Basics of Modern Computer Science

Evolutionary Algorithms
Evolution of Symbolic Structures

Ivan Zelinka

MBCS CIPT, www.bcs.org/
http://www.springer.com/series/10624

Department of Computer Science
Faculty of Electrical Engineering and Computer Science, VŠB-TUO
17. listopadu 15, 708 33 Ostrava-Poruba
Czech Republic
www.ivanzelinka.eu
Topics

• Special class of evolutionary algorithms used to synthesize symbolic objects and structures
• Overview of used techniques in symbolic regression
• Evolvable hardware
• Genetic programming
• Grammatical evolution
• Analytic programming
• Frontiers
The objectives of the lesson are:

• Show that there is special class of evolutionary algorithms that can handle symbolic objects.
• Demonstrate that complex structures can be synthesized from symbolic objects.
• Give slight introduction about so called evolutionary hardware.
• Discuss in details the main evolutionary techniques for symbolic regression from oldest to newest.
Evolution of Symbolic Structures
Brief Overview

- Symbolic regression is a process of more complex structure building by small simple building blocks.
- It is synthesis of complex structures according to user defined conditions.
  - For example, measured data approximation by suitable synthesized function.
  - Synthesis of control program for robot optimal trajectory.
  - Suitable design of logic and electronic circuits.
  - Many others for which this principle is a suitable design.
Evolution of Symbolic Structures

Brief Overview

Evolutionary manipulation with simple predefined objects essential for the synthesis of more complex structures which satisfy the predetermined conditions. As an example electronic circuit can be used.
Evolution of Symbolic Structures
Brief Overview

Examples:
- Robot control program
- Antenna
- Controller for feedback control
- ...

http://www.nelsonrobotics.org/evolutionary_robotics_web/links.html
Evolution of Symbolic Structures
Brief Overview

Introduction to Symbolic Regression by Means of Evolutionary Algorithms

Motivation
- Contemporary state
  - Hierarchy

Genetic programming
- Definition and ideas
- GP principles
- Data representation
- Complexity effort
- Examples

Grammar evolution
- Definition and ideas
- GE principles
- Data representation
- Examples
- Cos(x) fitting

Analytic programming
- Definition and ideas
- AP principles
- Data representation
- Complexity effort
- Algorithms used
- Examples

Conclusion
- Demo of AP

Comparative study
- Two box problem
  - Quintic
  - Sextic
  - Sin three
  - Sin four
- Multiagent problem

- Random selection
  - Quintic
  - Sextic
  - Sin three
  - Sin four
  - ODE solving
Evolution of Symbolic Structures

History

The field of evolutionary computation (according to Koza)
- Evolutionary programming
- Evolution strategies
- Classifier systems
- Evolvable hardware
- GA
- GP
- Grammar evolution
- Analytic programming
Evolution of Symbolic Structures

Problem Decomposition

Decompose

Original problem

Subproblem 1

Solution to subproblem 1

Subproblem 2

Solution to subproblem 2

Subproblem 3

Solution to subproblem 3

Solve subproblems

Solution to original problem

Solve original problem

Identify regularities

First recording rule

Original representation of the problem

Second recording rule

New representation of the problem

Third recording rule

Change representation

Solve

Solution to the problem
Evolution of Symbolic Structures

Problem Decomposition

Decompose

Solve subproblems

Solve original problem

Original problem 1
\[ \frac{d}{dx} x^2 + \sin x \]

Subproblem 1.1: Differentiate \( x^2 \)
\[ \frac{d}{dx} x^2 = 2x \]

Solution to subproblem 1.1:
\[ \frac{d}{dx} x^2 = 2x \]

Solution to original problem 1:
\[ x^2 \cos x + 2x \sin x \]

Subproblem 1.2: Differentiate \( \sin x \)
\[ \frac{d}{dx} \sin x = \cos x \]

Solution to subproblem 1.2:
\[ \frac{d}{dx} \sin x = \cos x \]

Decompose problem 2

Solve problem 2's subproblems

Solve original problem 2

Original problem 2
\[ \frac{d}{dx} x^2 + \sin x + x^2 \]

Subproblem 2.1
\[ \frac{d}{dx} x^2 + \sin x \]

Decompose subproblem 2.1

Solve subproblem 2.1's subproblems

Solution to subproblem 2.1:
\[ x^2 \cos x + 2x \sin x \]

Solution to original problem 2:
\[ x^2 \cos x + 2x \sin x + 2x \]

Subproblem 2.2
\[ \frac{d}{dx} x^2 \]

Reuse solution to sub-subproblem 2.1.1
\[ \frac{d}{dx} x^2 = 2x \]
Evolution of Symbolic Structures
Problem Decomposition

Decompose

Solve subproblems

Solve original problem

Original problem 3
\[
\frac{d}{dx} (x^3 + x^4)
\]

Subproblem 3.1:
Differentiate \( x^3 \)
\[
\frac{d}{dx} x^3 = 3x^2
\]

Solution to subproblem 3.1:
\[
\frac{d}{dx} x^3 = 3x^2
\]

Solution to original problem 3
\[
3x^2 + 4x^3
\]

Subproblem 3.2:
Differentiate \( x^4 \)
\[
\frac{d}{dx} x^4 = 4x^3
\]

Solution to subproblem 3.2:
\[
\frac{d}{dx} x^4 = 4x^3
\]

Solution to original problem 3
\[
3x^2 + 4x^3
\]

Decompose

Solve subproblem

Solve original problem

Original problem 3
\[
\frac{d}{dx} (x^3 + x^4)
\]

Subproblem 3.3:
Differentiate \( x^m \)
\[
\frac{d}{dx} x^m = mx^{m-1}
\]

Solution to subproblem 3.3:
\[
\frac{d}{dx} x^m = mx^{m-1}
\]

Solution to original problem
\[
\begin{align*}
m &= 3 \\
\frac{d}{dx} x^3 &= 3x^2 \\
\frac{d}{dx} x^4 &= 4x^3 \\
\end{align*}
\]

Solution to original problem 3
\[
3x^2 + 4x^3
\]
Evolution of Symbolic Structures
Selected Methods


Evolution of Symbolic Structures
Hierarchy

AP
Till now Mathematica, but possible is also
LISP, Fortran, C++, Perl,…
DSH – discrete set handling

GE
LISP, Fortran, C++, Perl,…
Backus-Naur Form

GP
LISP, trees
Evolution of Symbolic Structures
An Overview

• When we speak of symbolic regression in the context of evolutionary algorithms, there are currently few tools. The most famous is certainly genetic programming (Koza, 1998) (Koza, 1999) and grammatical evolution (O'Neill, 2003).

