Pattern-Guided Genetic Programming

Krzysztof Krawiec[1] Jerry Swan[2]

[1]Institute of Computing Science
Poznan University of Technology, Poland

[2]Computing Science and Mathematics
University of Stirling, Scotland

July 9, 2013

Objective

Synthesize a program that exhibits *desired behavior.*

- Desired behavior = mapping input onto desired output.
- Search driver: fitness function.
  - Measures program's conformance with the desired behavior.

Only the **final effect** of program execution (program output) matters.

Motivation

The problem

Programs that arrive at correct partial results can end up with incorrect output.

Example task

Design an algorithm that calculates the median of an array of numbers.

Solution:

1. Sort the array
2. Return the central element

The problem

Programs that arrive at correct partial results can end up with incorrect output.

Example task

Design an algorithm that calculates the median of an array of numbers.

Solution:
1. Sort the array
2. Return the central element

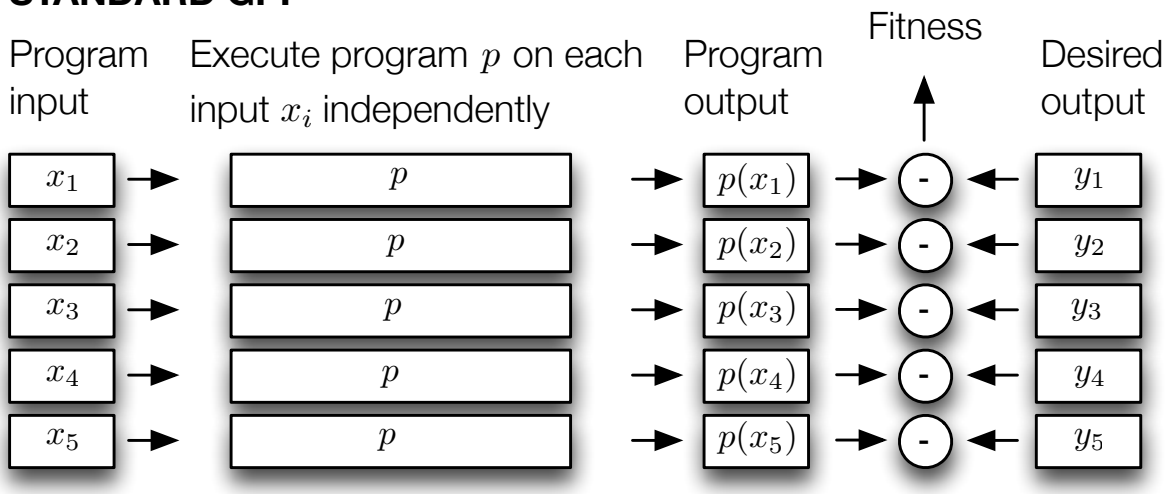
Question

- The sorted array is the *desired intermediate effect.*
- Human programmers can anticipate such effects.

