Evolving Levels and Game Agents Using Grammatical Evolution

Evolutionary Computing 2014

Bert Massop, Simon Prins, Tom Tervoort

January 23, 2014

Universiteit Utrecht

Grammatical Evolution

- Explained in paper *Grammatical Evolution* by Michael O'Neill et al., 2001 [2]
- Seeks to evolve sentences in a given context-free grammar.
- Heavily inspired by biology.
- Implemented in the GEVA framework.

Encoding and Evolution

- Solutions, or “chromosomes”, encoded as a finite sequence of (32-bit) natural numbers called “codons”.
- Mehcanism:
  - Move to the next codon for each encountered nonterminal.
  - Pick a production to expand it to based on the codon value.
  - $\textit{production} = \textit{codon value} \pmod{\textit{production count}}$
  - Wrap when reaching the end of the sequence.
  - Stop when there are no more nonterminals, or a certain depth is reached.

- Different genetic operators can be used.
- Here: one-point crossover and mutation by randomizing one codon.

Example

$$
\begin{array}{lcll}
\langle exp \rangle & ::= & \langle exp \rangle \ \langle operator \rangle \ \langle exp \rangle & (0) \\
 & | & \texttt{1.0} & (1) \\
 & | & \texttt{X} & (2) \\
\langle operator \rangle & ::= & \texttt{+} & (0) \\
 & | & \texttt{-} & (1) \\
 & | & \texttt{*} & (2) \\
 & | & \texttt{/} & (3)
\end{array}
$$

- Chromosome: **12 7 33 51 2 44 22 19**
- Start with nonterminal <exp>.
- $12 = 0 \pmod 3$, so pick first production.

Example

$$
\begin{array}{lcll}
\langle \mathit{exp} \rangle & ::= & \langle \mathit{exp} \rangle \; \langle \mathit{operator} \rangle \; \langle \mathit{exp} \rangle & (0) \\
& | & \texttt{1.0} & (1) \\
& | & \texttt{X} & (2) \\
\\
\langle \mathit{operator} \rangle & ::= & \texttt{+} & (0) \\
& | & \texttt{-} & (1) \\
& | & \texttt{*} & (2) \\
& | & \texttt{/} & (3)
\end{array}
$$

- Chromosome: **~~12~~ 7 33 51 2 44 22 19**
- Sentence: $\langle \mathit{exp} \rangle \; \langle \mathit{operator} \rangle \; \langle \mathit{exp} \rangle$
- First expand leftmost subexpression: $7 = 1 \; (\mathbf{mod}\; 3)$.

Example

$$
\begin{array}{llll}
\langle exp \rangle & ::= & \langle exp \rangle\ \langle operator \rangle\ \langle exp \rangle & (0) \\
 & | & \texttt{1.0} & (1) \\
 & | & \texttt{X} & (2) \\
\langle operator \rangle & ::= & \texttt{+} & (0) \\
 & | & \texttt{-} & (1) \\
 & | & \texttt{*} & (2) \\
 & | & \texttt{/} & (3)
\end{array}
$$

- Chromosome: **~~12~~ ~~7~~ 33 51 2 44 22 19**
- Sentence: $\texttt{1.0}\ \langle operator \rangle\ \langle exp \rangle$
- Operator is next: $33 = 1 \pmod 4$.

Example

$$
\begin{array}{llll}
\langle exp \rangle & ::= & \langle exp \rangle \ \langle operator \rangle \ \langle exp \rangle & (0) \\
& | & \texttt{1.0} & (1) \\
& | & \texttt{X} & (2) \\
\\
\langle operator \rangle & ::= & \texttt{+} & (0) \\
& | & \texttt{-} & (1) \\
& | & \texttt{*} & (2) \\
& | & \texttt{/} & (3)
\end{array}
$$

- Chromosome: **~~12~~ ~~7~~ ~~33~~ 51 2 44 22 19**
- Sentence: $\texttt{1.0 - } \langle exp \rangle$
- Remaining subexpression: $51 = 0 \pmod 3$.