• Another approach based on either evolutionary or hybrid techniques using symbolic regression were introduced e.g. Johnson (Johnson 2003) working with artificial immune system.

• Salustowicz introduced a system of program generation of adaptive probability distribution of all possible solutions in the probabilistic incremental program evolution (Probabilistic Incremental Program Evolution - PIPE) (Salustowicz, 1997)
Evolution of Symbolic Structures
An Overview

- **Grammatical evolution** preceded **GADS** (Genetic Algorithm for deriving Software), improved genetic programming approach that solves the grammar of the programming (Paterson, 2003), (Paterson, 1996).

- Approaches that differ in the representation and grammar are described in the **programming of gene expression** (Gene Expression Programming) (Ferreira, 2006), **multiexpression programming** (Multiexpression Programming) (Oltean, 2003), **meta-modeling** for symbolic regression and **Pareto simulated annealing** (Stinstra 2006).

- Group of hybrid techniques includes **numerical methods associated with evolutionary systems**, e.g. (Davidson, 2003).

- Finally, the method for symbolic regression also includes the **analytic programming** (Oplatková, Zelinka, 2006) (Zelinka, 2002) (Zelinka, 2005).
Evolution of Symbolic Structures
Genetic Programming

- **Genetic Programming** (GP) was introduced in the late 80's of the last century by John Koza (Koza, 1998) (Koza, 1999).
- He proposed modification of genetic algorithms (Holland, 1992) for the so called programs creation (user programs, functions,...) within the framework of symbolic regression and named it the **genetic programming**.
- In this modification new population was not bred by classical numerical approach, but **on a symbolic level**.
- This means that the population did not contain numbers, **but symbolic objects** themselves (mathematical function, the user subroutines,...).
- Another possibility is Read's code.
Evolution of Symbolic Structures
Genetic Programming

- Simple elementary objects can be used to create more complex form, which describes the behavior of a given problem.
- For example, the approximation of functions by basic objects could be mathematical operators, $+, -, cos, tan$, variables $x$, constants $c_1, c_2, \ldots$, if necessary other numbers. For these objects can then rise structure such as

$$\frac{2x - \sin(y)}{\pi}$$

and possibly something far more complex.
- Finding the most appropriate expression in symbolic form.
Evolution of Symbolic Structures
Genetic Programming

Set of elementary objects like +, -, *, computer commands, functions and user defined functions

Process of symbolic regression

functions formulas
control commands
Evolution of Symbolic Structures
Genetic Programming

- **Inventor:** J. R. Koza
- **Main aim:** Automatic program creation by means of genetic algorithm
- **Used language:** Lisp
- **Program representation:** symbolic in List, visualization as tree
- **Homepage:** [www.genetic-programming.org](http://www.genetic-programming.org)
Evolution of Symbolic Structures
Genetic Programming

- Arithmetic operations: PLUS, MINUS, MULT, DIV, ... 
- Relations: LESS, EQUAL, GREATER, ... 
- Mathematical functions: SIN, COS, EXP, LOG, ... 
- Boolean operations: AND, OR, NOT, ... 
- Instruction sequences: PROGN 
- Conditionals: IF-THEN-ELSE, COND, ... 
- Loops: DO-UNTIL, WHILE-DO, FOR-DO, ... 
- Problem-specific functions: MOVE-RANDOM, IF-FOOD-HERE, PICK-UP, ... 

Terminals → Functions → Fitness measure → Parameters → Termination criterion → Architecture → GP → A computer program
Program can also contain complex substructures like:
- Subroutines
- Loops
- Recursions
- Storage
- ...
Evolution of Symbolic Structures
Genetic Programming - Definition

- Genetic programming is different from all other approaches to artificial intelligence, machine learning, neural networks, adaptive systems, reinforcement learning, or automated logic in all (or most) of the following seven ways:
  - Representation: Genetic programming overtly conducts its search for a solution to the given problem in program space.
  - Role of point-to-point transformations in the search: genetic programming does not conduct its search by transforming a single point in the search space into another single point, but instead it transforms a set (population) of points into another set of points.
Evolution of Symbolic Structures
Genetic Programming - Definition

- Role of hill climbing in the search: genetic programming does not rely exclusively on greedy hill climbing to conduct its search, but instead it allocates a certain number of trials, in a principled way; to choices that are known to be inferior.
- Role of determinism in the search: genetic programming conducts its search probabilistically.
- Role of an explicit knowledge base: none.
- Role of the inference methods of formal logic in the search: none.
- Underpinnings of the technique: biologically inspired.
Evolution of Symbolic Structures
Genetic Programming – Tree Representation

\[
((A \lor C) \land ((A \lor A \land (A \lor B)) \lor (C \lor B))) \land (((C \land ((A \land \neg B \land C) \lor (\neg A \land B) \lor (\neg A \land \neg C) \lor (B \land \neg C))) \lor A) \land \\
((A \lor B) \land ((A \land \neg B \land C) \lor (\neg A \land B) \lor (\neg A \land \neg C) \lor (B \land \neg C))) \land \\
(((A \land \neg B \land C) \lor (\neg A \land B) \lor (\neg A \land \neg C) \lor (B \land \neg C)))
\]
Evolution of Symbolic Structures
Genetic Programming – Tree Representation

\[(A \land B \land \neg C) \lor (A \land \neg B \land C) \lor (\neg A \land B \land C) \lor (\neg A \land \neg B \land \neg C)\]
Evolution of Symbolic Structures
Genetic Programming – Tree Representation

