Overview

Genetic Programming: Syntax & Semantics

1. Setting the Stage
   - What is Natural Computing?
   - What is Evolutionary Computation?
   - An Introduction to Genetic Programming (GP)

2. Grammar-based GP

3. Semantic methods & Open Issues in GP
An Introduction to Genetic Programming
Genetic Programming

Automatic Programming

- Assemblers;
- Compilers 2GL;
- Automatic Parallelisation.

Arthur Samuel, 1959:

"Tell the computer what to do, not how to do it."
Genetic Programming

John Koza’s AP attributes...

- Start with **high-level problem description** that results in a solution in the form of a computer program;
- Automatically determine the program’s **size and architecture**;
- Automatically organise a group of **instructions** so that they may be **re-used** by a program;
- **Problem-independence**;
- **Scalability** to larger versions of the same problem;
- Capability of producing **human competitive results**;
- Evolutionary Automatic Programming / Genetic Programming.
Genetic Programming

Individual is OR represents/encodes a program

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char* argv){
    float x=0.0, y=0.0, z=0.0;
    x=atof(argv[1]);
    y=atof(argv[2]);
    z=atof(argv[3]);
    x = 2.0*sin(y) + 4.0*sin(x);
    z = (x*x) + exp(z);
    printf("The answer is: z=%f\n",z);
    return(0);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void turnLeft(float degrees);
void turnRight(float degrees);
void moveForward(float distance);

int main(int argc, char* argv){
    turnLeft(90);
    if(sensorValue(0) > 1000)
        moveForward(10);
    else
        turnRight(90);
    return(0);
}
```
Recall GAs

- Binary string;
- Fixed-length chromosomes;
- Problem specific encoding.
- But how to encode a program?...
The Road to GP

- Historical Perspective on Representation
  - Friedberg (1958/1959);
  - Cramer (1985);

- Distinguishing features:
  - Variable-length;
  - GP operates on solution directly.
GP trees

- Koza (1992) popularised Lisp S-expressions;
- Expressions (trees) generated from:
  - *Function Set*: boolean, arithmetic, loops, user-defined functions...
  - *Terminal Set*: inputs, constants, variables...
Different types of functions/terminals

- Conditionals;
- `(if (> x 3.14) x y)`
- `IF 'condition' THEN 'return X' ELSE 'return y'`
Representation

Features

- Sufficiency:
  - Function + Terminal sets: powerful enough to represent a solution.

- Parsimony:
  - Smaller Function + Terminal sets are better.

- Closure:
  - Each function should gracefully handle all values it ever receives;
  - ( / 5 0 ) ?!
Initialisation

Tree Creation

- Max Depth - Maximum Program Size;
- **Grow** and **Full** initialisation;
- Desire structural diversity:
  - Ramped half-and-half.
Initialisation
Genetic Operators

Operators

- Crossover;
- Mutation;
- Reproduction.

Diagram showing the process of crossover with labeled points and genetic strings.
Genetic Operators

### Crossover

1. Select two parents;
2. For first parent, randomly pick node from 1 to $n$;
3. Independently pick node from 1 to $n$ for second parent;
4. Swap sub-trees.
Crossover

\[ (x \times y) \text{ IFLTE } \frac{3.2}{0.4} + (x - y \times x) \]

\[ + \]

\[ 6.2 + (x - y \times x) \]
Crossover

