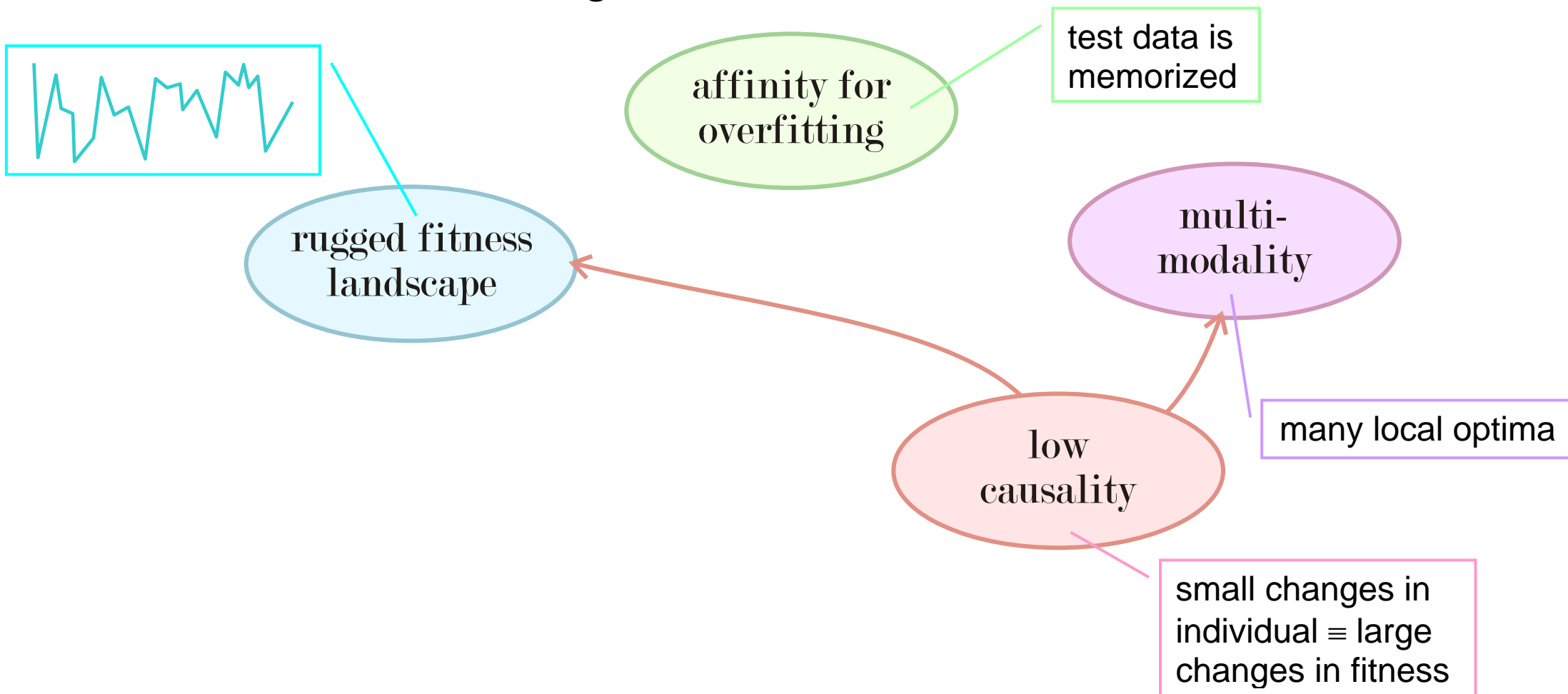
test data is memorized

multi-modality

many local optima

Problems in Genetic Programming

- Evolution of "real algorithms" is hard



Problems in Genetic Programming

- Low causality: Small changes have large impact on fitness

```

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

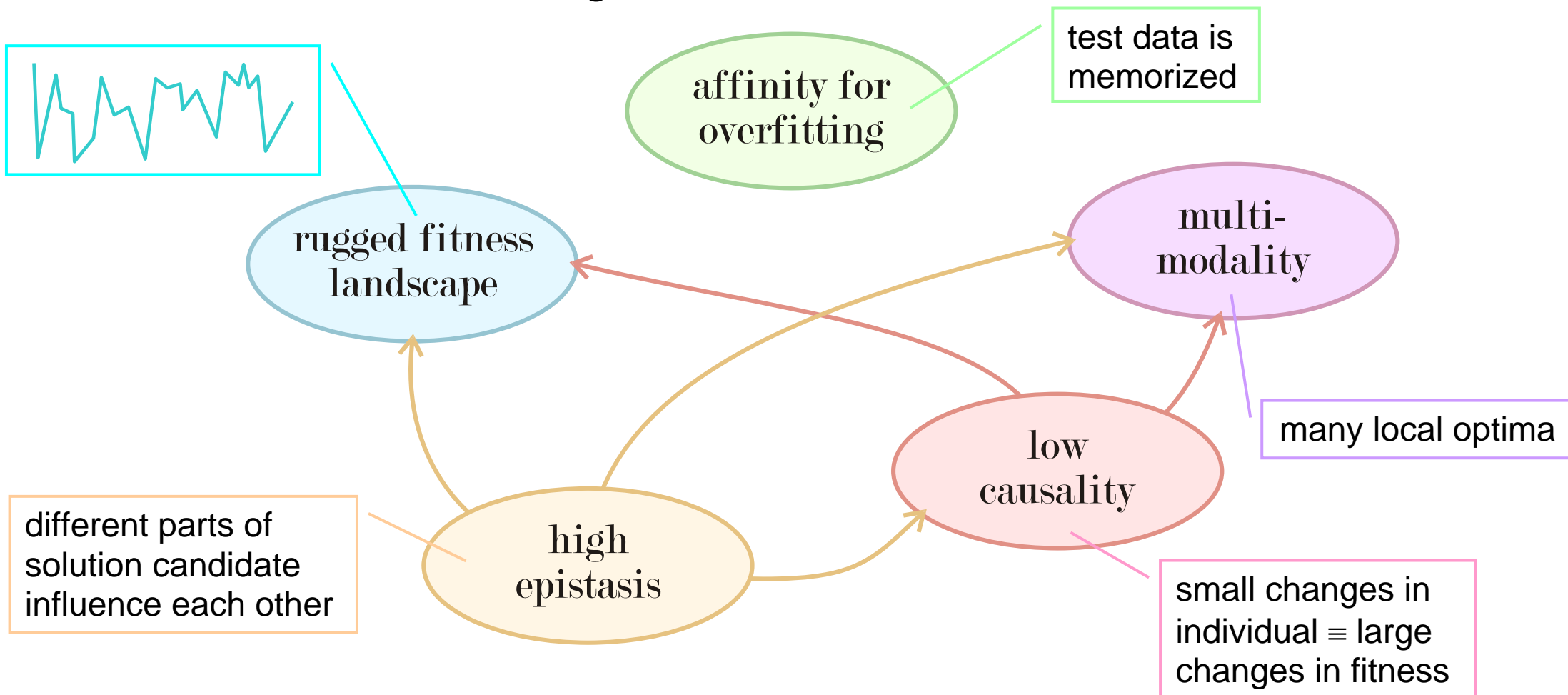
vs.

```

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

Problems in Genetic Programming

- Evolution of "real algorithms" is hard



Epistasis in Genetic Programming

- *semantic epistasis*: The **semantics** of one instruction determines the semantics of others

```

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:     p = p * i;
5:     i = i + 1;
6: }
```

Epistasis in Genetic Programming

- *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: }
```

Epistasis in Genetic Programming

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

Epistasis in Genetic Programming

- Positional epistasis in linear Genetic Programming forms

```

1: mov i , a
2: mov p, 1
3: cmp i , 0
4: l e 8
5: mul p, i
6: dec i
7: j mp 3
8:
    