[Diagram of a tree-like structure with nodes labeled 'IfFoodAhead', 'Move', 'Prog2', 'Prog3', and 'Left', 'Right'. The tree structure illustrates the evolution of symbolic structures through genetic programming, specifically with tree representations.]
Evolution of Symbolic Structures
Genetic Programming – Crossover

Parent 1

Parent 2

Z * 0.234 + X - 0.789

Z * Y * (Y + Z * 0.314)

Offspring 1

Offspring 2

(Y + Z * 0.314) + X - 0.789

Z * Y * (Z * 0.243)
Evolution of Symbolic Structures
Genetic Programming – Mutation

Before mutation

After mutation

Mutation
Evolution of Symbolic Structures
Genetic Programming – Read Code

![Symbolic Structure Diagram](image-url)
Evolution of Symbolic Structures
Genetic Programming – Read Code

Mutation

22010210200

2010

10 10

210200

2010

10 10

220010

0 0 0 0

2 1

220010

0 0 0 0
Evolution of Symbolic Structures
Genetic Programming – Read Code

Crossover selection

22010210200

2010

10

10

210200

22010210220010

2010

10

10

210200

220010

2

220010

1

0

0

0

0

0

0

0

0

0
Evolution of Symbolic Structures
Genetic Programming – Read Code

Crossover result

220010 210200

22010210220010

2010 210200

0 10 10

0 0 0

0 0 0

0 0 0
Evolution of Symbolic Structures
Genetic Programming

• As in the case of genetic algorithms, genetic programming, and also used parameter mutation probability and the probability of crossover. In the case of genetic programming to more states and the probability of a node mutation. The creation of new solutions can be written in words:
  – First is the maximum tree depth $D_{\text{max}}$, then there are two methods for creating new solutions:
    • Filling method (each branch has a length $d = D_{\text{max}}$)
    • Nodes at depth $d < D_{\text{max}}$ are randomly selected from the set of non-terminals (functions)
    • Nodes at depth $d = D_{\text{max}}$ are selected from a plurality of terminals $T$
Evolution of Symbolic Structures
Genetic Programming

- The growth method (each branch has a length $d \leq D_{\text{max}}$)
  - Nodes at depth $d < D_{\text{max}}$ randomly selected from a unified set of functions $F$ and $T$ terminals.
  - Nodes at depth $d = D_{\text{max}}$ randomly selected from terminals $T$.
- The first population is made up of half and half commonly two methods.
- In genetic programming during the evolution an effect called **Bloat** discovers. Typically, the average length of an individual increase linearly with the number of iterations.
- A penalty of long solutions adapted operators mutation and crossover, respectively. use multi-criteria optimization.
- There are several approaches to genetic programming
  - (Oltean, 2003): linear genetic programming.
  - (Brameier, 2001), multiexpression programming, Cartesian Genetic Programming.
  - (Miller, 2002) and the infix form of genetic programming (Oltean, 2003b).
Genetic Programming

- To run demonstration you need to download and install program [http://www.wolfram.com/cdf-player/](http://www.wolfram.com/cdf-player/).
- Follow instruction in selected demonstration program.
Evolution of Symbolic Structures
Grammatical Evolution

- **Inventor**: C. Ryan
- **Main aim**: Automatic program creation by means of an arbitrary computer language
- **Used language**: Lisp, C++, Java, XML, Perl, Fortran, ...
- **Program representation**: Backus-Naur Form
- **Homepage**: [www.gramatical-evolution.org](http://www.gramatical-evolution.org)
Grammatical Evolution (GE) can be seen as a type of genetic programming (GP) based on grammar (O'Neill, Ryan, 2003).

Genetic programming was originally designed for the language LISP.

Using grammatical evolution, programs can be created in any language if we use the Backus-Naur form (BNF).

BNF grammar consists of terminals, which are objects that can exist in a given language, such as +, -, sine, log, etc., and non-terminals.

Non-terminals can be replaced by one or more terminals and non-terminals.

A non-terminal symbol is any symbol that can be overwritten by another string of symbols.
Evolution of Symbolic Structures
Grammatical Evolution

- Conversely terminal symbol **may no longer** be overwritten.
- The main advantage compared to GE GP is given the opportunity to generate multi-line functions **in any programming language**.
- Programs GE not written directly in the tree structure as in GP, **but using a linear genome**, which may be for example a sequence of integers.
- Transfer of genotype to phenotype is performed using modulo operation, where \( n \) is determined by selecting the maximum sensible, e.g. a given number of functions.
Evolution of Symbolic Structures
Grammatical Evolution

- Grammar will be called quadruple $G = \{ N, T, P, S \}$, where
  - $N$ is a finite set of non-terminal symbol.
  - $T$ is a finite set of terminal symbols.
  - $S$ is the start symbol.
  - $P$ is a set of rewrite rules.

- Comparing with GP, GE differs in major three ways:
  - Using linear genomes.
  - Ontogenetic performs mapping from genotype to phenotype (program).
  - Uses the grammar to create legal structures in space phenotypes.
Evolution of Symbolic Structures
Grammatical Evolution

• Used Backus – Naur form grammar definition for mapping program. It is the (normal) form of notation of context-free grammars originally designed to describe the syntax rules that are written in the form:

\[
\langle \text{symbol} \rangle \ ::= \langle \text{options} \rangle
\]

where \langle \text{symbol} \rangle belong to the alphabet and \langle \text{options} \rangle are strings of terminals and non-terminals separated by '||', which separates the possibilities of generative grammar.