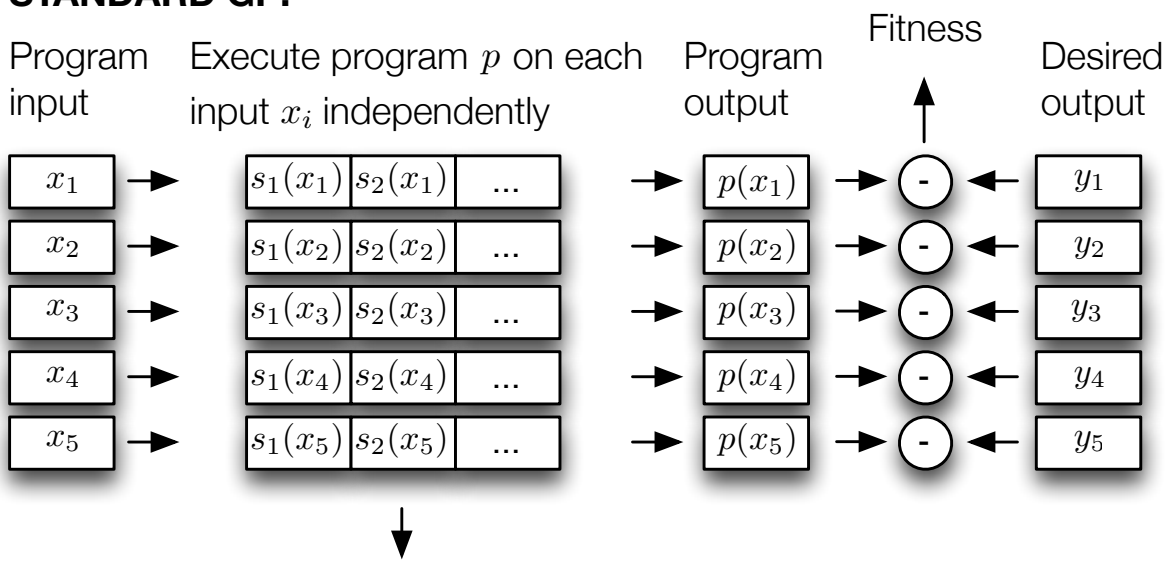
Question

Can we help evolution in (i) detecting and (ii) promoting the desired intermediate computation states?

STANDARD GP:


Program input
Execute program p on each input x_i independently
Program output
Fitness
Desired output
x_1
x_2
x_3
x_4
x_5
p
p
p
p
p
$p(x_1)$
$p(x_2)$
$p(x_3)$
$p(x_4)$
$p(x_5)$
-
-
-
-
-
y_1
y_2
y_3
y_4
y_5

STANDARD GP:



PANGEA: **1.** Use the intermediate states of program execution to build a machine learning dataset

f_1	f_2	f_3

2. Build a ML classifier

3. Use the characteristics of the classifier to augment fitness

Program execution in Push

Framework: PushGP (Spector, Keijzer, et al. 2004):

- Stack-based
- Separate stacks for code and different data types
- Program input: the initial stack state
- Program output: the stack state left by the program after completion

Step	EXEC	INT	BOOL
0	(* + <)	(1 3 4 5)	( )
1	(+ <)	(3 4 5)	( )
2	(<)	(7 5)	( )
3	( )	( )	(F)

(Stack tops on the left)

Program execution in Push

		Fitness case 1		Fitness case 2	
Step	EXEC	INT	BOOL	INT	BOOL
0	(* + <)	(1 3 4 5)	( )	(2 2 4 2)	( )
1	(+ <)	(3 4 5)	( )	(4 4 2)	( )
2	(<)	(7 5)	( )	(8 2)	( )
3	( )	( )	(F)	( )	(F)

Program execution in Push

		Fitness case 1		Fitness case 2		Fitness case 3	
Step	EXEC	INT	BOOL	INT	BOOL	INT	BOOL
0	(* + <)	(1 3 4 5)	( )	(2 2 4 2)	( )	(1 2 3 8)	( )
1	(+ <)	(3 4 5)	( )	(4 4 2)	( )	(2 3 8)	( )
2	(<)	(7 5)	( )	(8 2)	( )	(5 8)	( )
3	( )	( )	(F)	( )	(F)	( )	(T)

Step	EXEC	INT	BOOL	INT	BOOL	INT	BOOL
0	(* + <)	(1 3 4 5)	( )	(2 2 4 2)	( )	(1 2 3 8)	( )
1	(+ <)	(3 4 5)	( )	(4 4 2)	( )	(2 3 8)	( )
2	(<)	(7 5)	( )	(8 2)	( )	(5 8)	( )
3	( )	( )	(F)	( )	(F)	( )	(T)

Step 1: Gather trace features in a dataset:

Case	I_0	I_1	I_2	I_3	B_0	B_1	B_2	B_3
1	1	3	7	0	?	?	?	F
2	2	4	8	0	?	?	?	F
3	1	2	5	0	?	?	?	T

- Fitness cases ~ examples (rows)
- Trace elements ~ features (columns)

Step 2: Extend the dataset with decision attribute:

Case	I_0	I_1	I_2	I_3	B_0	B_1	B_2	B_3	**Desired output**
1	1	3	7	0	?	?	?	F	**T**
2	2	4	8	0	?	?	?	F	**F**
3	1	2	5	0	?	?	?	T	**F**

Step 3: Train a classifier on the dataset

Step 4: Use classifier size and classifier error to augment fitness

$$\textit{fitness} = \textit{program error} \times \textit{classifier size} \times \textit{classifier error}$$

Motivation

The classifier discovers the execution states *related* to the desired output.

