On the Automatic Design of a Representation for Grammar-based Genetic Programming

Eric Medvet and Alberto Bartoli

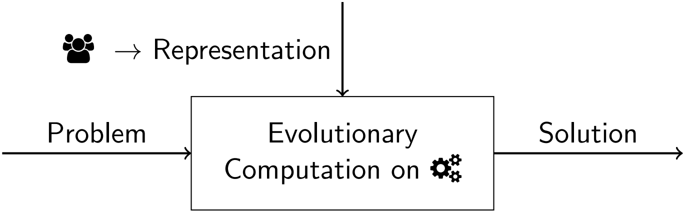
Department of Engineering and Architecture
University of Trieste
Italy

EuroGP, 5/4/2018, Parma (Italy)

http://machinelearning.inginf.units.it

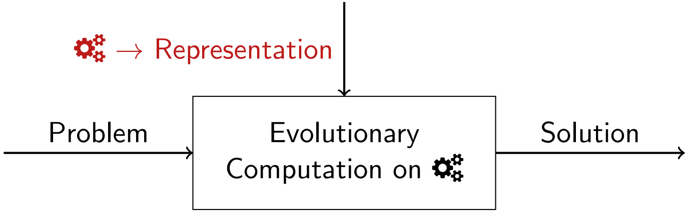
What we know

We can use a machine for obtaining automatically a good solution for "any" problem at hand, by means of an Evolutionary Computation.



What we know

We can use a machine for obtaining automatically a good solution for "any" problem at hand, by means of an Evolutionary Computation.



Can we obtain automatically a good representation too?

Table of Contents

Then and now

2007 (30 years perspective): "perhaps the most difficult and least understood area of EA design is that of adapting its internal representation."[1]

[1] De Jong, "Parameter setting in EAs: a 30 year perspective", 2007.

[2] Spector, "Introduction to the peer commentary special section on "On the Mapping of Genotype to Phenotype in Evolutionary Algorithms" by Peter A. Whigham, Grant Dick, and James Maclaurin", Sept. 2017.

Then and now

2007 (30 years perspective): "perhaps the most difficult and least understood area of EA design is that of adapting its internal representation."[1]

2017: "How should the representations that are used in evolutionary algorithms, on which variation and selection act, be chosen and justified?"[2]

[1] De Jong, "Parameter setting in EAs: a 30 year perspective", 2007.

[2] Spector, "Introduction to the peer commentary special section on "On the Mapping of Genotype to Phenotype in Evolutionary Algorithms" by Peter A. Whigham, Grant Dick, and James Maclaurin", Sept. 2017.

Then and now

2007 (30 years perspective): "perhaps the most difficult and least understood area of EA design is that of adapting its internal representation."[1]

2017: "How should the representations that are used in evolutionary algorithms, on which variation and selection act, be chosen and justified?"[2]

Large debate, many arguments/POVs, weak agreement...

[1] De Jong, "Parameter setting in EAs: a 30 year perspective", 2007.

[2] Spector, "Introduction to the peer commentary special section on "On the Mapping of Genotype to Phenotype in Evolutionary Algorithms" by Peter A. Whigham, Grant Dick, and James Maclaurin", Sept. 2017.

Choose/design and justify: so far

- How to choose/design?
- How to justify?

Choose/design and justify: so far

- How to choose/design? $\rightarrow$ inspiration by Nature/guidelines
- How to justify?

Choose/design and justify: so far

- How to choose/design? $\rightarrow$ inspiration by Nature/guidelines
- How to justify? $\rightarrow$ study its properties

Choose/design and justify: so far

- How to choose/design? $\rightarrow$ inspiration by Nature/guidelines
- How to justify? $\rightarrow$ study its properties

Outcome:

☹ *Nature* not inspiring enough

☹ no wide agreement on practical *guidelines*

Choose/design and justify: so far

- How to choose/design? → inspiration by Nature/guidelines
- How to justify? → study its properties

Outcome:

☹ *Nature* not inspiring enough

☹ no wide agreement on practical *guidelines*

☺ some agreement on which *properties* matter

- variational inheritance principle

Idea!