• Generative grammar is used to create syntactically correct program from the registered rules of succession. It can therefore write:

\[
\langle \text{symbol} \rangle \ ::= \langle \text{choice 1} \rangle \ | \langle \text{choice 2} \rangle \ | \langle \text{choice 3} \rangle \ |... \ | \langle \text{choice n} \rangle
\]
Evolution of Symbolic Structures
Grammatical Evolution

\[ \langle \text{symbol} \rangle :: = \langle \text{choice } 1 \rangle \mid \langle \text{choice } 2 \rangle \mid \langle \text{choice } 3 \rangle \mid \ldots \mid \langle \text{choice } n \rangle \]

where options can substitute terminals and non-terminals. An example is the definition of rules for the processing of positive decimal numbers:

1.  \[ \langle \text{digit} \rangle :: = 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \]
2.  \[ \langle \text{unsigned integer} \rangle :: = \langle \text{digit} \rangle \mid \langle \text{unsigned integer} \rangle \langle \text{digit} \rangle \]
3.  \[ \langle \text{decimal fraction} \rangle :: = \langle \text{digit} \rangle \]
4.  \[ \langle \text{decimal number} \rangle :: = \langle \text{unsigned integer} \rangle \mid \langle \text{decimal fraction} \rangle \mid \langle \text{unsigned integer} \rangle \langle \text{decimal fraction} \rangle \]
Rules can be then interpreted as follows:

- Rule 1 - digit is one of the terminals of the interval \([0, 9]\).
- Rule 2 - unsigned integer is one or more digits.
- Rule 3 - fractional part is a period followed by unsigned integers.
- Rule 4 - decimal number is either integer or fractional part only, or unsigned integer followed by a decimal part.

Terminals are the digits '0 '... '9 ' and a “dot” symbol.

The start symbol is <decimal number>.
Evolution of Symbolic Structures
Grammatical Evolution

- Grammatical evolution **differs** then, when we only use terminals and when a set of rules is also nonterminal.
- The **terminal** is an object that has no other option branching (constant, independent variable,...).
- **Nonterminal** contains terminals, as well as other non-terminals. There are also used names like
  - T - rule to the right side (all substitutions) terminals only.
  - N - then rule is a rule which also includes non-terminals or at least one (O'Neill, 2003).
- Backus-Naur form can be written as follows. First defined non-terminals N, terminals T and also start symbol S:
  - N = \{expr, op, pre_op, var\}.
  - T = \{sin, +, -, /, *, x, 1\}.
  - S = expr.
Evolution of Symbolic Structures
Grammatical Evolution

- Subsequently, they are then broken down by substituting various options - grammar rules by Backus-Naur form:

<table>
<thead>
<tr>
<th>Non-and-teminals</th>
<th>Unfolding</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) &lt;expr&gt;</td>
<td>: : = &lt;expr&gt; &lt;op&gt; &lt;expr&gt;</td>
<td>(0)</td>
</tr>
<tr>
<td></td>
<td>( &lt;expr&gt; &lt;op&gt; &lt;expr&gt; )</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>&lt;pre-op&gt; ( &lt;expr&gt; )</td>
<td>(2)</td>
</tr>
<tr>
<td></td>
<td>&lt;var&gt;</td>
<td>(3)</td>
</tr>
<tr>
<td>B) &lt;op&gt;</td>
<td>: : = +</td>
<td>(0)</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>(1)</td>
</tr>
<tr>
<td></td>
<td>/</td>
<td>(2)</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td>(3)</td>
</tr>
<tr>
<td>C) &lt;pre-op&gt;</td>
<td>: = Sin</td>
<td>(0)</td>
</tr>
<tr>
<td>D) &lt;var&gt;</td>
<td>: = x</td>
<td>(0)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>(1)</td>
</tr>
</tbody>
</table>
Evolution of Symbolic Structures
Grammatical Evolution

- In GE are conducted substitutions which gradually replace the head of non-terminals on the left to the right terminals. Application of the rules in relation works by replacing the rule \(<\text{expr}> \ D \ <\text{var}>: \)
  \[
  \text{\langle expr\rangle \ \op\ \langle expr\rangle} \Rightarrow \text{\langle var\rangle \ \op\ \langle expr\rangle}
  \]
- These rules can be written in the form of a tree, which is often easier to read.
Evolution of Symbolic Structures
Grammatical Evolution

- In the canonical version of the individual in the form of binary strings.
- 8-bit sequence are known as **codons**.
- Codons are converted to the integer values, according to which they then pick the corresponding rules according to

\[
\text{rule} = \text{Integer value codon} \mod \text{number of options the current nonterminal}
\]

which performs integer division - modulo (MOD).

- The following picture shows an example of an individual grammatical evolution containing integer values that were generated from 8-bit numbers (codons).

```
```
Evolution of Symbolic Structures  
Grammatical Evolution

• The process then proceeds to a transcript of non-terminals gradually replaced by the relevant rules, until such time as all non-terminals are replaced terminals.

• It starts usually \(<\text{expr}>\) nonterminal, which is defined as starting symbol \(S\). The first is the number of codon 220, after MOD operation we obtain a value of zero. This means that in the rule and use the terminal (0). Used rule we can write as \(A0\) (\textit{rule A with the terminal 0}). Thus we get

\[
<\text{expr}> \ <\text{op}> \ <\text{expr}>
\]

• Continuing the first nonterminal, which is the leftmost (<\text{expr}>). Through the second codon of the individual and re-apply operation, i.e. 40 MOD 4 We get again zero and is replaced <\text{expr}> <\text{expr}> <\text{op}> <\text{expr}>. The result is then

\[
<\text{expr}> \ <\text{op}> \ <\text{expr}> \ <\text{op}> \ <\text{expr}>
\]
Evolution of Symbolic Structures
Grammatical Evolution

• Again starting from the leftmost expression. The third codon leads again with the resulting rule $A_0$ entry

\[
<expr> \ <op> \ <expr> \ <op> \ <expr> \ <op> \ <expr>
\]

• The fourth codon 203 with the value we have already changed the expression $<expr>$ rule $A_3$, $<var>$.

\[
<var> \ <op> \ <expr> \ <op> \ <expr> \ <op> \ <expr>
\]

• The following codon determines the value $<var>$ us, here are two options - variable or constant 1. The value of the codon 101, after the function modulo 2 will rule $D_1$. So constant is chosen first.