The experiment

Acronym	Desired output	Output type
AEQ	All elements equal?	BOOL
CNF	Top element deeper in the stack?	BOOL
ISO	Is sorted?	BOOL
MAJ	Are more than half elements 1s?	BOOL
COZ	How many zeroes?	INT
MAX	The maximum	INT

- Program input: list of up to m integers placed on INT stack
  - Small tasks: $m = 3$, big tasks: $m = 4$
- Fitness cases: All combinations of up to m integers $\in [0, m-1]$

Success rate (100 runs of each method and task)

Method	AEQ3	CNF3	COZ3	ISO3	MAJ3	MAX3	AEQ4	CNF4	COZ4	ISO4	MAJ4	MAX4
GP	0.93	0.17	0.47	0.20	0.46	0.07	0.32	0.00	0.02	0.00	0.20	0.00
PANGEA	**1.00**	**0.61**	**0.76**	**0.76**	**0.97**	**0.22**	**1.00**	0.00	**0.04**	0.00	**0.70**	0.00

Impact of fitness components

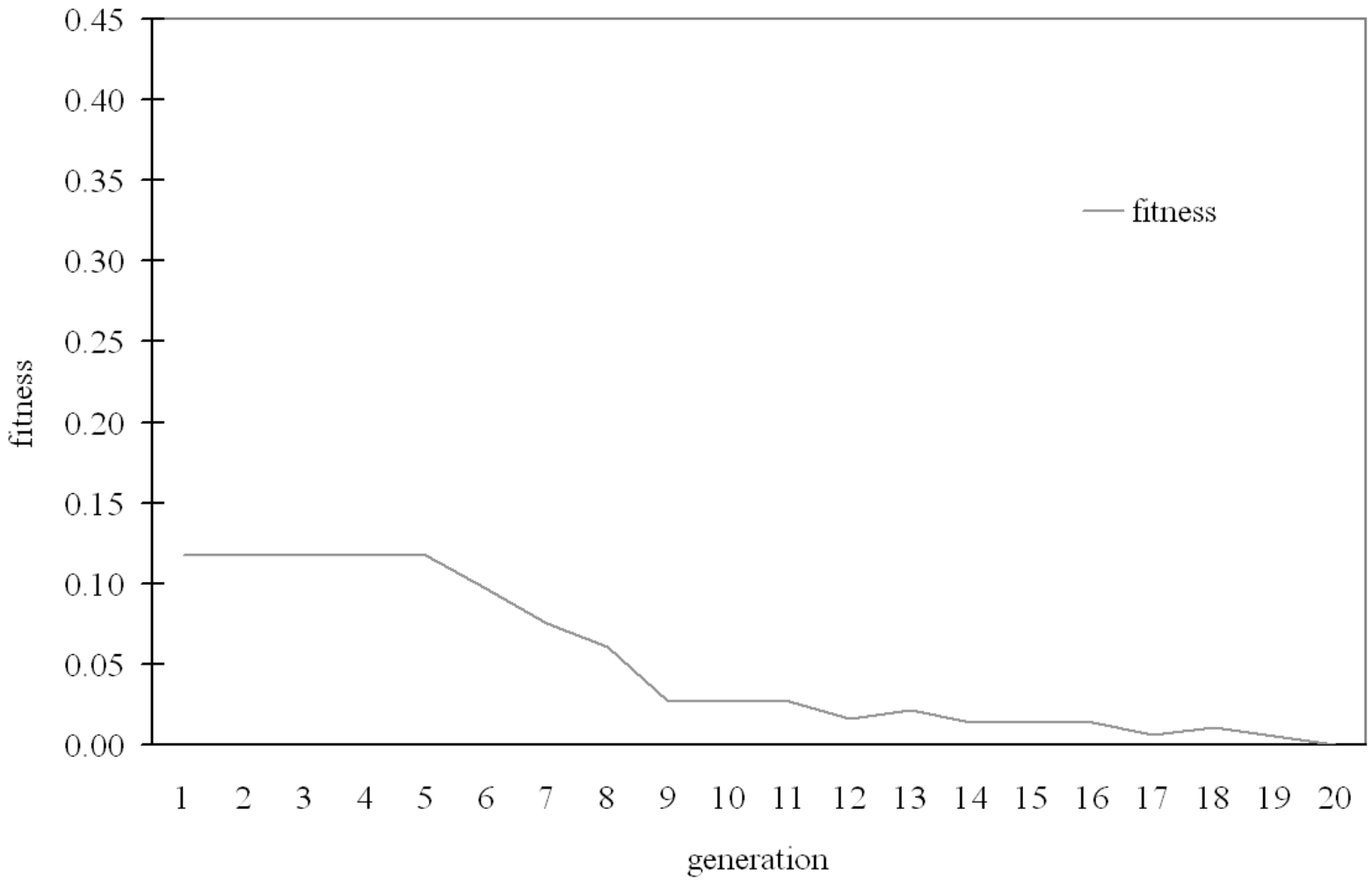
$$fitness = program\ error \times classifier\ size \times classifier\ error$$

	program error	*classifier size*	*classifier error*	*Rank*
GP	✓	✗	✗	3.67
PANGEA	✓	✓	✓	**1.92**
	✗	✓	✓	2.50
	✓	✗	✓	4.04
	✓	✓	✗	2.88