```
+      +
|      |
|      |
y + x

IFLTE

x - y

3.2 0.4

6.2

x * y
```
Genetic Operators

**Mutation**

1. Randomly pick node from 1 to $n$;
2. Delete subtree at selected node;
3. Grow new subtree (as per initialisation).
Mutation

\[
\begin{align*}
&\text{Mutation} \\
&\text{IFLTE} + x - y \\
&\text{IFLTE} + x - y/2.1 x
\end{align*}
\]
**Problem Definition**

- Find function that gives output \( y \) for specific input \( x \);
- \( y = f(x) \);
- E.g. \( f(x) = x^2 + x \)

**Preparatory Steps**

- Define function and terminal set;
- Define fitness measure;
- Define evolutionary parameters:
  - Max-depth, popsize, xover prob., mut. prob., reproduction prob., selection & replacement strategies, termination criteria.
Fitness Measure

Fitness Function

- Problem specific;
- Design to give graded and continuous feedback for selection:
  - Continuous fitness function;
  - Standardised fitness;
  - Normalised fitness.
**Fitness Measure**

\[ f(x) = x^2 + x \]

**Fitness Function**

<table>
<thead>
<tr>
<th>case #</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>90</td>
</tr>
</tbody>
</table>

- Simple discrete form: \( f_p = \sum_{i=1}^n (p_i = o_i) \)
- Simple continuous form: \( f_p = \sum_{i=1}^n |p_i - o_i| \)
- Mean Squared Error: \( f_p = \frac{\sum_{i=1}^n (p_i - o_i)^2}{n} \)
Fitness Measure

## Fitness Function

<table>
<thead>
<tr>
<th>case #</th>
<th>Input</th>
<th>Output ($o_i$)</th>
<th>Prediction ($p_i$)</th>
<th>Error</th>
<th>Squared Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>20</td>
<td>16</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>56</td>
<td>49</td>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>90</td>
<td>81</td>
<td>9</td>
<td>81</td>
</tr>
</tbody>
</table>

\[
23 \quad 151 / 5 = 30.2
\]

- Different fitness functions = different fitness landscapes;
- Co-evolution:
  - No explicit fitness value;
  - Fitness relative to other individuals.
- Multi-objective fitness functions.
### Toy Problems

- Symbolic Regression;
- Artificial Ant;
- Intertwined Spirals;
- Broom Balancer;
- Block Stacking;
- Cellular Automata;
- Image Compression;
- Box Mover;
- Boolean Function Learning;
- ...

### Applications

- Human Competitive non-patent results;
- 20th Century Patents;
- 21st Century Patents;
- New Patented Inventions.
## GP 1987-2002

<table>
<thead>
<tr>
<th>System</th>
<th>Dates</th>
<th>Speed up</th>
<th>HC Results</th>
<th>Problem Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial LISP</td>
<td>1987-1994</td>
<td>1 (base)</td>
<td>0</td>
<td>Toy Problems</td>
</tr>
<tr>
<td>64 transputer</td>
<td>1994-1997</td>
<td>9</td>
<td>2</td>
<td>human-competitive results not patent related</td>
</tr>
<tr>
<td>64 PowerPC</td>
<td>1995-2000</td>
<td>204</td>
<td>12</td>
<td>20\textsuperscript{th} Century Patented Inventions</td>
</tr>
<tr>
<td>70 Alpha</td>
<td>1999-2001</td>
<td>1,481</td>
<td>2</td>
<td>20\textsuperscript{th} Century Patented Inventions</td>
</tr>
<tr>
<td>1,000 Pentium II</td>
<td>2000-2002</td>
<td>13,900</td>
<td>12</td>
<td>21\textsuperscript{st} Century Patented Inventions</td>
</tr>
<tr>
<td>4-week runs on Pentium IIs</td>
<td>2002-2003</td>
<td>130,000</td>
<td>2</td>
<td>Patentable new inventions</td>
</tr>
</tbody>
</table>
## Human-Competitive Results (non-Patent)

<table>
<thead>
<tr>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmembrane segment identification problem for proteins</td>
</tr>
<tr>
<td>Motifs for DEAD box family and manganese superoxide dismutase family of proteins</td>
</tr>
<tr>
<td>Cellular automata rule for Gacs-Kurdyumov-Levin (GKL) problem</td>
</tr>
<tr>
<td>Quantum algorithm for the Deutsch-Jozsa early promise problem</td>
</tr>
<tr>
<td>Quantum algorithm for Grovers database search problem</td>
</tr>
<tr>
<td>Quantum algorithm for the depth-two AND/OR query problem</td>
</tr>
<tr>
<td>Quantum algorithm for the depth-one OR query problem</td>
</tr>
<tr>
<td>Protocol for communicating information through a quantum gate</td>
</tr>
<tr>
<td>Quantum dense coding</td>
</tr>
<tr>
<td>Soccer-playing program that won its first two games in the 1997 Robo Cup competition</td>
</tr>
<tr>
<td>Soccer-playing program that ranked in the middle of field in 1998 Robo Cup competition</td>
</tr>
<tr>
<td>Antenna designed by NASA for use on spacecraft</td>