• The expression has the form

\[
1 \ <op> \ <expr> \ <op> \ <expr> \ <op> \ <expr>
\]
Evolution of Symbolic Structures
Grammatical Evolution

- Now proceed with the operation `<op>` expression. Another codon produces rule **B1**. (53 MOD 4 is 1). The next step then expands again `<expr>` rule **A2**. We get

\[
1 - <\text{pre-op}> (<\text{expr}>) <\text{op}> <\text{expr}> <\text{op}> <\text{expr}>
\]

- Because `<pre-op>` in our case has only one option, do not use codon to the transcript to the terminal, but the value will replace immediately.

\[
1 - \sin (<\text{expr}>) <\text{op}> <\text{expr}> <\text{op}> <\text{expr}>
\]

- In the same way we continue until all the rules (non-terminals) are replaced by their terminals. The resulting function in this case is

\[
1 - \sin (x) * \sin (x) - \sin (x) * \sin (x)
\]
Evolution of Symbolic Structures
Grammatical Evolution

- If it happens that you lack complete codons for finding the resulting function, then begin to use cyclically from the beginning (O'Neill, 2003).
- Because grammatical evolution is actually a combination of genetic algorithms and appropriate grammar, following the process of breeding new offspring is performed as in genetic algorithms, thus selecting parents, crossing and mutation.
- It is necessary to accomplish scanning algorithm with variable chain of individuals.
- Objective function then usually forms as opposed to the desired behavior. Its value is then converted to fitness, which is necessary to evaluate the quality of the individual.
### Evolution of Symbolic Structures

#### Grammatical Evolution

T={+, -, *, /} and F={expr, op, var}

<table>
<thead>
<tr>
<th>Non-and-terminals</th>
<th>Unfolding</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) &lt;expr&gt;</td>
<td>: = &lt;expr&gt; &lt;op&gt; &lt;expr&gt; (0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1)</td>
</tr>
<tr>
<td>B) &lt;op&gt;</td>
<td>: = + (0')</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3')</td>
</tr>
<tr>
<td>D) &lt;var&gt;</td>
<td>: = x (0'')</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0'')</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(1'')</td>
</tr>
</tbody>
</table>
### Evolution of Symbolic Structures

**Grammatical Evolution**

<table>
<thead>
<tr>
<th>kodon 1</th>
<th>kodon 2</th>
<th>kodon 3</th>
<th>kodon 4</th>
<th>kodon 5</th>
<th>kodon 6</th>
<th>kodon 7</th>
<th>kodon 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>00101000</td>
<td>11000111</td>
<td>00001100</td>
<td>10100010</td>
<td>01111101</td>
<td>11100111</td>
<td>10010010</td>
<td>10001011</td>
</tr>
</tbody>
</table>

**Binary**

<table>
<thead>
<tr>
<th>Binary</th>
<th>Integer</th>
<th>BNF index</th>
</tr>
</thead>
<tbody>
<tr>
<td>00101000</td>
<td>40</td>
<td>(0)</td>
</tr>
<tr>
<td>11000111</td>
<td>162</td>
<td>(2’)</td>
</tr>
<tr>
<td>00001100</td>
<td>67</td>
<td>(1)</td>
</tr>
<tr>
<td>10100010</td>
<td>12</td>
<td>(0’’)</td>
</tr>
<tr>
<td>01111101</td>
<td>125</td>
<td>(1)</td>
</tr>
<tr>
<td>11100111</td>
<td>231</td>
<td>(1’’)</td>
</tr>
<tr>
<td>10010010</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>10001011</td>
<td>139</td>
<td></td>
</tr>
</tbody>
</table>

Two kodon remains unused

```
expr
  op
    expr
    expr

  *
  var
    X

  *
  var
    Y
```

\[ X * Y \]
Evolution of Symbolic Structures
Grammatical Evolution

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>16</td>
<td>120</td>
<td>38</td>
<td>51</td>
<td>230</td>
<td>79</td>
<td>17</td>
<td>84</td>
<td>63</td>
</tr>
<tr>
<td>56</td>
<td>80</td>
<td>71</td>
<td>168</td>
<td>214</td>
<td>147</td>
<td>31</td>
<td>3</td>
<td>91</td>
<td>112</td>
</tr>
</tbody>
</table>

Parent 1

\[ \begin{array}{c}
+ \\
* \\
X \\
Y \\
/ \\
X \\
Y
\end{array} \]

Parent 2

\[ \begin{array}{c}
+ \\
X \\
Y \\
- \\
Y \\
X
\end{array} \]

Offspring 1

\[ \begin{array}{c}
+ \\
* \\
X \\
X \\
Y
\end{array} \]

Offspring 2

\[ \begin{array}{c}
+ \\
X \\
Y \\
- \\
Y \\
/ \\
X \\
Y
\end{array} \]
• Demonstration program in *Mathematica* is accessible at website [http://www.ivanzelinka.eu/hp/Vyuka.html](http://www.ivanzelinka.eu/hp/Vyuka.html).

• To run demonstration you need to download and install program [http://www.wolfram.com/cdf-player/](http://www.wolfram.com/cdf-player/).

• Follow instruction in selected demonstration program.
Evolution of Symbolic Structures
Analytic Programming

- **Inventor:** I. Zelinka
- **Main aim:** Automatic program creation by means of an arbitrary computer language and arbitrary EA
- **Used language:** Mathematica, C++, C#, Java, ...
- **Program representation:** DSH – Discrete Set Handling
- **Homepage:** [www.ivanzelinka.eu](http://www.ivanzelinka.eu)
This part describes another method of automatic (evolutionary) programming, which is called "analytical programming" (Zelinka, Oplatkova, Noll, 2005), (Oplatkova, Zelinka, 2006).