```

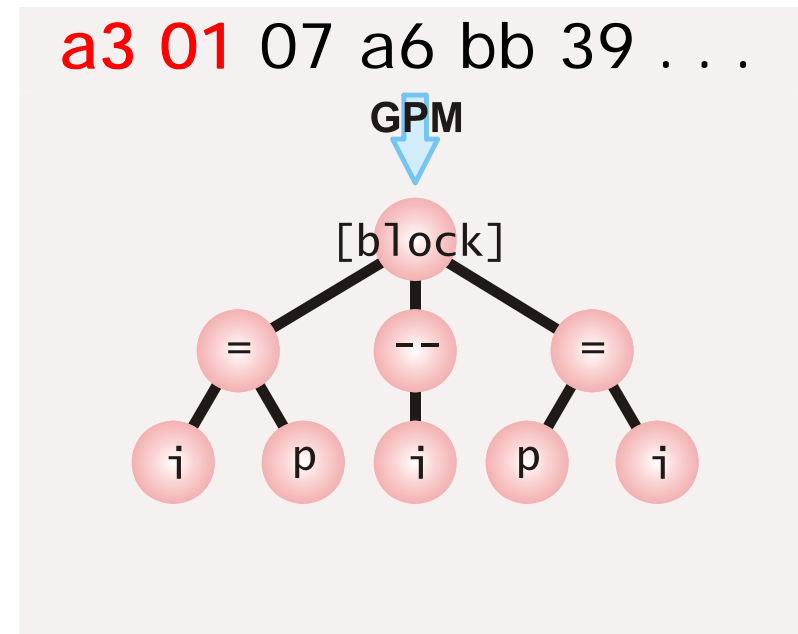
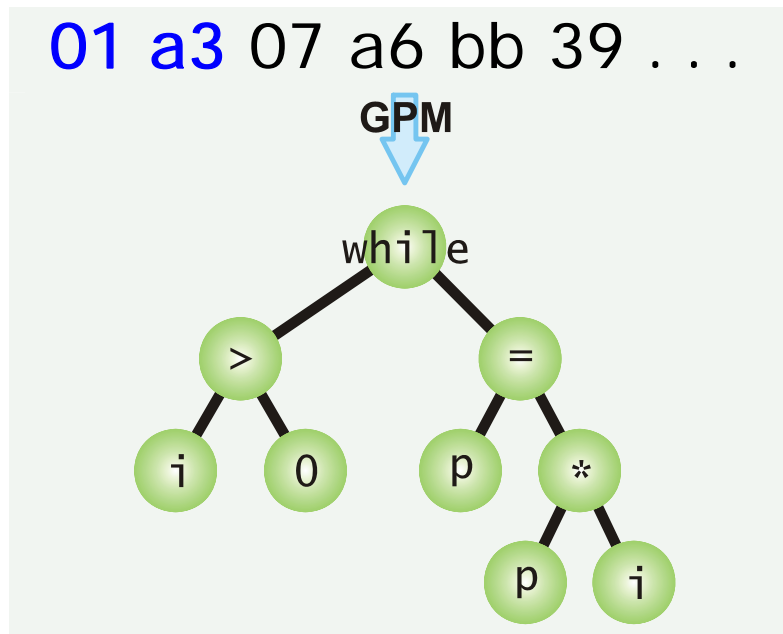
vs.

```

1: mov i , a
2: mov p, 1
3: cmp i , 0
4: mul p, i
5: l e 8
6: dec i
7: j mp 3
8:
    
