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.

Can we obtain automatically a good representation too?
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

1. Background and motivation
2. Evolving a representation
3. Experimental evaluation
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.”

---


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

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.”

2017: “How should the representations that are used in evolutionary algorithms, on which variation and selection act, be chosen and justified?”

Large debate, many arguments/POVs, weak agreement…

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? → inspiration by Nature/guidelines
- How to justify?

Outcome:
- Nature not inspiring enough
- No wide agreement on practical guidelines, some agreement on which properties matter
Choose/design and justify: so far

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

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”

- meta-evolution, self-adaptation, hyper-heuristic

---

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”
- meta-evolution, self-adaptation, hyper-heuristic

We did it for a challenging case: GE!

---

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, $\pi$GE, HGE/WHGE (and SGE)
Table of Contents

1. Background and motivation
2. Evolving a representation
3. Experimental evaluation
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
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

\[ g = 111001...000111 \]

\[ p = ((2^y)/2) \]
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

\[
g = 111001...000111
\]

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

\[
p = ( (2 \times y) / 2 )
\]

CFGs may be (and usually are) recursive \(\rightarrow\) infinite languages!
Evolving a representation

Mapping function template

\[ \text{CFG} \rightarrow \text{Mapping function} \]

\[ g \rightarrow \text{Mapping function} \quad p \]

\[ \langle \text{expr} \rangle \quad \langle \text{expr} \rangle \quad \langle \text{const} \rangle \quad \langle \text{op} \rangle + \langle \text{expr} \rangle \quad \langle \text{var} \rangle x(\text{Returns a derivation tree}) \]

Recursive
Actual behavior depends on Choose() and Divide()
Mapping function template

- Returns a derivation tree
Evolving a representation

Mapping function template

- Returns a derivation tree
- Recursive
Evolving a representation

Mapping function template

\[ g, s, d \]

Mapping function

\[ \text{CFG} \]

\[ \langle \text{expr} \rangle \]
\[ \langle \text{const} \rangle \]
\[ 1 \]
\[ \langle \text{op} \rangle \]
\[ + \]
\[ \langle \text{var} \rangle \]
\[ x \]

\[ \langle \text{expr} \rangle \]

\[ \langle \text{expr} \rangle \]

\[ \langle \text{var} \rangle \]

\[ \langle \text{const} \rangle \]

\[ 1 \]

\[ + \]

\[ g, s, d \]

\[ \text{Mapping function} \]

\[ \text{Choose()} \]

\[ \text{Divide()} \]

Returns a derivation tree

Recursive

Actual behavior depends on \text{Choose()} and \text{Divide()}

Medvet, Bartoli (UniTs)
A recursive $\text{Map}(g, s, d)$ 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 $\text{Choose}(g, \ldots)$
3. “split” $g$ in pieces with $\text{Divide}(g, \ldots)$
4. for each piece $g_i$
   1. call $\text{Map}(g_i, s_i, d + 1)$
   2. append result
Mapping function template: details

A recursive $\text{Map}(g, s, d)$ 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 $\text{Choose}(g, \ldots)$
3. “split” $g$ in pieces with $\text{Divide}(g, \ldots)$
4. for each piece $g_i$
   1. call $\text{Map}(g_i, s_i, d + 1)$
   2. append result

Meets requirements:

- valid output
- in a finite number of steps
A recursive $\text{MAP}(g, s, d)$ 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 $\text{CHOOSE}(g, \ldots)$
3. “split” $g$ in pieces with $\text{DIVIDE}(g, \ldots)$
4. for each piece $g_i$
   1. call $\text{MAP}(g_i, s_i, d + 1)$
   2. append result

Meets requirements:

- valid output
- in a finite number of steps
Evolving a representation

**Choose()** and **Divide()**

A language for the 2 functions, as a CFG:

\[
\text{mapper} ::= \langle n \rangle \langle lg \rangle \\
\langle n \rangle ::= \langle \text{const} \rangle \mid \langle \text{var} \rangle \mid \langle \text{fun.g} \rangle (\langle g \rangle) \mid \langle \text{fun.n.n} \rangle (\langle n \rangle, \langle n \rangle) \mid \langle \text{fun.n.ln} \rangle (\langle ln \rangle) \mid \langle \text{fun.n.lg} \rangle (\langle lg \rangle) \\
\langle ln \rangle ::= \langle \text{var} \rangle \mid \langle \text{fun.n.n} \rangle (\langle n \rangle) \mid \langle \text{fun.ln.n} \rangle (\langle n \rangle, \langle n \rangle) \mid \langle \text{fun.lg.n} \rangle (\langle lg \rangle, \langle n \rangle) \mid \langle \text{fun.lg.g.n} \rangle (\langle g \rangle, \langle n \rangle) \mid \langle \text{fun.ln.n.n} \rangle (\langle n \rangle, \langle n \rangle) \mid \langle \text{fun.lg.g.n} \rangle (\langle g \rangle, \langle ln \rangle) \mid \langle \text{fun.ln.ln} \rangle (\langle ln \rangle, \langle ln \rangle) \\
\langle \text{var.n} \rangle ::= \text{depth} \mid \text{g.count.r} \mid \text{g.count.rw} \\
\langle \text{var.g} \rangle ::= g \\
\langle \text{var.ln} \rangle ::= \text{ln} \\
\langle \text{fun.n.g} \rangle ::= \text{size} \mid \text{weight} \mid \text{weight.r} \mid \text{int} \\
\langle \text{fun.n.n} \rangle ::= + \mid - \mid * \mid / \mid % \\
\langle \text{fun.n.ln} \rangle ::= \text{length} \mid \text{max.index} \mid \text{min.index} \\
\langle \text{fun.n.lg} \rangle ::= \text{get} \\
\langle \text{fun.n.ln} \rangle ::= \text{seq} \\
\langle \text{fun.n.lg} \rangle ::= \text{length} \\
\langle \text{fun.ln.n} \rangle ::= \text{repeat} \\
\langle \text{fun.g.n} \rangle ::= \text{rotate.left} \mid \text{rotate.right} \mid \text{substring} \\
\langle \text{fun.lg.n} \rangle ::= \text{get} \\
\langle \text{fun.lg.g.n} \rangle ::= \text{split} \mid \text{repeat} \\
\langle \text{fun.lg.ln} \rangle ::= \text{split.w}
\]

- typed language: numbers \( \langle n \rangle \), lists of numbers \( \langle ln \rangle \), bit strings \( \langle g \rangle \), list of bit strings \( \langle lg \rangle \)
**Choose()** and **Divide()**

A language for the 2 functions, as a CFG:

\[
\text{mapper} ::= \langle n \rangle \langle \lg \rangle \\
\langle n \rangle ::= \langle \text{const.} n \rangle \mid \langle \text{var.} n \rangle \mid \langle \text{fun.g} n \rangle (\langle g \rangle) \mid \langle \text{fun.n.n} n \rangle (\langle n \rangle, \langle n \rangle) \mid \langle \text{fun.n.ln} n \rangle (\langle \ln \rangle, \langle n \rangle) \mid \langle \text{fun.n.lg} n \rangle (\langle \lg \rangle) \\
\langle \ln \rangle ::= \langle \text{var.} \ln \rangle \mid \langle \text{fun.ln} n \rangle (\langle n \rangle) \mid \langle \text{fun.ln.n} n \rangle (\langle n \rangle, \langle n \rangle) \mid \langle \text{fun.ln} \rangle (\langle \ln \rangle) \\
\langle g \rangle ::= \langle \text{var.g} \rangle \mid \langle \text{fun.g.g} n \rangle (\langle g \rangle, \langle n \rangle) \mid \langle \text{fun.g.ln} n \rangle (\langle \lg \rangle, \langle n \rangle) \\
\langle \lg \rangle ::= \langle \text{fun.lg.g} n \rangle (\langle g \rangle, \langle n \rangle) \mid \langle \text{fun.lg.ln} n \rangle (\langle g \rangle, \langle \ln \rangle) \mid \langle \text{fun.lg.g} \rangle (\langle g \rangle, \langle \lg \rangle) \mid \langle \text{fun.lg.ln} \rangle (\langle g \rangle, \langle \ln \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 ::= g \\
\langle \text{var.ln} \rangle ::= \ln \\
\langle \text{fun.n.g} \rangle ::= \text{size} \mid \text{weight} \mid \text{weight.r} \mid \text{int} \\
\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.lg} \rangle ::= \text{get} \\
\langle \text{fun.n.ln} n \rangle ::= \text{seq} \\
\langle \text{fun.n.lg} \rangle ::= \text{length} \\
\langle \text{fun.ln.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.ln.n} \rangle ::= \text{repeat} \\
\langle \text{fun.lg.g} n \rangle ::= \text{split} \mid \text{repeat} \\
\langle \text{fun.lg.ln} \rangle ::= \text{split.w} 
\]

- **typed language**: numbers \(\langle n \rangle\), lists of numbers \(\langle \ln \rangle\), bit strings \(\langle g \rangle\), list of bit strings \(\langle \lg \rangle\)
- **Map()** as a pair of **Choose()** (\(\langle n \rangle\)) and **Divide()** (\(\langle \lg \rangle\))
Evolving a representation

**Choose()** and **Divide()**

A language for the 2 functions, as a CFG:

\[
\langle \text{mapper} \rangle ::= \langle n \rangle \langle \lg \rangle \\
\langle n \rangle ::= \langle \text{const.n} \rangle \mid \langle \text{var.n} \rangle \mid \langle \text{fun.n.g} \rangle (\langle g \rangle) \mid \langle \text{fun.n.n.n} \rangle (\langle n \rangle, \langle n \rangle) \mid \langle \text{fun.n.ln} \rangle (\langle \ln \rangle) \mid \langle \text{fun.n.lg} \rangle (\langle \lg \rangle) \\
\langle \ln \rangle ::= \langle \text{var.ln} \rangle \mid \langle \text{fun.n.n.n.n} \rangle (\langle n \rangle, \langle n \rangle) \mid \text{apply} (\langle \text{fun.n.g} \rangle, \langle \lg \rangle) \\
\langle g \rangle ::= \langle \text{var.g} \rangle \mid \langle \text{fun.g.g.g} \rangle (\langle g \rangle, \langle n \rangle) \mid \langle \text{fun.g.lg.n} \rangle (\langle \lg \rangle, \langle n \rangle) \\
\langle \lg \rangle ::= \langle \text{fun.lg.g.g} \rangle (\langle g \rangle, \langle n \rangle) \mid \langle \text{fun.lg.lg.n} \rangle (\langle g \rangle, \langle \ln \rangle) \mid \text{apply} (\langle \text{fun.g.g.g} \rangle, \langle \ln \rangle, \langle 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 g.\text{count.r} \mid g.\text{count.rw} \\
\langle \text{var.g} \rangle ::= 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} \\
\langle \text{fun.n.n.n} \rangle ::= + \mid - \mid \ast \mid / \mid \% \\
\langle \text{fun.n.ln} \rangle ::= \text{length} \mid \text{max.index} \mid \text{min.index} \\
\langle \text{fun.n.lg} \rangle ::= \text{get} \\
\langle \text{fun.g.g.g} \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.g} \rangle ::= \text{split} \mid \text{repeat} \\
\langle \text{fun.lg.g.ln} \rangle ::= \text{split.w} 
\]

- typed language: numbers \( \langle n \rangle \), lists of numbers \( \langle \ln \rangle \), bit strings \( \langle g \rangle \), list of bit strings \( \langle \lg \rangle \)

**MAP()** as a pair of **Choose()** (\( \langle n \rangle \)) and **Divide()** (\( \langle \lg \rangle \))

- relevant (protected) functions: size, rotate.left, max.index, …
Evolving a representation

Expressiveness?