This method, unlike methods already described is not bound to a single algorithm and is not connected with any grammar or tree representation as is the case with grammatical evolution and genetic programming.

This is an experimental method that can be understood as an alternative approach due to the GP and GE.

The first part will explain the theoretical principles of analytic programming (AP) and then will use examples.
**Evolution of Symbolic Structures**

**Analytic Programming**

- AP works as well as GP or GE with a set of functions, operators and terminal values:
  - Functions: $sin$, $tan$, $cos$, $And$, $Or$,
  - Operators: $+$, $-$, $*$, $/$, $dt$,...
  - Terminals: $2.73$, $t$...

- These objects in AP are sorted by the number of its arguments, and this number is a determining factor in the fact that symbolic objects will be used.

- This causes the sort that this sorted set (for the purposes of this publication, called GFS - General functional set (a set of general functional)) consists of hierarchically arranged subsets that contain the same number of function arguments (see next slide).

- These subsets bear $GFS_{all}$ (all elements GFS) $GFS_{0arg}$ (terminals, i.e. "function" with 0 arguments), $GFS_{1arg}$ (function with one argument), etc.
Evolution of Symbolic Structures
Analytic Programming – GFS Structure
Evolution of Symbolic Structures
Analytic Programming

- Set of functions and its possible arguments (terminals)
- Rule for construction of analytic solution from given individual
- Rule for critical situations treating:
  - Pathological functions (without arguments, self-looped...)
  - Functions with imaginary or real part (if not expected)
  - Infinity in functions
  - „Frozen“ functions (extremely long time to get their cost value – hrs, days,...)
- Rule for cost function evaluation

\[
\begin{align*}
GFS_{all} &= \{+, -, /, ^, d/dt, \sin, \cos, \tan, t, C1, \text{Mod}, \ldots\} \\
GFS_{3arg} &= \{\text{BetaRegularized,} \ldots\} \\
GFS_{2arg} &= \{+, -, /, ^, \log, \text{Mod, GammaRegularized} \ldots\} \\
GFS_{1arg} &= \{\sin, \cos, \tan, \text{Abs, Re, Im}, \ldots\} \\
GFS_{0arg} &= \{t, x, y, z, C1, C2, \text{Kinchin}, \ldots\}
\end{align*}
\]
Evolution of Symbolic Structures
Analytic Programming

- Fitness of an individual is given by difference between behavior of actual individual and expected behavior.
- It is usually summa of absolute values of differences.

\[ F_{\text{cost}} = |\text{DataSet} - F_{\text{AP}}(t)| \]
Evolution of Symbolic Structures
Analytic Programming

- Synthesis of the resulting program then runs in $GFS_{all}$ initially, but during synthesis it is progressively directed to objects of function with fewer arguments and depending on how much is left in the integer index of free individuals. $GFS_{all}$ is the union of all other $GFS_{arg}$:

$$GFS_{all} = GFS_{3arg} \cup GFS_{2arg} \cup GFS_{1arg} \cup GFS_{0arg}$$

$$GFS_{3arg} = \{LerchPhi, BetaRegularized, +, -, /, *, cos, tan, sin, log, x, t, \pi\}$$

$$GFS_{2arg} = \{+, -, /, *, cos, tan, sin, log, x, t, \pi\}$$

$$GFS_{1arg} = \{cos, tan, sin, log, x, t, \pi\}$$

$$GFS_{0arg} = \{x, t, \pi\}$$

- Custom activity of AP is trivial. An individual in a population consists of so-called integer pointer (pointer) which has specific value indices - points to a set of fundamental symbolic objects.
Evolution of Symbolic Structures
Analytic Programming

- The core of AP is DSH technique
- It allows map integer individuals into space of possible solutions

Individual \( \{x_1, x_2, x_3, x_4, x_5, x_6, \ldots \} \)

Discrete set \( \{-1, 44.35, 99, 231, \text{True, False, } -65.44, -0.01, \ldots \} \)

Integer index \( \{1, 2, 3, 4, 5, 6, 7, 8, \ldots \} \)

Cost function \( F_{cost} (x_1, \ldots, x_4, \ldots) = \sqrt{x_3 \sin(x_2)} x_4 + \ldots \)
Two kind of mapping procedure. In optimal case closed Structure is synthesized. No security procedures are needed.

Individual parameters \{1, 6, 7, 8, 9, 9\} are used by AP like pointers into GFS and through serie of mappings m1 - m5 final formula \(\sin(\tan(t)) + \cos(t)\) is created.

\[
\begin{align*}
\text{Individual} &= \{1, 6, 7, 8, 9, 9\} \\
\text{GFS}_{\text{all}} &= \{+, -, /, ^{\wedge}, d/dt, \sin, \cos, \tan, t, \Omega, \text{mod}, \ldots\}
\end{align*}
\]
Evolution of Symbolic Structures
Analytic Programming

Individual in population = \{1, 6, 7, 8, 9, 11\}

\[
\text{GFS}_{\text{all}} = \{+, -, /, ^, \frac{d}{dt}, \text{Sin}, \text{Cos}, \text{Tan}, t, C1, \text{Mod}, \ldots\}
\]

\[
\text{GFS}_{0\text{arg}} = \{t, x, y, z, C1, C2, \text{Kinchin}, t, C4, C5, \omega, \ldots\}
\]

Resulting function by AP = \text{Sin}(\text{Tan}(\omega)) + \text{Cos}(t)

Usually it is not a common case and thus security procedures are needed to synthesize closed structures.
Evolution of Symbolic Structures
Analytic Programming