```

Epistasis in Genetic Programming

- Positional epistasis in forms the involve GPMs

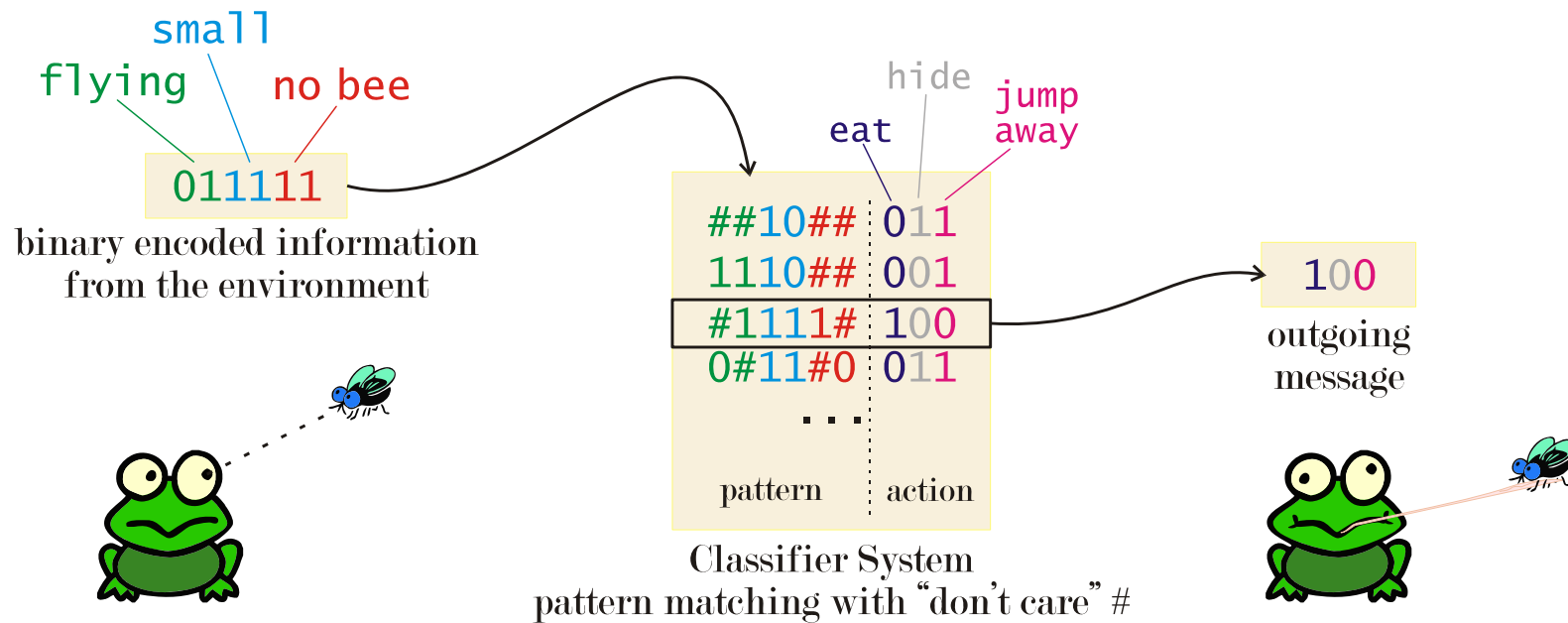


Epistasis in Genetic Programming

- All these approaches perform great, although this systemic epistasis exists.
- Can we construct a method with less epistasis?
- If so, how will it perform?

(Learning) Classifier Systems

- Developed by Holland in the late 1970s (Michigan approach)



- + learning algorithm, + GA for finding new rules
- Pittsburgh approach: sets of rules instead of single rules

Rule-based Genetic Programming

- Program \equiv set of rules
- Rule \equiv conditional part \Rightarrow action part
- Conditional part \equiv logical concatenation of two conditions

$$\text{condition}_1 \wedge \vee \text{condition}_2 \Rightarrow \text{action}$$

- Action part
 - Executed if conditional part evaluates to true
 - Changes written to temporary storage which is committed after all rules have been processed
- Rules evaluated in a cycle until no further change is possible

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: (startt=1) ∧ true ⇒
      it+1 = at
2: (startt=1) ∧ true ⇒
      pt+1 = 1
3: (it>0) ∧ true ⇒
      pt+1 = pt * it
4: (it>0) ∧ true ⇒
      it+1 = it - 1
    
```