Design automatically the representation
aiming at obtaining good properties

[3] Spector, "Introduction to the peer commentary special section on "On the Mapping of Genotype to Phenotype in Evolutionary Algorithms" by Peter A. Whigham, Grant Dick, and James Maclaurin", Sept. 2017.

Idea!

Design automatically the representation
aiming at obtaining good properties

Sound! Already conjectured and partially attempted:

- "design guidelines [...] may be met not through clever engineering [...], but through the action of the evolutionary process itself"[3]
- meta-evolution, self-adaptation, hyper-heuristic

[3] Spector, "Introduction to the peer commentary special section on "On the Mapping of Genotype to Phenotype in Evolutionary Algorithms" by Peter A. Whigham, Grant Dick, and James Maclaurin", Sept. 2017.

Idea!

Design automatically the representation
aiming at obtaining good properties

Sound! Already conjectured and partially attempted:

- "design guidelines [...] may be met not through clever engineering [...], but through the action of the evolutionary process itself"[3]
- meta-evolution, self-adaptation, hyper-heuristic

We did it for a challenging case: GE!

[3] Spector, "Introduction to the peer commentary special section on "On the Mapping of Genotype to Phenotype in Evolutionary Algorithms" by Peter A. Whigham, Grant Dick, and James Maclaurin", Sept. 2017.

Why Grammatical Evolution (GE)?

- Great practical interest
- Inspired by Nature
- Challenging, widely studied **indirect representation**

Why Grammatical Evolution (GE)?

- Great practical interest
  - works on any problem with solutions described by a CFG
  - trendy!
- Inspired by Nature
- Challenging, widely studied **indirect representation**

Why Grammatical Evolution (GE)?

- Great practical interest
  - works on any problem with solutions described by a CFG
  - trendy!
- Inspired by Nature
  - (at least loosely)
- Challenging, widely studied **indirect representation**

Why Grammatical Evolution (GE)?

- Great practical interest
  - works on any problem with solutions described by a CFG
  - trendy!
- Inspired by Nature
  - (at least loosely)
- Challenging, widely studied **indirect representation**
  - familiar bit string genotype
  - many experimental studies on properties: redundancy, locality, uniformity
  - many representation variants: GE, πGE, HGE/WHGE (and SGE)

Table of Contents

What we need

Problem: evolving a representation with good properties for GE

We need to:

1. define the representation (of grammar-based representations)
2. define the fitness function

Representation of grammar-based representations

Bit string grammar-based representation, a *mapping function* which:

- maps any bit string to a valid string w.r.t. the user-provided CFG
- in a finite number of steps

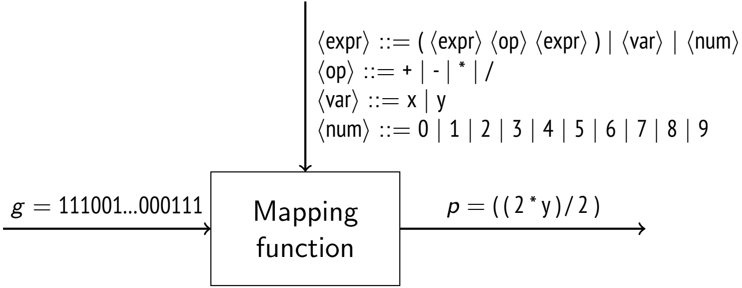
$\langle \text{expr} \rangle$ ::= ($\langle \text{expr} \rangle$ $\langle \text{op} \rangle$ $\langle \text{expr} \rangle$) | $\langle \text{var} \rangle$ | $\langle \text{num} \rangle$
$\langle \text{op} \rangle$ ::= + | - | * | /
$\langle \text{var} \rangle$ ::= x | y
$\langle \text{num} \rangle$ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

g = 111001...000111 → Mapping function → p = ((2 * y) / 2)