- AP has gradually been tried and tested in **three versions**.
- All versions use the same set of functional and terminal object as it is also in GP (Koza, 1998; Koza et al., 1999).
- **The first** version is $\text{AP}_{\text{basic}}$ approached the synthesis of programs as well as the canonical GP, i.e. that together with the functions and operators were randomly generated and constants that were part of the synthesis process.
- It should be noted that the cardinality of the set of all possible combinations is big and it has grown thanks to amount generated constants.
- If the cardinality of the GFS is **5** objects, their number combination is $5!$ which is **120** possible programs.
- If we add to this **10** constants, then in symbolic form it can achieve up to $15!$ combination or **1 307 674 368 000** possible programs.
Therefore developed two more versions of AP: \( \text{AP}_{\text{meta}} \) and \( \text{AP}_{\text{nf}} \). In their case was to GFS Adding another single constant \( K (5! \text{ changed only to } 6! = 720) \), which was prior to the actual valuation of the unknown constants \( K_1, K_2, \ldots, K_n \). The formulas below demonstrates that shape the environment of \textit{Mathematica} syntax. After that they were already different constants numerically (version \( \text{AP}_{\text{nf}} \)).

\[
\begin{align*}
\mathbf{x} + \mathbf{x} K[1] & + \frac{(\mathbf{x} - K[3]^2) \left( -K[4] + \frac{x}{K[5]} + \frac{x^2}{K[6]} \right) \left( -\frac{x}{K[7]} - x^2 \left( -x + K[8] \right) \right)}{K[2]} \\
2.22045 \times 10^{-16} \mathbf{x} & + 0.604203 \left( -1. + x \right) \\
\left( -2.59916 \times 10^{-16} + 1.65507 \mathbf{x} + 1.65507 \mathbf{x}^2 \right) \left( -1. \mathbf{x} - \left( -4.98165 \times 10^{-9} - \mathbf{x} \right) \mathbf{x}^2 \right) \\
\mathbf{x} \left( 6.50027 \times 10^{-17} + 1. \mathbf{x} - \\
1.18557 \times 10^{-16} \mathbf{x}^2 - 2. \mathbf{x}^3 + 1.32469 \times 10^{-16} \mathbf{x}^4 + 1. \mathbf{x}^5 \right)
\end{align*}
\]
Evolution of Symbolic Structures
Analytic Programming

\[ f(x) = (6.50027 \times 10^{-17} + 1.0 \times x - 1.18557 \times 10^{-16} x^2 - 2.0 x^3 + 1.32469 \times 10^{-16} x^4 + 1.0 x^5) \]
The core of the second version, $\text{AP}_{\text{meta}}$, was a general improvement of constants selected by \textit{evolutionary program}, which turned out to be time consuming.

Since EA (subject - slave) "working under" EA (senior - master), the structure of the process is given by the following simplified regulation: $\text{EA}_{\text{master}} \rightarrow \text{the synthesis} \rightarrow \text{indexing} \rightarrow \text{EA}_{\text{slave}} \rightarrow K_n$ setting, this version is called the AP with meta-evolution, marked as $\text{AP}_{\text{meta}}$.

Another version known as $\text{AP}_{\text{nf}}$, which differs by the estimation of constants used suitable numerical methods (such as numerical fitting etc...).

Many experiments have shown that $\text{AP}_{\text{nf}}$ is a promising combination of the other two versions and the results achieved with it were often better than outcomes with GP.

Some results of comparative simulations with GP can be found e.g. in (Zelinka 2002; Zelinka, Oplatkova 2003, Zelinka et al., 2005) or (Oplatkova, Zelinka, 2006).
Evolution of Symbolic Structures
Analytic Programming

- The core of GFS is a hierarchical structure, i.e. it contains $GFS_{all}$ $GFS_{?Arg-0arg}$.
- This "feature" is used to avoid synthesis of pathological programs such as calling a wrong functions, division by zero, recursive calls (if not required), etc.
- As already indicated, the core of AP mapping is of basic symbolic objects into a set of possible programs.
- The display is performed by DSH techniques (Price, 1999), which allow the numerical manipulation of non-numerical objects (expressions, programs,...).
- Security procedures guarantee that no pathological programs are synthesized.
Use solution (in bold) without corrective procedures (left) and their increase after the application of correction procedures (right).
Evolution of Symbolic Structures
Analytic Programming

$$
\tan(csch^{-1}(t)) \quad \sinh\left(\tan^{-1}\left(\frac{e^{\cos(t)}}{\text{Khinchin}}\right)\right)
$$

$$
\frac{csch^{-1}(e^{Glaisher})}{\sqrt{1 - csch^{-1}(e^{Glaisher})^2}}
$$

$$
\sqrt{\frac{csc(e^{sech(cosh(cosh(sin(t))))})}{\pi}} \quad \sqrt{1 - \pi^2 \sin^2(e^{sech(cosh(cosh(sin(t))))})}
$$

$$
\sqrt{\frac{\sech^{-1}\left(\sech^{-1}\left(e^{\sinh^{-1}(\cot^{-1}\left(\csc\left(\sin^{-1}(t)\right)\right)}\right)\right)}{\sech^{-1}\left(e^{\sinh^{-1}(\cot^{-1}(Glaisher) + \frac{\log(\cos^{-1}(\gamma))}{\log(2 \gamma)})\right)}\right)}
$$

$$
\tan\left(\cot^{-1}\left(\frac{t}{\gamma}\right)\sin^{-1}(e^{\phi-t})\right)
$$
Evolution of Symbolic Structures
Analytic Programming

- A set of randomly generated individuals has been created.
- Its “behavior”, when visualized, generates very interesting behavior, that cannot be “normally” expected, as demonstrated on following graphs.

- Each of presented graph was generated by mathematical formula, not by program containing programmer instructions like If, For, etc...