Friedman test: positive ($p \approx 0.0005$)

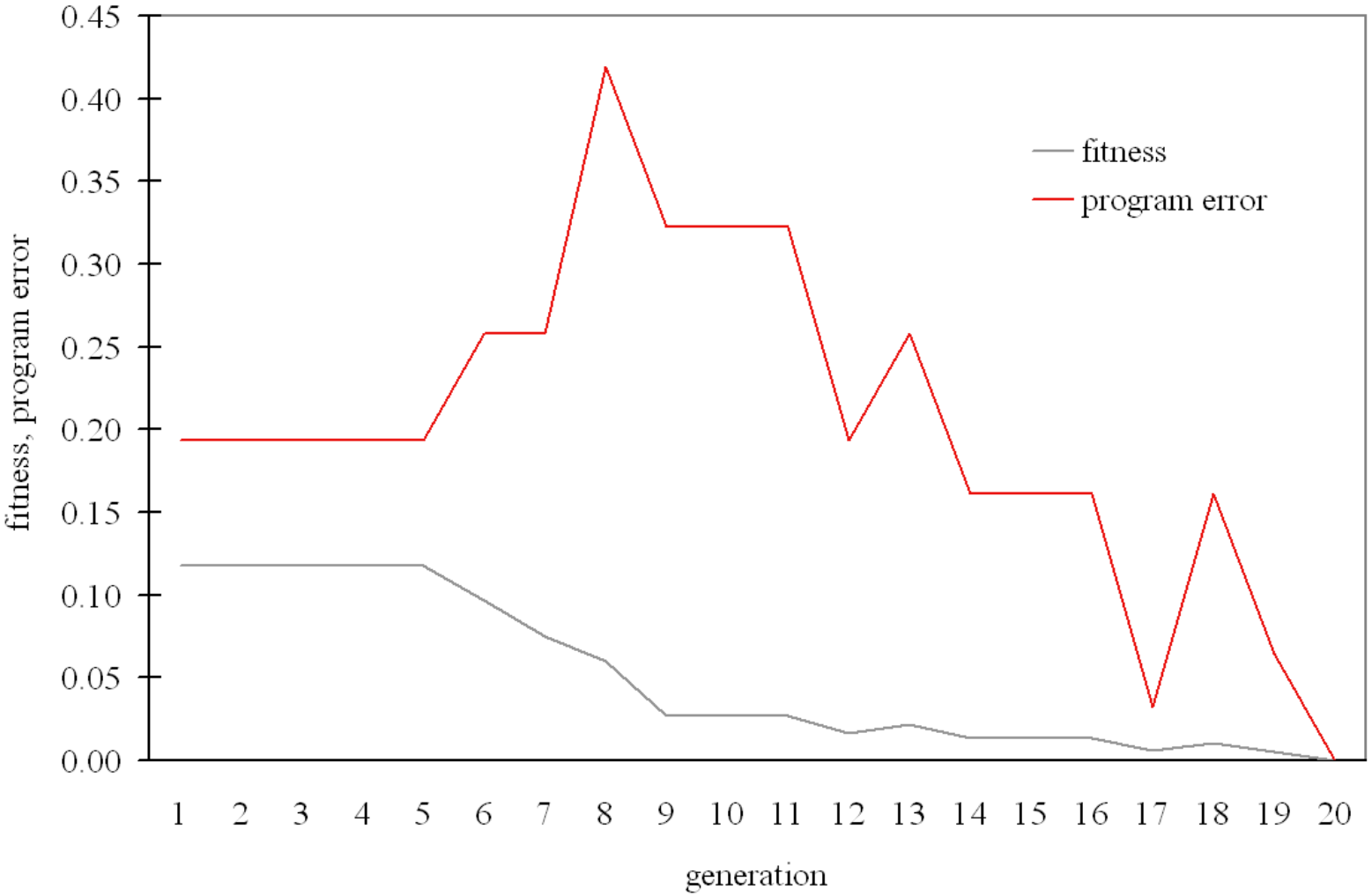
Exemplary run

Fitness of the best-of-generation individuals (a MAJ4 run):