Rule-based Genetic Programming

start = 1

1. $(start_t=1) \wedge true \Rightarrow i_{t+1} = a_t$
2. $(start_t=1) \wedge true \Rightarrow p_{t+1} = 1$
3. $(i_t > 0) \wedge true \Rightarrow p_{t+1} = p_t * i_t$
4. $(i_t > 0) \wedge true \Rightarrow i_{t+1} = i_t - 1$

evaluate
all rules

start = 0

commit temporary storage: $t \rightarrow t+1$

Rule-based Genetic Programming

- Algorithm representation in RBGP

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

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

$$3: (i_t > 0) \wedge \text{true} \Rightarrow p_{t+1} = p_t * i_t$$

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



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

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

$$3: (i_t > 0) \wedge \text{true} \Rightarrow i_{t+1} = i_t - 1$$

$$4: (i_t > 0) \wedge \text{true} \Rightarrow p_{t+1} = p_t * i_t$$

Rule-based Genetic Programming

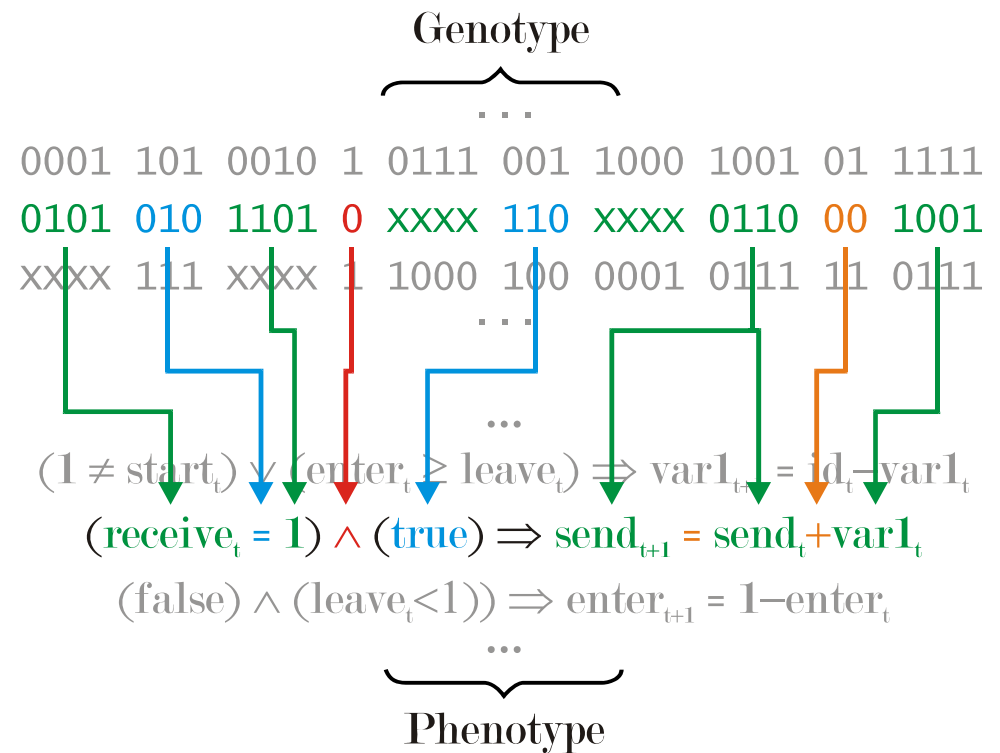
- Neutrality in RBGP

<p>1: $(start_t=1) \wedge true \Rightarrow$ $i_{t+1} = a_t$</p> <p>2: $(start_t=1) \wedge true \Rightarrow$ $p_{t+1} = 1$</p> <p>3: $(i_t > 0) \wedge true \Rightarrow$ $p_{t+1} = p_t * i_t$</p> <p>4: $(i_t > 0) \wedge true \Rightarrow$ $i_{t+1} = i_t - 1$</p>	<p>≡</p>	<p>1: $(i_t > 0) \wedge true \Rightarrow$ $i_{t+1} = i_t - 1$</p> <p>2: $(start_t=1) \wedge true \Rightarrow$ $p_{t+1} = 1$</p> <p>3: $(i_t > 0) \wedge true \Rightarrow$ $p_{t+1} = p_t * i_t$</p> <p>4: $(start_t=1) \wedge true \Rightarrow$ $i_{t+1} = a_t$</p> <p>5: $(i_t > 0) \wedge true \Rightarrow$ $p_{t+1} = p_t * i_t$</p>
---	----------	--