Universiteit Utrecht

Example

$$
\begin{array}{llll}
\langle exp \rangle & ::= & \langle exp \rangle \ \langle operator \rangle \ \langle exp \rangle & (0) \\
& | & \texttt{1.0} & (1) \\
& | & \texttt{X} & (2) \\
\\
\langle operator \rangle & ::= & \texttt{+} & (0) \\
& | & \texttt{-} & (1) \\
& | & \texttt{*} & (2) \\
& | & \texttt{/} & (3)
\end{array}
$$

- Chromosome: **~~12~~ ~~7~~ ~~33~~ ~~51~~ 2 44 22 19**
- Sentence: $\texttt{1.0 - } (\langle exp \rangle \ \langle operator \rangle \ \langle exp \rangle)$
- Continue...

Universiteit Utrecht

Example

$$
\begin{array}{llll}
\langle exp \rangle & ::= & \langle exp \rangle\ \langle operator \rangle\ \langle exp \rangle & (0) \\
& \mid & \texttt{1.0} & (1) \\
& \mid & \texttt{X} & (2) \\
\\
\langle operator \rangle & ::= & \texttt{+} & (0) \\
& \mid & \texttt{-} & (1) \\
& \mid & \texttt{*} & (2) \\
& \mid & \texttt{/} & (3)
\end{array}
$$

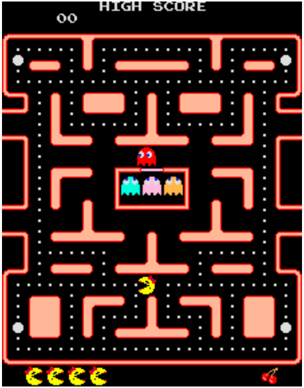
- Chromosome: **~~12~~ ~~7~~ ~~33~~ ~~51~~ ~~2~~ ~~44~~ ~~22~~ 19**
- Sentence: `1.0 - (X + 1.0)`
- No more nonterminals in sentence, so we are done.

Alternative approach

- Grammars are trees.
- Genetic Programming
  - Method for evolving tree structures.
  - Solutions represented as trees in memory.
  - Crossover: swap branches.
  - Mutation: change node's contents, while ensuring the tree remains valid.

Evolving a Ms. Pac-Man Controller Using Grammatical Evolution

E. Galván, J. Swafford, M. O'Neill, and A. Brabazon



Ms. Pac-Man

- Popular arcade game from 1982.
- Sequel to *Pac-Man*.
  - Generally considered superior due to more varied levels and smarter ghosts.
- Paper considers a Java-based derivate, not the original game.
- Premise: eat pills, avoid ghosts.
  - Eating the big power pills will temporarily enable you to eat the ghosts.

Goal

- Use GE to evolve rules for an agent playing Ms. Pac-Man.
- Try to maximize the score.
    - Stay alive while eating as many pills (and ghosts) as possible.
    - Here: one life, one level.

Universiteit Utrecht

Grammar and fitness function

```
<prog>    ::= <ifs> (0) | <ifs> <elses> (1)

<ifs>     ::= if( <vars> <equals> <vars> ){ <prog> }      (0)
            | if( <vars> <equals> <vars> ){ <action> }    (1)

<elses>   ::= else{ <prog> } (0) | else{ <action> } (1)

<action>  ::= goto(nearestPill)                           (0)
            | goto(nearestPowerPill)                      (1)
            | goto(nearestEdibleGhost)                    (2)

<equals>  ::= < (0) | <= (1) | > (2) | >= (3) | == (4)

<vars>    ::= thresholdDistanceGhost                      (0)
            | inedibleGhostDistance                       (1)
            | avgDistanceBetGhosts                        (2)
            | windowSize                                  (3)
```

- Fitness function: simulate the game.
  - Game score on death becomes the fitness score.

Experiment setup

- Do 100 runs.
- Afterwards, compare the best evolved agent against:
  - Random agent.
  - Random non-backtracking agent.
  - Simple greedy pill eater.
  - Hand-coded agent.
- Play against three different ghost AI's.