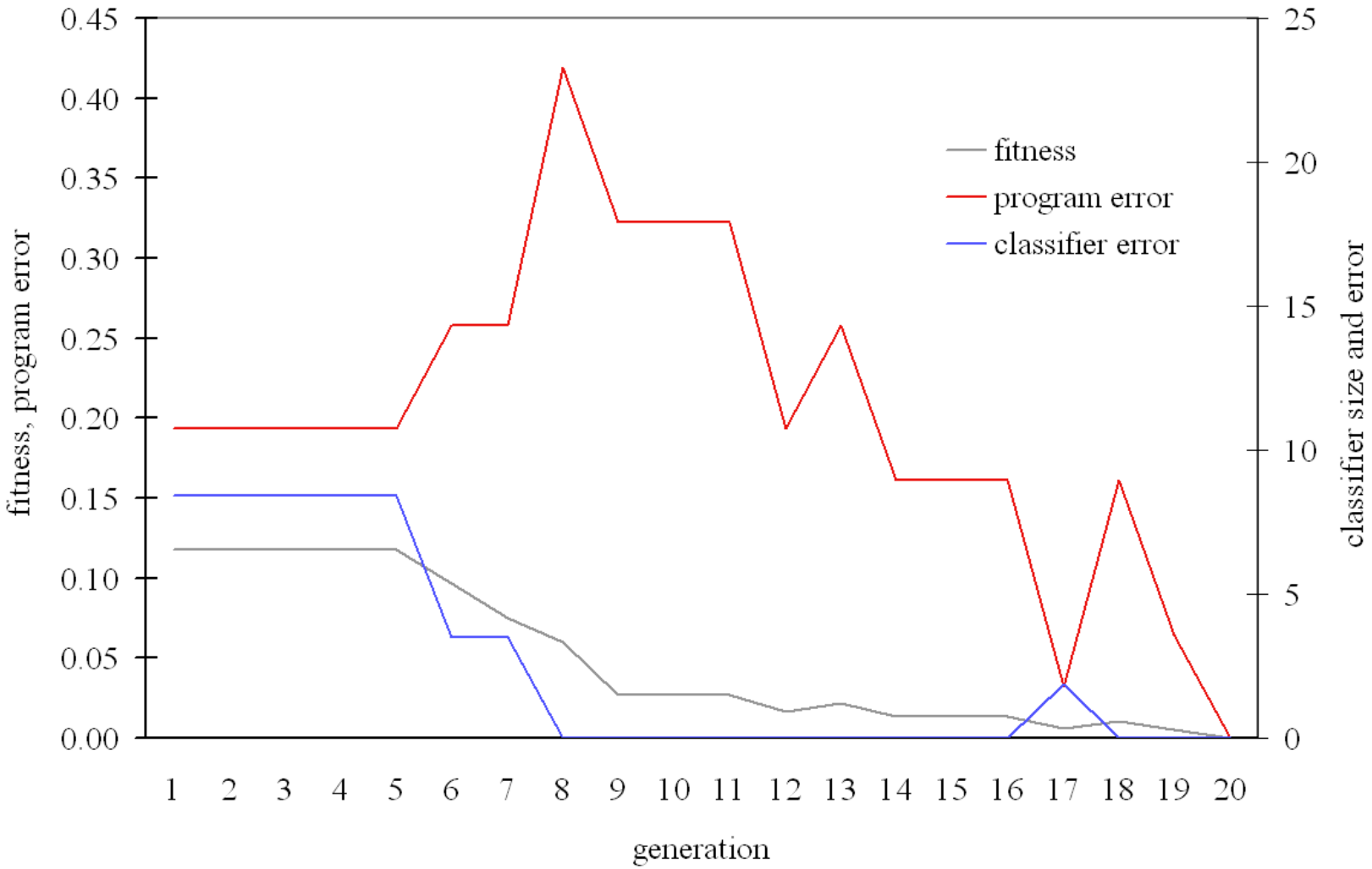
Exemplary run

Fitness components of the best-of-generation individuals (a MAJ4 run):