Rule-based Genetic Programming

- Genome: variable-length binary strings (fixed gene size)

Symbol	Encoding	Comp.	Enc.	Concat.	Enc.
0	0000, 1100	>	000	∧	0
1	0001, 1101	≥	001	∨	1
start	0010, 1110	=	010		
id	0011, 1111	≤	011		
netSize	0100	<	100		
receive	0101	≠	101		
send	0110	true	110	Action	Enc.
enter	0111	false	111	= x+y	00
leave	1000			= x-y	01
var1	1001			= x	10
var2	1010			= 1-x	11



Examples in Distributed Computing

- Election Algorithms
 - n nodes in a system chose one for special tasks
 - each node has an unique ID
 - when the algorithm terminates
 - * exactly one node has been elected
 - * all nodes know the id of this elected node
- Bully Algorithm
 - 1982, Garcia-Molina
 - node with largest ID wins election

Examples in Distributed Computing

- Election Algorithms

- minimize number of different IDs known by the nodes after the simulation (punish invalid IDs)
- minimize number of rules in Program
- minimize number of transmitted messages

- $(1 > b_t) \quad \vee \quad (b_t > a_t) \quad \Rightarrow \quad b_{t+1} = b_t + \text{start}_t$
- $(i n_t \geq a_t) \quad \vee \quad (0 == b_t) \quad \Rightarrow \quad i d_{t+1} = i n_t$
- $(i d_t > \text{out}_t) \quad \wedge \quad \text{true} \quad \Rightarrow \quad \text{send}$
- $(0 \leq 1) \quad \vee \quad \text{true} \quad \Rightarrow \quad b_{t+1} = b_t \% i n_t$
- $(i n_t \geq a_t) \quad \vee \quad \text{false} \quad \Rightarrow \quad i d_{t+1} = i n_t$
- $(i n_t \geq a_t) \quad \vee \quad \text{false} \quad \Rightarrow \quad a_{t+1} = i n_t$
- $\text{true} \quad \vee \quad \text{false} \quad \Rightarrow \quad \text{out}_{t+1} = i d_t$

Examples in Distributed Computing

- Load Balancing
 - central solution: single distributor
 - decentral distribution, for instance by P2P network
- Agent-based Load Balancing
 - each job is transported and executed by a mobile agent
 - agents can access performance data of the workstations and their neighbors
 - agents may migrate to neighboring workstations
 - agents on the same workstation can communicate via message exchange

Examples in Distributed Computing

- Agent-based Load Balancing
 - minimize variance of the number of agents per station
 - minimize number of rules
 - minimize number of migrations
 - ensure communication
- [Simulation]

Examples in Distributed Computing

- Agent-based Load Balancing

$\text{true} \wedge (\text{receive}_t > \text{remainingWork}_t) \Rightarrow \text{start}_{t+1} = \text{start}_t * \text{knownHostCount}$

$(a_t \neq \text{start}_t) \vee \text{false} \Rightarrow \text{migrate knownHostCount}$

$\text{true} \wedge (\text{send}_t = 1) \Rightarrow \text{migrate } 1$

$(1 \leq \text{start}_t) \vee (\text{receive}_t = \text{requiredWork}_t) \Rightarrow \text{send}_{t+1} = 1 - \text{send}_t$

$(1 \leq \text{start}_t) \vee (\text{receive}_t = \text{requiredWork}_t) \Rightarrow \text{send}_{t+1} = 1 - \text{send}_t$

$\text{true} \wedge (a_t = 1) \Rightarrow \text{migrate } 1$

$\text{true} \wedge (\text{remainingWork}_t = 1) \Rightarrow \text{migrate } 1$

[useless] $(\text{requiredWork}_t \neq 1) \wedge (b_t > 1) \Rightarrow \text{receive}_{t+1} = \text{receive}_t \% 0$