Can express existing human-designed representations!

- **standard GE**
  - \texttt{CHOOSE()} = \text{int}(\text{substring}(	ext{rotate.left}(\text{g}, *(\text{gl.count.rw}, 8)), 8))
  - \texttt{DIVIDE()} = \text{repeat}(\text{g}, \text{length}(\text{ln}))

- **HGE (Hierarchical GE)**
  - \texttt{CHOOSE()} = \text{max.index}(\text{apply}(\text{weight.r}, \text{split}(\text{g}, \text{length}(\text{ln}))))
  - \texttt{DIVIDE()} = \text{split}(\text{g}, \text{ln})

- **WHGE (Weighted HGE)**
  - \texttt{CHOOSE()} = \text{max.index}(\text{apply}(\text{weight.r}, \text{split}(\text{g}, \text{length}(\text{ln}))))
  - \texttt{DIVIDE()} = \text{split.w}(\text{g}, \text{ln})
The representation of grammar-based representations is grammar-based! (*meta-GE*)

We can use any grammar-based EA for evolving these representations!
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 \textit{with good properties} for GE
Problem: evolving a representation with good properties for GE

Redundancy (R)

“Known” to be important: the lower, the better
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)
Experimental evaluation

Table of Contents

1. Background and motivation
2. Evolving a representation
3. Experimental evaluation
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?
Experimental evaluation

Procedure

1. Learning (tot. 30 runs)
   - fitness (properties R, NL, NU) computed on a symbolic regression CFG (Page1) with short genotypes (256 bit)

2. Validation
   - properties R, NL, NU computed on 3 CFGs (Page1, 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)
Experimental evaluation

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
Experimental evaluation

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?

<table>
<thead>
<tr>
<th>Search variant</th>
<th>Redundancy</th>
<th>Non-locality</th>
<th>Non-uniformity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evolved</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0.266</td>
<td>0.291</td>
<td>0.284</td>
</tr>
<tr>
<td>R/NL</td>
<td>0.247</td>
<td>0.28</td>
<td>0.292</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>0.261</td>
<td>0.29</td>
<td>0.288</td>
</tr>
<tr>
<td>Human</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GE</td>
<td>0.990</td>
<td>1.000</td>
<td>0.000</td>
</tr>
<tr>
<td>HGE</td>
<td>0.620</td>
<td>0.403</td>
<td>2.211</td>
</tr>
<tr>
<td>WHGE</td>
<td>0.410</td>
<td>0.412</td>
<td>2.689</td>
</tr>
</tbody>
</table>
**RQ1: properties**

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

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

- 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?

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

- **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?

<table>
<thead>
<tr>
<th>Search variant</th>
<th>Final best fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
</tr>
<tr>
<td>KL_5</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0.11</td>
</tr>
<tr>
<td>R/NL</td>
<td>0.58</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>0.55</td>
</tr>
<tr>
<td>GE</td>
<td>1</td>
</tr>
<tr>
<td>HGE</td>
<td>0.58</td>
</tr>
<tr>
<td>WHGE</td>
<td>0.6</td>
</tr>
<tr>
<td>Page_1</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>3.42</td>
</tr>
<tr>
<td>R/NL</td>
<td>3.32</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>7.42</td>
</tr>
<tr>
<td>GE</td>
<td>20.99</td>
</tr>
<tr>
<td>HGE</td>
<td>4.32</td>
</tr>
<tr>
<td>WHGE</td>
<td>2.75</td>
</tr>
<tr>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>6.5</td>
</tr>
<tr>
<td>R/NL</td>
<td>7</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>8.33</td>
</tr>
<tr>
<td>GE</td>
<td>9.2</td>
</tr>
<tr>
<td>HGE</td>
<td>5.4</td>
</tr>
<tr>
<td>WHGE</td>
<td>5.4</td>
</tr>
</tbody>
</table>
RQ2: search effectiveness