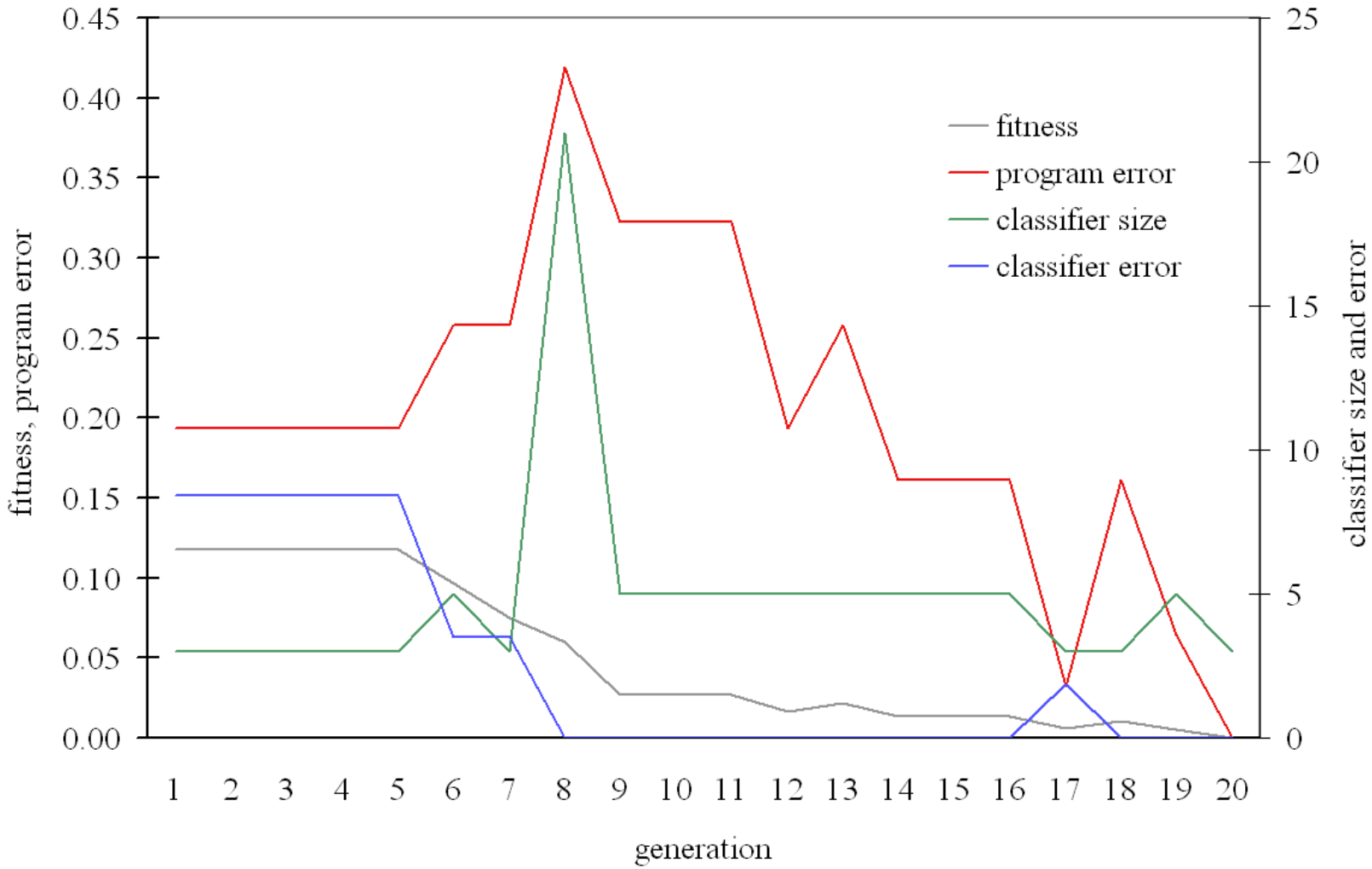
Exemplary run

Fitness components of the best-of-generation individuals (a MAJ4 run):



Exemplary run

Fitness components of the best-of-generation individuals (a MAJ4 run):



Conclusions

- Useful intermediate results of program execution can be detected and exploited.
- **Patterns** emerge in program execution that can be exploited to solve the task.
- Original objective function not necessarily the best search driver.

euroGP 2014

17th European Conference on Genetic Programming
23-25 April 2014, Baeza, Spain
www.evostar.org

The Conference

EuroGP is the premier annual conference on Genetic Programming, the oldest and the only meeting worldwide devoted specifically to this branch of evolutionary computation. It is always a very enjoyable event attract-

Venue

Baeza is located in the southernmost region of Spain, Andalusia. It is a charming small town with an incredible Renaissance architectural heritage, classified as a UNESCO

Areas of Interest

Innovative applications of GP
Theoretical developments
GP performance and behaviour
Fitness landscape analysis of GP
Algorithms, representations and operators

Where does it fail?