Representation of grammar-based representations

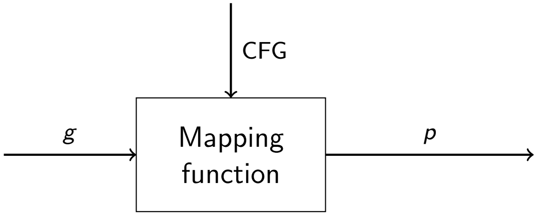
Bit string grammar-based representation, a *mapping function* which:

- maps any bit string to a valid string w.r.t. the user-provided CFG
- in a finite number of steps

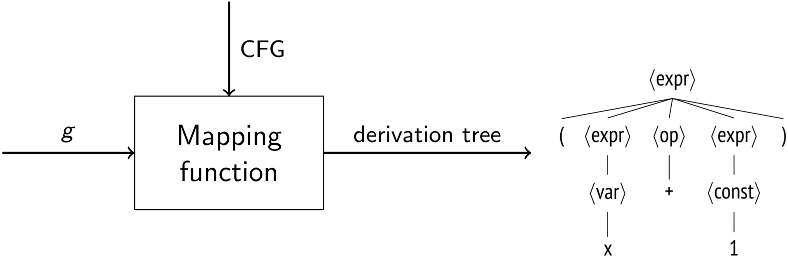


CFGs may be (and usually are) recursive → infinite languages!

Mapping function template

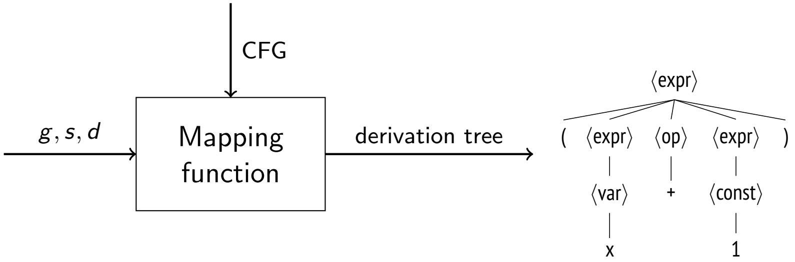


Mapping function template



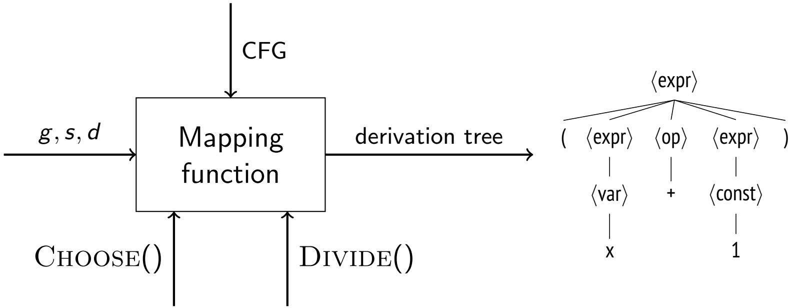
- Returns a derivation tree

Mapping function template



- Returns a derivation tree
- Recursive

Mapping function template



- Returns a derivation tree
- Recursive
- Actual behavior depends on Choose() and Divide()

Mapping function template: details

A recursive $\textsc{Map}(g, s, d)$ function (inspired by WHGE):

1. consider derivation options for symbol s
2. if depth $d > d_{\mathsf{max}}$
    1. choose a predefined option

    else
    1. choose option with $\textsc{Choose}(g, \dots)$
3. "split" g in pieces with $\textsc{Divide}(g, \dots)$
4. for each piece g_i
    1. call $\textsc{Map}(g_i, s_i, d + 1)$
    2. append result

Mapping function template: details

A recursive $\textsc{Map}(g, s, d)$ function (inspired by WHGE):

1. consider derivation options for symbol s
2. if depth $d > d_{\mathsf{max}}$
   1. choose a predefined option

   else
   1. choose option with $\textsc{Choose}(g, \dots)$
3. "split" g in pieces with $\textsc{Divide}(g, \dots)$
4. for each piece g_i
   1. call $\textsc{Map}(g_i, s_i, d+1)$
   2. append result