$(\text{remainingWork}_t \geq 0) \vee (\text{receive}_t \neq \text{send}_t) \Rightarrow \text{send}_{t+1} = \text{start}_t$

$\text{true} \wedge (1 < \text{knownHostCount}_t) \Rightarrow \text{send}(t+1) = \text{send}_t * c_t$

$(\text{remainingWork}_t \geq 0) \wedge (\text{requiredWork}_t \neq \text{send}_t) \Rightarrow \text{send}_{t+1} = \text{start}_t$

[useless] $\text{false} \vee \text{false} \Rightarrow \text{agentCount}_{t+1} = \text{agentCount}_t - \text{receive}_t$

$(\text{requiredWork}_t = \text{remainingWork}_t) \vee (\text{remainingWork}_t < \text{receive}_t) \Rightarrow \text{migrate } 1$

Summary

- RBGP
 - low-epistasis GP method in its early stage
 - lots of room for discussion
- Successfully used to grow real, distributed algorithms
- Future work:
 - experiments/comparisons with other approaches

Thanks for your kind attention.

Questions?

Our work is available at

<http://www.sigoa.org/>

<http://www.it-weise.de/projects/book.pdf>

<http://dgpf.sourceforge.net/>

Thomas Weise

Distributed Systems Group
FB 16

University of Kassel
Wilhelmshöher Allee 73
D-34121 Kassel
Germany

Tel: +49-(0)561-804 62 83

Fax: +49-(0)561-804 62 77

weise@vs.uni-kassel.de

Examples in Distributed Computing

- Distributed Critical Section Problem
 - sharing resources in a distributed system
 - only one node at a time is allowed to access the resource (*mutual exclusion* needed)
 - solved in the late 1970s: Lamport

- Distributed Critical Section Problem
 - minimize violations of critical section
 - maximize utilization of critical section
 - minimize rules
 - minimize communication

Epistasis

- 1909, Bateson, *Mendel's Principles of Heredity*, Cambridge University Press
 - one gene suppresses the phenotypical expression of another gene
- 1935, Lush, *Journal of Dairy Science*:
 - interaction between genes is epistatic if effect on fitness from altering one gene depends on allelic state of other genes
- 1987, Kauffman, Kauffman's NK fitness landscape
- Our usage: The change in one property of a solution candidate, for example due to a reproduction operation, can lead to a change in another, (seemingly) unrelated property

Differences between RBGP and LCS

- LCS are/can be Turing complete but expressing complex operations like multiplication is complicated
- RBGP has explicit, variable-based storage
- RBGP has more explicit semantics and targets another problem domain
- LCS use learning algorithms, RBGP purely based on objective functions
- LCS are online/offline learners, RBGP is purely offline

More Complex Examples for RBGP

- Up to a certain degree of complexity, RBGP representation is straightforward

```

1: i = a;
2: p = 1;
3: while(i > 0) {
4:     if(i < p)
5:         p = p * i;
6:     i = i - 1;
7: }
    
```



```

1: (startt=1) ∧ true ⇒
      it+1 = at
2: (startt=1) ∧ true ⇒
      pt+1 = 1
3: (it>0) ∧ (it<pt) ⇒
      pt+1 = pt * it
4: (it>0) ∧ true ⇒
      it+1 = it - 1
    
```

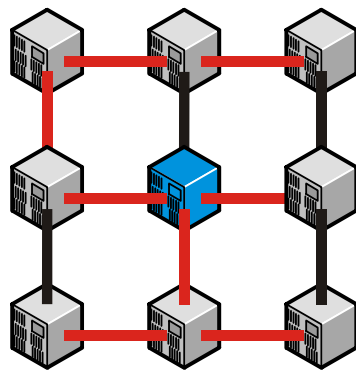
More Complex Examples for RBGP

- More complex control flows can be expressed by using additional variables.
- This is not very elegant, so we need additional research.
- Possibility:
 - Arbitrary long conditional parts
 - Genetic Operators work on RBGP representation