Example: factorial.

- Fitness cases: $(n, n!)$: (1,1), (2,2), (3,6), (4,24), ...
- Most features collected from traces highly correlated with n
- One-to-one correspondence between the inputs and the desired outputs.
- Trivial tree that perfectly maps inputs to outputs.
- The classification error $e(p) = 0$ for most individuals in the run
- The MDL-related components (*size*, *error*) cease to augment the fitness function.

Conclusion:

- PANGEA is helpful the most for problems where nontrivial patterns engage *multiple* features.
- The number of possible output values (#decision classes) is an important factor.

Feature extraction

- Two types of features: backward- and forward-aligned.
- Total number of features:
  - 3 stacks $\times$ 2 alignment methods $\times$ 3 steps $+ 1 = 37$

The experiment

- PANGEA vs. standard PushGP (STD)
  - 1000 individuals,
  - 100 generations,
  - mutation : crossover : reproduction : 2 : 7 : 1,
  - tournament size 7.
  - Programs terminated after 150 steps
- INTEGER, BOOLEAN, and CODE stacks
- Instruction set

---

boolean: and or xor = not dup flush pop rot stackdepth swap
integer: + - * / % < = > dup flush pop rot stackdepth swap
exec: = dup flush pop rot stackdepth swap if do*count do*range do*times
boolean.frominteger integer.fromboolean integer.erc
true false 0 1 2

The features:

- Gathered from the last 3 steps before program termination.
- Additional nominal features:
  - the top element of CODE stack (3 features)
  - the number of instructions executed (1 feature).
- Total number of features: 37
- Decision tree (unprunned): C4.5 (J48 from WEKA)

Impact of fitness components

Success rate (100 runs of each method and task)

Method	AEQ3	CNF3	COZ3	ISO3	MAJ3	MAX3	AEQ4	CNF4	COZ4	ISO4	MAJ4	MAX4	*Rank*
STD	0.93	0.17	0.47	0.20	0.46	0.07	0.32	0.00	0.02	0.00	0.20	0.00	3.67
PANGEA	**1.00**	**0.61**	**0.76**	**0.76**	0.97	**0.22**	**1.00**	0.00	0.04	0.00	**0.70**	0.00	**1.92**
PANGEA-E	**1.00**	0.18	0.42	0.46	**1.00**	0.00	**1.00**	0.00	0.02	0.00	0.67	0.00	2.50
PANGEA-M	**1.00**	0.32	0.40	0.67	**1.00**	0.22	0.99	0.00	**0.07**	0.00	0.50	0.00	4.04
PANGEA-X	0.97	0.12	0.46	0.03	0.46	0.01	0.42	0.00	0.00	0.00	0.09	0.00	2.88

The outlooks

- The potential in detecting and exploiting *any* aspects of the search process (c.f. hyper-heuristics)
- Mining for **patterns** in:
  - problem description,
  - genotype-to-phenotype mapping,
  - solution-state trajectory,
  - algorithm-state trajectory,
  - operator-sequence trajectory.
  - + combinations.

Step 2: Extend the dataset with decision attribute:

Case	I_0	I_1	I_2	I_3	B_0	B_1	B_2	B_3	**Desired output**
1	1	3	7	0	?	?	?	F	**T**
2	2	4	8	0	?	?	?	F	**F**
3	1	2	5	0	?	?	?	T	**F**

Step 3: Train a classifier on the dataset

Step 4: Use classifier size and classifier error to augment fitness

$$f(p) = \overbrace{err(p)}^{\textit{program error}} \times \overbrace{\log_2(l(p)+1)}^{\textit{classifier size}} \times \overbrace{\frac{e(p)+1}{n+1}}^{\textit{classifier error}}.$$

Motivation

The classifier discovers the intermediate execution states that *relate* to the task.

Thank you.

Acknowledgments:

DAASE: Dynamic Adaptive Automated Software Engineering

(EPSRC: EP/J017515/1)