Meets requirements:

- valid output
- in a finite number of steps

Mapping function template: details

A recursive $\textsc{Map}(g, s, d)$ function (inspired by WHGE):

1. consider derivation options for symbol s
2. if depth $d > d_{\text{max}}$
    1. choose a predefined option

    else

    1. choose option with $\textsc{Choose}(g, \dots)$
3. "split" g in pieces with $\textsc{Divide}(g, \dots)$
4. for each piece g_i
    1. call $\textsc{Map}(g_i, s_i, d + 1)$
    2. append result

Meets requirements:

- valid output
- in a finite number of steps

CHOOSE() and DIVIDE()

A language for the 2 functions, as a CFG:

$$
\begin{aligned}
\langle\text{mapper}\rangle &::= \langle\text{n}\rangle\ \langle\text{lg}\rangle\\
\langle\text{n}\rangle &::= \langle\text{const.n}\rangle \mid \langle\text{var.n}\rangle \mid \langle\text{fun.n.g}\rangle\ (\ \langle\text{g}\rangle\ ) \mid \langle\text{fun.n.n,n}\rangle\ (\ \langle\text{n}\rangle\ ,\ \langle\text{n}\rangle\ ) \mid \langle\text{fun.n.ln}\rangle\ (\ \langle\text{ln}\rangle\ ) \mid\\
&\quad\ \langle\text{fun.n.ln,n}\rangle\ (\ \langle\text{ln}\rangle\ ,\ \langle\text{n}\rangle\ ) \mid \langle\text{fun.n.lg}\rangle\ (\ \langle\text{lg}\rangle\ )\\
\langle\text{ln}\rangle &::= \langle\text{var.ln}\rangle \mid \langle\text{fun.ln.n}\rangle\ (\ \langle\text{n}\rangle\ ) \mid \langle\text{fun.ln.n,n}\rangle\ (\ \langle\text{n}\rangle\ ,\ \langle\text{n}\rangle\ ) \mid \text{apply}\ (\ \langle\text{fun.n.g}\rangle\ ,\ \langle\text{lg}\rangle\ )\\
\langle\text{g}\rangle &::= \langle\text{var.g}\rangle \mid \langle\text{fun.g.g,n}\rangle\ (\ \langle\text{g}\rangle\ ,\ \langle\text{n}\rangle\ ) \mid \langle\text{fun.g.lg,n}\rangle\ (\ \langle\text{lg}\rangle\ ,\ \langle\text{n}\rangle\ )\\
\langle\text{lg}\rangle &::= \langle\text{fun.lg.g,n}\rangle\ (\ \langle\text{g}\rangle\ ,\ \langle\text{n}\rangle\ ) \mid \langle\text{fun.lg.g,ln}\rangle\ (\ \langle\text{g}\rangle\ ,\ \langle\text{ln}\rangle\ ) \mid \text{apply}\ (\ \langle\text{fun.g.g,n}\rangle\ ,\ \langle\text{ln}\rangle\ ,\ \langle\text{g}\rangle\ )\\
\langle\text{const.n}\rangle &::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\\
\langle\text{var.n}\rangle &::= \text{depth} \mid \text{g.count.r} \mid \text{g.count.rw}\\
\langle\text{var.g}\rangle &::= \text{g}\\
\langle\text{var.ln}\rangle &::= \text{ln}\\
\langle\text{fun.n.g}\rangle &::= \text{size} \mid \text{weight} \mid \text{weight.r} \mid \text{int}
\end{aligned}
$$