Results

Ghost Team	*Minimum Score*	*Maximum Score*	*Standard Deviation*	*Sum of all Runs*
	Random Agent			
Random Team	70	810	160.95	24,450
Legacy Team	40	200	31.75	8,670
Pincer Team	40	410	4.33	10,460
	Random Non-Reverse Agent			
Random Team	80	2,800	59.92	89,760
Legacy Team	80	5,310	74.40	69,950
Pincer Team	80	3,810	74.19	73,510
	Simple Pill Eater Agent			
Random Team	240	4,180	108.70	146,010
Legacy Team	250	5,380	107.04	154,720
Pincer Team	240	4,780	96.33	174,370
	Hand-coded Agent			
Random Team	180	11,220	242.68	579,590
Legacy Team	190	11,740	236.58	404,640
Pincer Team	790	12,820	327.10	409,040
	Evolved Agent			
Random Team	480	**11,640**	274.94	428,860
Legacy Team	470	**12,350**	311.60	394,560
Pincer Team	470	**13,830**	405.07	636,180

Results of the different experiments. [1]

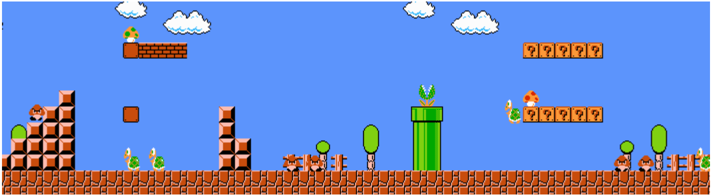
Universiteit Utrecht

Evolving Levels for Super Mario Bros Using Grammatical Evolution

N. Shaker, M. Nicolau, G. Yannakakis, J. Togelius, and M. O'Neill

Universiteit Utrecht

Goal



- Evolve levels for a side-scrolling platform game through Grammatical Evolution.
- Secondary goal: to provide a framework for analyzing and comparing expressivity of different level generators.

Universiteit Utrecht

Grammar and fitness function

- A level is a collection of *chunks* (functional attributes) at specified positions.
- Simplified grammar:

$$\begin{array}{lcll} \langle chunks \rangle & ::= & \langle chunk \rangle & (0) \\ & | & \langle chunk \rangle \; \langle chunks \rangle & (1) \end{array}$$

$$\begin{array}{lcll} \langle chunk \rangle & ::= & \texttt{coin(}\langle x \rangle\texttt{,}\langle y \rangle\texttt{,}\langle w \rangle\texttt{)} & (0) \\ & | & \texttt{enemy(}\langle x \rangle\texttt{,}\langle y \rangle\texttt{,}\langle w \rangle\texttt{)} & (1) \\ & | & \texttt{hill(}\langle x \rangle\texttt{,}\langle y \rangle\texttt{,}\langle w \rangle\texttt{)} & (2) \\ & | & [\ldots] & \end{array}$$

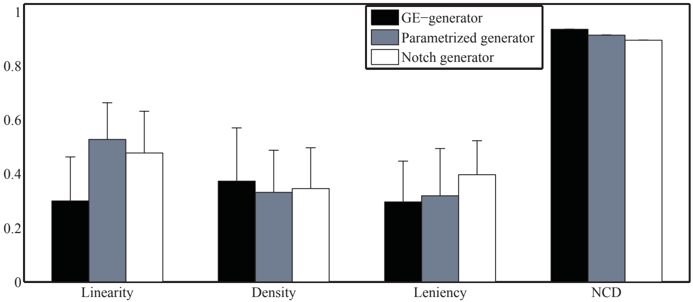
- The fitness is a weighted sum of two properties:
  - Difference between the *number of chunks* and a certain treshold.
  - The number of overlapping chunks.
- Unfortunately, the weights are not given.

Universiteit Utrecht

Experiment setup

- Evolve level for the game *Infinite Mario Bros.* by Markus 'Notch' Persson.
- Compare against Notch's level generator and against an adapted ('parametrized') version of that generator.
- Use the following expressivity measures to compare them:
  - **Linearity**, the flatness of the level
  - **Density**, the clusteredness of the chunks
  - **Leniency**, the playability of the level
  - **Compression distance**, a randomness measure

Results



Measured expressivity for the different generators. [3]

Discussion

- As mentioned, Grammatical Evolution is heavily inspired by biology.
  - *Not necessarily a good thing!*
  - In this case, most references to biological terms are confusing at best.
- Resembling nature is no guarantee for performance.
- Effectiveness is claimed based on graphs derived from experimental results.
  - Due to a lack of data, these experiments can neither be verified nor repeated.
- Grammatical Evolution as used here has major flaws.