Created by analytic programming

\[ f(x) \]

\[ x \]

Created by analytic programming

\[ f(x) \]

\[ x \]
Evolution of Symbolic Structures
Analytic Programming

Created by analytic programming

Created by analytic programming

Created by analytic programming

Created by analytic programming
Evolution of Symbolic Structures
Analytic Programming

Created by analytic programming

Created by analytic programming

Created by analytic programming

Created by analytic programming
Evolution of Symbolic Structures
Analytic programming - Crossover, Mutation and Other Evolutionary Operations

• During the operation of AP any evolutionary operations are obviously performed.
• It is the selection, crossover, mutation, etc.. It should be noted that these are fully used in the power of the evolutionary algorithm used in AP.
• In other words, the AP is only projection from space of integer individuals to the space of possible programs.
• Whether the AP is more or less success, depends mainly on the algorithm used.
• It can be expected (and experiments have confirmed this) that, for example when using a SA or DE along with the AP, the combination of DE + AP to perform better.
• All evolutionary operations thus relate to the algorithms used, not to AP itself.
• Performance is influenced by used EA.
During the basics simulations, it was found that the AP in the canonical version has not been able to show satisfactory results in a reasonable time. For this reason, the AP used a technique called “reinforced search”. It is based on idea of acceptation of partially good solutions. The introduction of useful information in the form of at least partially compliant to the basic set caused the AP gives performance comparable to, and often exceeding another techniques. If during the search is found program, whose fitness is even better, then the previous superseded by the new program. This leads to the fact that the cardinality of the set does not increase but increases the quality and speed of search process.
Start of simulation No. 5  Time: 22:7:54.351481
Appended suboptimal solution with CV = 4  Time: 22:7:54.547317
Appended suboptimal solution with CV = 3  Time: 22:7:54.577788
Appended suboptimal solution with CV = 2  Time: 22:8:41.529214
Appended suboptimal solution with CV = 1  Time: 22:10:14.346728

Number of cost function evaluations: 36270
Cost value: 0
The best individual:
{8, 5, 8, 3, 1, 9, 4, 2, 3, 9, 7, 4, 4, 10, 1, 10, 5, 8, 5, 7, 10, 6, 5, 10, 9, 9, 7, 2, 6, 3}
Solution: (((B \lor ((A \land C \land \neg B) \lor (B \land C \land \neg A) \lor (\neg A \land \neg B \land \neg C)) \land C)) \land (B \lor (C \land A))) \land
A \land ((A \lor ((A \land C \land \neg B) \lor (B \land C \land \neg A) \lor (\neg A \land \neg B \land \neg C)) \land C)) \land
((A \land C \land \neg B) \lor (B \land C \land \neg A) \lor (\neg A \land \neg B \land \neg C)) \lor
(A \lor ((A \lor ((A \land C \land \neg B) \lor (B \land C \land \neg A) \lor (\neg A \land \neg B \land \neg C)) \lor
((A \land C \land \neg B) \lor (B \land C \land \neg A) \lor (\neg A \land \neg B \land \neg C)) \lor
((A \land C \land \neg B) \lor (B \land C \land \neg A) \lor (\neg A \land \neg B \land \neg C)) \lor
((A \land C \land \neg B) \lor (B \land C \land \neg A) \lor (\neg A \land \neg B \land \neg C)) \lor
((A \land C \land \neg B) \lor (B \land C \land \neg A) \lor (\neg A \land \neg B \land \neg C)))))

{2005, 11, 28, 22, 14, 45.450393}

Reinforced learning in Mathematica – screen shot.
During development, the AP has done many different simulations that were designed to verify the viability of the AP, especially in comparison with GP. Simulations were initially focused on areas in which the GP was not always used, but gradually comparative studies were done with GP. They are for example:

- Randomly generated individuals and their transformation into programs. It was done in 1000 repeated simulations and no pathology program was identified.
- Trivial solution of differential equations: \( u''(t) = \cos(t), \ u'(0) = 1, \ u'(n) = -1, \ u'(0) = 0, \ u'(n) = 0 \) In this case, the synthesized function in which the solution of equation (Zelinka, 2002).
- Solving the differential equation of the civil engineering: \( ((4 + x) \ u''(x)) + 600U(x) = 5000 (x-x2), \ u(0) = 0, \ u(1) = 0, \ u(0) = 0, \ u'(1) = 0. \)
Evolution of Symbolic Structures
Analytic programming – reinforced learning

- Comparative study on Boolean problems according to (Koza, 1998), see (Zelinka et al., 2005).
- Comparative study on Sextic and Quintic problems (Koza, 1998), see (Zelinka, Oplatkova, 2003).
- Synthesis of robot trajectory according to (Koza, 1998) for a comparative study (Oplatkova, 2005; Oplatkova, Zelinka, 2006).
- Synthesis of evolutionary algorithms (Oplatkova, 2008).
- Controller synthesis.
- Chaotic systems synthesis and identification.
- ...
Analytic programming

- To run demonstration you need to download and install program [http://www.wolfram.com/cdf-player/](http://www.wolfram.com/cdf-player/).
- Follow instruction in selected demonstration program.
Evolution of Symbolic Structures
Hierarchical structure of algorithms - speculation

Space of a generalised evolutionary algorithms
EA synthesis by means of SR and other EAs

SR
GP
GE
AP

EAs
GA
SA
SS
SOMA
DE
ES
PS
Evolution of symbolic structures

Video
Want to know more?

- As a basic literature on genetic programming is recommended (Koza, 1998) and (Koza, 1999) for example.
- Grammatical evolution is well described in (O'Neill, 2003), (O'Sullivan, 2002).
- Analytical programming, as an experimental approach using a non-grammar in (Zelinka, Oplatková, Nolle, 2005) and (Oplatková, Zelinka, 2006).
- Moreover, you can still visit the home page of the relevant evolutionary techniques:
  - Analytical programming: www.ivanzelinka.eu.
Conclusions

- A special class of evolutionary algorithms that can handle with symbolic objects exists today.
- Synthesis of complex structures can be done from simple basic block elements.
- Evolvable hardware exists today and becomes usable technology.
- Methods like:
  - Genetic programming
  - Grammatical evolution
  - Analytic programming
have been introduced.
THANK YOU FOR YOUR ATTENTION

ivan.zelinka@ieee.org
www.ivanzelinka.eu
Copyright

This didactic material is meant for the personal use of the student only, and is copyrighted. Its reproduction, even for a partial utilization, is strictly forbidden in compliance with and in force of the law on Authors rights.

Copyright©NAVY.CS.VSB.CZ