$$
\begin{aligned}
\langle\text{fun.n.n,n}\rangle &::= + \mid - \mid * \mid / \mid \%\\
\langle\text{fun.n.ln}\rangle &::= \text{length} \mid \text{max.index} \mid \text{min.index}\\
\langle\text{fun.n.ln,n}\rangle &::= \text{get}\\
\langle\text{fun.n.lg}\rangle &::= \text{length}\\
\langle\text{fun.ln.n}\rangle &::= \text{seq}\\
\langle\text{fun.ln.n,n}\rangle &::= \text{repeat}\\
\langle\text{fun.g.g,n}\rangle &::= \text{rotate.left} \mid \text{rotate.right} \mid \text{substring}\\
\langle\text{fun.g.lg,n}\rangle &::= \text{get}\\
\langle\text{fun.lg.g,n}\rangle &::= \text{split} \mid \text{repeat}\\
\langle\text{fun.lg.g,ln}\rangle &::= \text{split.w}
\end{aligned}
$$

- typed language: numbers $\langle\text{n}\rangle$, lists of numbers $\langle\text{ln}\rangle$, bit strings $\langle\text{g}\rangle$, list of bit strings $\langle\text{lg}\rangle$

Choose() and Divide()

A language for the 2 functions, as a CFG:

```
⟨mapper⟩ ::= ⟨n⟩ ⟨lg⟩
     ⟨n⟩ ::= ⟨const.n⟩ | ⟨var.n⟩ | ⟨fun.n.g⟩ ( ⟨g⟩ ) | ⟨fun.n.n,n⟩ ( ⟨n⟩ , ⟨n⟩ ) | ⟨fun.n.ln⟩ ( ⟨ln⟩ ) |
             ⟨fun.n.ln,n⟩ ( ⟨ln⟩ , ⟨n⟩ ) | ⟨fun.n.lg⟩ ( ⟨lg⟩ )
    ⟨ln⟩ ::= ⟨var.ln⟩ | ⟨fun.ln.n⟩ ( ⟨n⟩ ) | ⟨fun.ln.n,n⟩ ( ⟨n⟩ , ⟨n⟩ ) | apply ( ⟨fun.n.g⟩ , ⟨lg⟩ )
     ⟨g⟩ ::= ⟨var.g⟩ | ⟨fun.g.g,n⟩ ( ⟨g⟩ , ⟨n⟩ ) | ⟨fun.g.lg,n⟩ ( ⟨lg⟩ , ⟨n⟩ )
    ⟨lg⟩ ::= ⟨fun.lg.g,n⟩ ( ⟨g⟩ , ⟨n⟩ ) | ⟨fun.lg.g,ln⟩ ( ⟨g⟩ , ⟨ln⟩ ) | apply ( ⟨fun.g.g,n⟩ , ⟨ln⟩ , ⟨g⟩ )
⟨const.n⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  ⟨var.n⟩ ::= depth | g.count.r | g.count.rw
  ⟨var.g⟩ ::= g
 ⟨var.ln⟩ ::= ln
⟨fun.n.g⟩ ::= size | weight | weight.r | int

 ⟨fun.n.n,n⟩ ::= + | - | * | / | %
  ⟨fun.n.ln⟩ ::= length | max.index | min.index
⟨fun.n.ln,n⟩ ::= get
  ⟨fun.n.lg⟩ ::= length
  ⟨fun.ln.n⟩ ::= seq
⟨fun.ln.n,n⟩ ::= repeat
 ⟨fun.g.g,n⟩ ::= rotate.left | rotate.right | substring
⟨fun.g.lg,n⟩ ::= get
⟨fun.lg.g,n⟩ ::= split | repeat
⟨fun.lg.g,ln⟩ ::= split.w
```

- typed language: numbers $\langle \mathrm{n} \rangle$, lists of numbers $\langle \mathrm{ln} \rangle$, bit strings $\langle \mathrm{g} \rangle$, list of bit strings $\langle \mathrm{lg} \rangle$
- Map() as a pair of Choose() ($\langle \mathrm{n} \rangle$) and Divide() ($\langle \mathrm{lg} \rangle$)

CHOOSE() and DIVIDE()

A language for the 2 functions, as a CFG:

⟨mapper⟩ ::= ⟨n⟩ ⟨lg⟩
⟨n⟩ ::= ⟨const.n⟩ | ⟨var.n⟩ | ⟨fun.n.g⟩ (⟨g⟩) | ⟨fun.n.n,n⟩ (⟨n⟩ , ⟨n⟩) | ⟨fun.n.ln⟩ (⟨ln⟩) | ⟨fun.n.ln,n⟩ (⟨ln⟩ , ⟨n⟩) | ⟨fun.n.lg⟩ (⟨lg⟩)
⟨ln⟩ ::= ⟨var.ln⟩ | ⟨fun.ln.n⟩ (⟨n⟩) | ⟨fun.ln.n,n⟩ (⟨n⟩ , ⟨n⟩) | apply (⟨fun.n.g⟩ , ⟨lg⟩)
⟨g⟩ ::= ⟨var.g⟩ | ⟨fun.g.g,n⟩ (⟨g⟩ , ⟨n⟩) | ⟨fun.g.lg,n⟩ (⟨lg⟩ , ⟨n⟩)
⟨lg⟩ ::= ⟨fun.lg.g,n⟩ (⟨g⟩ , ⟨n⟩) | ⟨fun.lg.g,ln⟩ (⟨g⟩ , ⟨ln⟩) | apply (⟨fun.g.g,n⟩ , ⟨ln⟩ , ⟨g⟩)
⟨const.n⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
⟨var.n⟩ ::= depth | g.count.r | g.count.rw
⟨var.g⟩ ::= g
⟨var.ln⟩ ::= ln
⟨fun.n.g⟩ ::= size | weight | weight.r | int

⟨fun.n.n,n⟩ ::= + | - | * | / | %
⟨fun.n.ln⟩ ::= length | max.index | min.index
⟨fun.n.ln,n⟩ ::= get
⟨fun.n.lg⟩ ::= length
⟨fun.ln.n⟩ ::= seq
⟨fun.ln.n,n⟩ ::= repeat
⟨fun.g.g,n⟩ ::= rotate.left | rotate.right | substring
⟨fun.g.lg,n⟩ ::= get
⟨fun.lg.g,n⟩ ::= split | repeat
⟨fun.lg.g,ln⟩ ::= split.w

- typed language: numbers ⟨n⟩, lists of numbers ⟨ln⟩, bit strings ⟨g⟩, list of bit strings ⟨lg⟩
- MAP() as a pair of CHOOSE() (⟨n⟩) and DIVIDE() (⟨lg⟩)
- relevant (protected) functions: size, rotate.left, max.index, ...

Expressiveness?

Can express existing human-designed representations!

- standard GE[4]
  - $\textsc{Choose}() = \text{int(substring(rotate.left(g, *(gl.count.rw, 8)), 8))}$
  - $\textsc{Divide}() = \text{repeat(g, length(ln))}$
- HGE (Hierarchical GE)
  - $\textsc{Choose}() = \text{max.index(apply(weight.r, split(g, length(ln))))}$
  - $\textsc{Divide}() = \text{split(g, ln)}$
- WHGE (Weighted HGE)
  - $\textsc{Choose}() = \text{max.index(apply(weight.r, split(g, length(ln))))}$
  - $\textsc{Divide}() = \text{split.w(g, ln)}$

[4] an optimized version with null-invalidity

Bonus

The representation of grammar-based representations is grammar-based! (*meta-GE*)

We can use any grammar-based EA for evolving these representations!

Bonus

The representation of grammar-based representations is grammar-based! (*meta-GE*)

We can use any grammar-based EA for evolving these representations!