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

<table>
<thead>
<tr>
<th>Search variant</th>
<th>Final best fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
</tr>
<tr>
<td>R</td>
<td>0.11</td>
</tr>
<tr>
<td>R/NL</td>
<td>0.58</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>0.55</td>
</tr>
<tr>
<td>GE</td>
<td>1</td>
</tr>
<tr>
<td>HGE</td>
<td>0.58</td>
</tr>
<tr>
<td>WHGE</td>
<td>0.6</td>
</tr>
</tbody>
</table>

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

RQ2: search effectiveness

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

<table>
<thead>
<tr>
<th>Search variant</th>
<th>Final best fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
</tr>
<tr>
<td>KLand-5</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0.11</td>
</tr>
<tr>
<td>R/NL</td>
<td>0.58</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>0.55</td>
</tr>
<tr>
<td>GE</td>
<td>1</td>
</tr>
<tr>
<td>HGE</td>
<td>0.58</td>
</tr>
<tr>
<td>WHGE</td>
<td>0.6</td>
</tr>
<tr>
<td>Page1</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>3.42</td>
</tr>
<tr>
<td>R/NL</td>
<td>3.32</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>7.42</td>
</tr>
<tr>
<td>GE</td>
<td>20.99</td>
</tr>
<tr>
<td>HGE</td>
<td>4.32</td>
</tr>
<tr>
<td>WHGE</td>
<td>2.75</td>
</tr>
<tr>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>6.5</td>
</tr>
<tr>
<td>R/NL</td>
<td>7</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>8.33</td>
</tr>
<tr>
<td>GE</td>
<td>9.2</td>
</tr>
<tr>
<td>HGE</td>
<td>5.4</td>
</tr>
<tr>
<td>WHGE</td>
<td>5.4</td>
</tr>
</tbody>
</table>

- best human-designed are better than the average evolved ones
- some evolved are better than the human-designed, on some problems
RQ2: search effectiveness

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

<table>
<thead>
<tr>
<th>Search variant</th>
<th>Final best fitness</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best</td>
<td>Mean</td>
</tr>
<tr>
<td>KLand.-5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>0.11</td>
<td>0.6</td>
</tr>
<tr>
<td>R/NL</td>
<td>0.58</td>
<td>0.66</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>0.55</td>
<td>0.7</td>
</tr>
<tr>
<td>GE</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>HGE</td>
<td>0.58</td>
<td></td>
</tr>
<tr>
<td>WHGE</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>Page1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>3.42</td>
<td>338.66</td>
</tr>
<tr>
<td>R/NL</td>
<td>3.32</td>
<td>114.39</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>7.42</td>
<td>45.61</td>
</tr>
<tr>
<td>GE</td>
<td>20.99</td>
<td></td>
</tr>
<tr>
<td>HGE</td>
<td>4.32</td>
<td></td>
</tr>
<tr>
<td>WHGE</td>
<td>2.75</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>6.5</td>
<td>65.12</td>
</tr>
<tr>
<td>R/NL</td>
<td>7</td>
<td>88.06</td>
</tr>
<tr>
<td>R/NL/NU</td>
<td>8.33</td>
<td>75.95</td>
</tr>
<tr>
<td>GE</td>
<td>9.2</td>
<td></td>
</tr>
<tr>
<td>HGE</td>
<td>5.4</td>
<td></td>
</tr>
<tr>
<td>WHGE</td>
<td>5.4</td>
<td></td>
</tr>
</tbody>
</table>

- best human-designed are better than the average evolved ones
- some evolved are better than 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!
- ... 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!