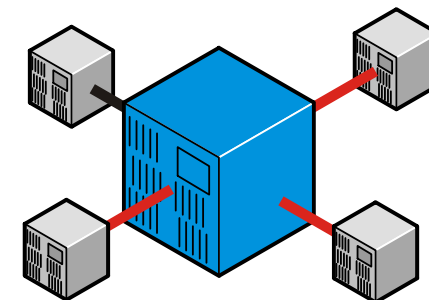
GP of Distributed Algorithms

- Transform a wanted global behavior of a network into algorithms locally run by the single nodes

functional objectives
non-functional objectives



Network Simulation



Program in
RBGP Syntax

Comparison with other Methods

- RBGP solved some problems we couldn't solve with other GP forms
- For a fair comparison we need however
 - run different program representations on same VMs
 - exactly the same network simulations
 - exactly the same objective functions
 - * non-functional objectives like code size?
- Development of such an environment currently in progress

How to Fight Overfitting

- Usage of many test cases
 - not necessary an improvement
- Complete Randomization for all tests
 - objective values are not comparable anymore
- Same test case for all individ., change in each generation
 - decrease in overfitting
 - however, overfitted individuals survive a few generations with bad fitness
- Same test case **s** for all individ., change in each generation
 - significant decrease in overfitting

```

@inproceedings{WZG2007RBGP,
author      = {Thomas Weise and Michael Zapf and Kurt Geihs},
title       = {Rule-based Genetic Programming},
booktitle   = {Proceedings of BIONETS 2007, 2nd International Conference on
Bio-Inspired Models of Network, Information, and Computing Systems},
ISBN        = {978-963-9799-05-9},
publisher   = {Institute for Computer Sciences, Social-Informatics and
Telecommunications Engineering (ICST), IEEE, ACM},
year        = {2007},
month       = {dec # {-10}},
affiliation = {University of Kassel},
location    = {Radisson SAS Beke Hotel, 43. Terez krt., Budapest H-1067, Hungary},
note        = {The work is online available at
http://www.it-weise.de/documents/index.html#WZG2007RBGP. \\
The paper can be downloaded at
http://www.it-weise.de/documents/files/WZG2007RBGP.pdf. \\
The presentation can be downloaded at
http://www.it-weise.de/documents/files/WZG2007RBGP\_slides.pdf. \\
The demonstration can be downloaded at
http://www.it-weise.de/documents/files/WZG2007RBGP\_demo.jar. \\
Contact Thomas Weise at tweise@gmx.de or http://www.it-weise.de/.},
abstract    = {In this paper we introduce a new approach for Genetic Programming,
called rule-based Genetic Programming, or RBGP in short. A program
evolved in the RBGP syntax is a list of rules. Each rule consists of two
conditions, combined with a logical operator, and an action part. Such
rules are independent from each other in terms of position (mostly) and
cardinality (always). This reduces the epistasis drastically and hence,
the genetic reproduction operations are much more likely to produce good
results than in other Genetic Programming methodologies. In order to
verify the utility of our idea, we apply RBGP to a hard problem in
distributed systems. With it, we are able to obtain emergent algorithms
for mutual exclusion at a distributed critical section.},
contents    = {* Introduction\\
* Related Work\\
- Epistasis in Genetic Programming\\
* Rule-based Genetic Programming\\
- Genotype and Phenotype\\
- New Dimensions of Independence\\
+ Positional Independence\\
+ Cardinality Independence\\
+ Neutrality\\
+ Crossover and Mutation\\
* The Distributed Critical Section\\
- Prerequisites\\
- Objective Functions\\
- Results\\
- Correctness and Discussion\\
* Conclusions and Future Work},
keywords    = {Rule-based Genetic Programming, Genetic Programming, GP, Distributed
Systems, Critical Section, Epistasis, Neutrality, Learning Classifier
Systems},
language    = {en},
url         = {http://www.it-weise.de/documents/index.html#WZG2007RBGP}
}

```