- we chose CFGGP (thought to be more efficient)
- with a diversity promotion mechanism

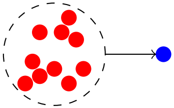
Fitness function

Problem: evolving a representation with good properties for GE

Fitness function

Problem: evolving a representation with good properties for GE

Redundancy (R)

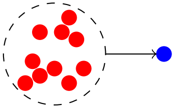


"Known" to be important: the lower, the better

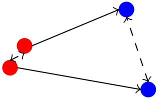
Fitness function

Problem: evolving a representation with good properties for GE

Redundancy (R)



Non-locality (NL)

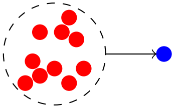


"Known" to be important: the lower, the better

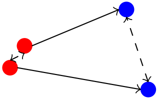
Fitness function

Problem: evolving a representation with good properties for GE

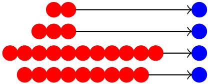
Redundancy (R)



Non-locality (NL)



Non-uniformity (NU)

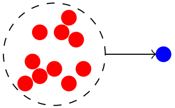


“Known” to be important: the lower, the better

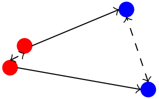
Fitness function

Problem: evolving a representation with good properties for GE

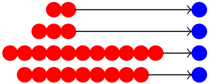
Redundancy (R)



Non-locality (NL)



Non-uniformity (NU)



"Known" to be important: the lower, the better

Three variants:

- R (single-objective)
- R/NL (multi-objective)
- R/NL/NU (multi-objective)

Table of Contents

Goals – research questions

RQ1 Can we evolve a representation which is better than the existing ones in terms of redundancy, locality, and uniformity?

RQ2 Are the evolved representations also effective when used inside an actual EA?

Procedure

1. Learning (tot. 30 runs)
   - fitness (properties R, NL, NU) computed on a symbolic regression CFG (Pagie1) with short genotypes (256 bit)
2. Validation
   - properties R, NL, NU computed on 3 CFGs (Pagie1, K-Landscape, Text) with longer genotypes (1024 bit)
   - search effectiveness on the 3 problems (tot. 2250 runs)
   - comparison against human-designed representations (GE, HGE, WHGE)

Procedure

1. Learning (tot. 30 runs)
   - fitness (properties R, NL, NU) computed on a symbolic regression CFG (Pagie1) with short genotypes (256 bit)
2. Validation
   - properties R, NL, NU computed on 3 CFGs (Pagie1, K-Landscape, Text) with longer genotypes (1024 bit)
   - search effectiveness on the 3 problems (tot. 2250 runs)
   - comparison against human-designed representations (GE, HGE, WHGE)

Answers to RQ1

Procedure

1. Learning (tot. 30 runs)
   - fitness (properties R, NL, NU) computed on a symbolic regression CFG (Pagie1) with short genotypes (256 bit)
2. Validation
   - properties R, NL, NU computed on 3 CFGs (Pagie1, K-Landscape, Text) with longer genotypes (1024 bit)
   - search effectiveness on the 3 problems (tot. 2250 runs)
   - comparison against human-designed representations (GE, HGE, WHGE)

Answers to RQ1 and RQ2

RQ1: properties

RQ1 Can we evolve a representation which is better than the existing ones in terms of redundancy, locality, and uniformity?

	Search variant	Redundancy	Non-locality	Non-uniformity
Evolved	R	0.266	0.291	0.284
	R/NL	0.247	0.28	0.292
	R/NL/NU	0.261	0.29	0.288
Human	GE	0.990	1.000	0.000
	HGE	0.620	0.403	2.211
	WHGE	0.410	0.412	2.689

RQ1: properties

RQ1 Can we evolve a representation which is better than the existing ones in terms of redundancy, locality, and uniformity?

	Search variant	Redundancy	Non-locality	Non-uniformity
Evolved	R	0.266	0.291	0.284
	R/NL	0.247	0.28	0.292
	R/NL/NU	0.261	0.29	0.288
Human	GE	0.990	1.000	0.000
	HGE	0.620	0.403	2.211
	WHGE	0.410	0.412	2.689

- yes, we can!

RQ1: properties

RQ1 Can we evolve a representation which is better than the existing ones in terms of redundancy, locality, and uniformity?