Universiteit Utrecht

A case against Grammatical Evolution

- Simple grammar about kittens[1] and nuclear missiles
- Genetic operators as in papers
- Goal: *make the kitten happy*

[1] No kittens were harmed in the production of this presentation.

A simple grammar

$$
\begin{aligned}
\langle \textit{prog} \rangle \quad & ::= \langle \textit{action} \rangle\ \langle \textit{object} \rangle & (0) \\
& \mid\ \texttt{if (}\langle \textit{object} \rangle\ \texttt{is}\ \langle \textit{property} \rangle\texttt{)} & \\
& \quad \texttt{\{}\ \langle \textit{prog} \rangle\ \texttt{\}} & (1) \\
\langle \textit{object} \rangle \quad & ::= \texttt{nuclear missile} & (0) \\
& \mid\ \texttt{kitten} & (1) \\
\langle \textit{property} \rangle \quad & ::= \texttt{dirty} & (0) \\
& \mid\ \texttt{clean} & (1) \\
\langle \textit{action} \rangle \quad & ::= \texttt{clean} & (0) \\
& \mid\ \texttt{launch} & (1) \\
& \mid\ \texttt{look at} & (2)
\end{aligned}
$$

Universiteit Utrecht

Some sentences

$\langle prog \rangle$	::=	$\langle action \rangle$ $\langle object \rangle$	(0)
	|	`if (`$\langle object \rangle$ `is` $\langle property \rangle$`)` `{` $\langle prog \rangle$ `}`	(1)
$\langle object \rangle$	::=	`nuclear missile`	(0)
	|	`kitten`	(1)
$\langle property \rangle$	::=	`dirty`	(0)
	|	`clean`	(1)
$\langle action \rangle$	::=	`clean`	(0)
	|	`launch`	(1)
	|	`look at`	(2)

1 7 4 4 6 3

```
if (kitten is dirty)
{
    clean kitten
}
```

kitten happiness $= 9$

8 5 3

```
look at kitten
```

kitten happiness $= 7$

Universiteit Utrecht

Genetic operator performance

Let's try 1-point crossover and int-flip mutation.

1 7 4 4 6 3	**8 5 3**

`if (kitten is dirty)`
`{`
`    clean kitten`
`}`
	`look at kitten`

kitten happiness = *9*	*kitten happiness* = *7*

Universiteit Utrecht

Genetic operator performance

Let's try 1-point crossover and int-flip mutation.

1 7 4 4 6 3	**8 5 3**
`if (kitten is dirty)`	`look at kitten`
`{`	
`    clean kitten`	
`}`	
kitten happiness = *9*	*kitten happiness* = *7*

1 **7 4 4 6 3**
8 5 3

Genetic operator performance

Let's try 1-point crossover and int-flip mutation.

1 7 4 4 6 3	**8 5 3**
`if (kitten is dirty)`	`look at kitten`
`{`	
`    clean kitten`	
`}`	
kitten happiness $= 9$	*kitten happiness* $= 7$

1 **7 4 4 6 3**
8 5 3

~~1~~ **7 4 4 6 3**
0

Universiteit Utrecht

Genetic operator performance

Let's try 1-point crossover and int-flip mutation.

1 7 4 4 6 3	**8 5 3**

`if (kitten is dirty)`
`{`
`    clean kitten`
`}`
	`look at kitten`

kitten happiness = *9*	*kitten happiness* = *7*

1 **7 4 4 6 3**
8 5 3
`launch nuclear missile`
kitten happiness = *0*

~~1~~ **7 4 4 6 3**
0
`launch nuclear missile`
kitten happiness = *0*

On the origin of mistakes

- Problems can be traced back to O'Neill et al., explaining Grammatical Evolution in 2001.
- They clearly misunderstood genetic algorithms:
  "As the population being evolved comprises simple binary strings, we do not have to employ any special crossover or mutation operators [...]" [2]

Luckily. . .

- These problems are operator-dependent.
- Newer GEVA versions offers more structure-sensitive crossover and mutation operators.
  - Similar to Genetic Programming.
  - *Too bad the authors did not use those!*

Not all papers are created equal...

- Several apparent problems with the papers.
- Titles are not entirely accurate.
- Experiments cannot be verified
  - Missing parameters
  - Lacking source code
  - Lacking data / results
- Non-justified parameter choices

Pac-Man trouble

- Lacking a solid performance reference point
  - Custom Pac-Man implementation
  - Only one level with one life
  - Weird ghost behaviour
  - *Not even close to Ms. Pac-Man*
- Lacking information
  - Unclear what exactly certain variables represent
  - The experiment is non-repeatable
- Evolved strategies seem rather trivial

Mario trouble

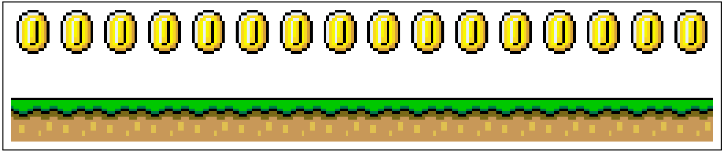
- Pointless fitness function.
  - Why not use the expressivity measures?
- Lack of justification for used measures.
- There is nothing wrong with referring to your own work, *but don't overdo it*.
  - 15/30 papers referenced have at least one author in common with this work.

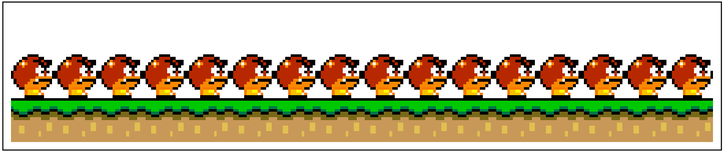
Mario trouble

- Pointless fitness function.
  - Why not use the expressivity measures?
- Lack of justification for used measures.
- There is nothing wrong with referring to your own work, *but don't overdo it*.
  - 15/30 papers referenced have at least one author in common with this work.
- There is a difference between Super Mario Bros. and Notch's Infinite Mario Bros.

Universiteit Utrecht

Two optimal levels





"[...] the main objective of the fitness function is to create levels with an acceptable number of chunks." [1]

‘Wij van WC-eend. . . ’

*“Procedural Content Generation (PCG) is a field that has recently emerged and **proven its potential** for automatically generating different aspects of game content such as game rulesets [4], [5], maps [6], [7], levels [8], [9], [10], racing tracks [11], [12] or even whole games [13], [14]. PCG can be used both offline, in order to make the game development process more efficient, and online, to allow the generation of endless variations of a game, make it infinitely replayable and adapting its content to the player [15], [16]. An overview of the **state of the art** can be found in [17], [18].”* [3]

Universiteit Utrecht

‘Wij van WC-eend...’

“Procedural Content Generation (PCG) is a field that has recently emerged and ***proven its potential*** *for automatically generating different aspects of game content such as game rulesets [4], [5], maps [6], [7], levels [8], [9], [10], racing tracks [11], [12] or even whole games [13], [14]. PCG can be used both offline, in order to make the game development process more efficient, and online, to allow the generation of endless variations of a game, make it infinitely replayable and adapting its content to the player [15], [16]. An overview of the* ***state of the art*** *can be found in [17], [18].”* [3]

Conclusions

- Effectiveness of Grammatical Evolution is questionable.
- Discussed papers fail to provide convincing evidence.
- Genetic Programming appears to be more promising.

Conclusions

- Effectiveness of Grammatical Evolution is questionable.
- Discussed papers fail to provide convincing evidence.
- Genetic Programming appears to be more promising.

- Don't use Grammatical Evolution.
  - Or at least not as described in the O'Neill papers.

Universiteit Utrecht

References

Edgar Galván-López, John Mark Swafford, Michael O'Neill, and Anthony Brabazon.

Evolving a Ms. PacMan controller using Grammatical Evolution.

In Applications of Evolutionary Computation, pages 161–170. Springer, 2010.

Michael O'Neill and Conor Ryan.

Grammatical Evolution.

IEEE Transactions on Evolutionary Computation, 5(4):349–358, 2001.

Noor Shaker, Miguel Nicolau, Georgios N Yannakakis, Julian Togelius, and Michael O'Neill.

Evolving levels for Super Mario Bros using Grammatical Evolution.

In IEEE Conference on Computational Intelligence and Games, pages 304–311. IEEE, 2012.