	Search variant	Redundancy	Non-locality	Non-uniformity
Evolved	R	0.266	0.291	0.284
	R/NL	0.247	0.28	0.292
	R/NL/NU	0.261	0.29	0.288
Human	GE	0.990	1.000	0.000
	HGE	0.620	0.403	2.211
	WHGE	0.410	0.412	2.689

- yes, we can!
- actual driving properties seem not to matter

RQ2: search effectiveness

RQ2 Are the evolved representations also effective when used inside an actual EA?

	Search variant	Final best fitness Best	Mean	Worst
KLand.-5	R	0.11	0.6	0.81
	R/NL	0.58	0.66	1
	R/NL/NU	0.55	0.7	1
	GE	1		
	HGE	0.58		
	WHGE	0.6		
Pagie1	R	3.42	338.66	4488.27
	R/NL	3.32	114.39	1142.28
	R/NL/NU	7.42	45.61	169.16
	GE	20.99		
	HGE	4.32		
	WHGE	2.75		
Text	R	6.5	65.12	176
	R/NL	7	88.06	176
	R/NL/NU	8.33	75.95	176
	GE	9.2		
	HGE	5.4		
	WHGE	5.4		

RQ2: search effectiveness

RQ2 Are the evolved representations also effective when used inside an actual EA?

		Final best fitness		
	Search variant	Best	Mean	Worst
KLand.-5	R	0.11	0.6	0.81
	R/NL	0.58	0.66	1
	R/NL/NU	0.55	0.7	1
	GE	1		
	HGE	0.58		
	WHGE	0.6		
Pagie1	R	3.42	338.66	4488.27
	R/NL	3.32	114.39	1142.28
	R/NL/NU	7.42	45.61	169.16
	GE	20.99		
	HGE	4.32		
	WHGE	2.75		
Text	R	6.5	65.12	176
	R/NL	7	88.06	176
	R/NL/NU	8.33	75.95	176
	GE	9.2		
	HGE	5.4		
	WHGE	5.4		

- best human-designed are better than the average evolved ones

RQ2: search effectiveness

RQ2 Are the evolved representations also effective when used inside an actual EA?

	Search variant	Final best fitness Best	Mean	Worst
KLand.-5	R	0.11	0.6	0.81
	R/NL	0.58	0.66	1
	R/NL/NU	0.55	0.7	1
	GE	1		
	HGE	0.58		
	WHGE	0.6		
Pagie1	R	3.42	338.66	4488.27
	R/NL	3.32	114.39	1142.28
	R/NL/NU	7.42	45.61	169.16
	GE	20.99		
	HGE	4.32		
	WHGE	2.75		
Text	R	6.5	65.12	176
	R/NL	7	88.06	176
	R/NL/NU	8.33	75.95	176
	GE	9.2		
	HGE	5.4		
	WHGE	5.4		

- best human-designed are better than the average evolved ones
- some evolved are better then the human-designed, on some problems

RQ2: search effectiveness

RQ2 Are the evolved representations also effective when used inside an actual EA?

	Search variant	Final best fitness Best	Mean	Worst
KLand.-5	R	0.11	0.6	0.81
	R/NL	0.58	0.66	1
	R/NL/NU	0.55	0.7	1
	GE	1		
	HGE	0.58		
	WHGE	0.6		
Pagie1	R	3.42	338.66	4488.27
	R/NL	3.32	114.39	1142.28
	R/NL/NU	7.42	45.61	169.16
	GE	20.99		
	HGE	4.32		
	WHGE	2.75		
Text	R	6.5	65.12	176
	R/NL	7	88.06	176
	R/NL/NU	8.33	75.95	176
	GE	9.2		
	HGE	5.4		
	WHGE	5.4		

- best human-designed are better than the average evolved ones
- some evolved are better then the human-designed, on some problems
- representations evolved with R only look better: redundancy is important
  - consistent with literature

Summarizing

Key findings:

- automatic design of representations of realistic complexity is feasible
- good properties can be achieved!
- $\ldots \not\Rightarrow$ good search effectiveness
  - shed some light on relevance of representation properties

Open issues:

- are R, NL, NU the right properties?
- which part of the property is “in the CFG”?

Thanks!