</tr>
<tr>
<td>Sallen-Key filter</td>
</tr>
<tr>
<td>20\textsuperscript{th} Century Patents</td>
</tr>
<tr>
<td>---------------------------------------</td>
</tr>
<tr>
<td>Campbell ladder topology for filters</td>
</tr>
<tr>
<td>Zobel M-derived half section and constant K filter sections</td>
</tr>
<tr>
<td>Crossover filter</td>
</tr>
<tr>
<td>Negative feedback</td>
</tr>
<tr>
<td>Cauer (elliptic) topology for filters</td>
</tr>
<tr>
<td>PID and PID-D2 controllers</td>
</tr>
<tr>
<td>Darlington emitter-follower section and voltage gain stage</td>
</tr>
<tr>
<td>Sorting network for seven items using only 16 steps</td>
</tr>
<tr>
<td>60 and 96 decibel amplifiers</td>
</tr>
<tr>
<td>Analog computational circuits</td>
</tr>
<tr>
<td>Real-time analog circuit for time-optimal robot control</td>
</tr>
<tr>
<td>Electronic thermometer</td>
</tr>
<tr>
<td>Voltage reference circuit</td>
</tr>
<tr>
<td>Philbrick circuit</td>
</tr>
<tr>
<td>NAND circuit</td>
</tr>
<tr>
<td>Simultaneous synthesis of topology, sizing, placement, and routing</td>
</tr>
<tr>
<td>Patent Title</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Low-voltage balun circuit</td>
</tr>
<tr>
<td>Mixed analog-digital variable capacitor circuit</td>
</tr>
<tr>
<td>High-current load circuit</td>
</tr>
<tr>
<td>Voltage-current conversion circuit</td>
</tr>
<tr>
<td>Cubic function generator</td>
</tr>
<tr>
<td>Tunable integrated active filter</td>
</tr>
</tbody>
</table>
GP Literature

Sample of references...

Grammar-based Genetic Programming
Alternative GP Representations

Representations

▶ Various explored since trees;
▶ Graphs (PADO);
▶ Linear (Friedberg, Cramer, CGP and DGP);
▶ Grammars:
  ▶ Tree-based (G³P);
  ▶ Linear (GADS, GE).
Alternative Representations

Grammars

- Backus Naur Form (BNF);
- BNF Grammar a 4-tuple \( < T, N, P, S > \):
  - \( T \): Terminal Set;
  - \( N \): Non-terminal Set;
  - \( P \): Set of Production Rules;
  - \( S \): Start Symbol (a member of \( N \)).
BNF Example

\[ T = \{ \sin, \cos, \tan, \log, +, -, /, *, X, (, ) \} \]
\[ S = \langle \text{expr} \rangle \]
\[ N = \{ \langle \text{expr} \rangle, \langle \text{op} \rangle, \langle \text{pre-op} \rangle, \langle \text{var} \rangle \} \]
\[ P = \]

\[ \langle \text{expr} \rangle ::= (\langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{expr} \rangle) \]
\[ \text{expr} ::= \langle \text{op} \rangle \langle \text{expr} \rangle \langle \text{expr} \rangle \]
\[ \langle \text{op} \rangle ::= + | - | / | * \]

\[ \langle \text{pre-op} \rangle ::= \sin | \cos | \tan | \log \]

\[ \langle \text{var} \rangle ::= x \]
Developmental GP

- Wolfgang Banzhaf;
- Linear, fixed-length, binary chromosomes;
- Genotype-Phenotype Mapping;
- Binary Codes for each Symbol in function and terminal sets;
- $n$-bit code - a codon.
Developmental GP

<table>
<thead>
<tr>
<th>Codon</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>a</td>
</tr>
<tr>
<td>001</td>
<td>b</td>
</tr>
<tr>
<td>010</td>
<td>+</td>
</tr>
<tr>
<td>011</td>
<td>*</td>
</tr>
<tr>
<td>100</td>
<td>a</td>
</tr>
<tr>
<td>101</td>
<td>b</td>
</tr>
<tr>
<td>110</td>
<td>+</td>
</tr>
<tr>
<td>111</td>
<td>*</td>
</tr>
</tbody>
</table>

- 000010101 represents $a + b$;
- Repair illegal raw sequences:
  - Editing.
- Determine legal symbol set;
- Determine minimal distance set;
- Symbol with lowest int values used.
**Developmental GP**

<table>
<thead>
<tr>
<th>Codon</th>
<th>Symbol</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>a</td>
<td>000 001 011 gives $ab^*$;</td>
</tr>
<tr>
<td>001</td>
<td>b</td>
<td>a is ok, b is illegal:</td>
</tr>
<tr>
<td>010</td>
<td>+</td>
<td>Look up &lt;op&gt; in grammar;</td>
</tr>
<tr>
<td>011</td>
<td>$*$</td>
<td>Nearest to b (001) is $*$ (011).</td>
</tr>
<tr>
<td>100</td>
<td>a</td>
<td>a * $*$:</td>
</tr>
<tr>
<td>101</td>
<td>b</td>
<td>Second $*$ illegal;</td>
</tr>
<tr>
<td>110</td>
<td>+</td>
<td>Look up &lt;var&gt;;</td>
</tr>
<tr>
<td>111</td>
<td>$*$</td>
<td>Closest to 011 is b (001);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a * b.</td>
</tr>
</tbody>
</table>
G³P

- Grammar Guided Genetic Programming;

- Use Derivation Trees:
  - Crossover: match NT symbol (no match, no XO);
  - Mutation: Replace with random derivation sequence.

```plaintext
<expr> ::= <expr> <op> <expr>
  | <pre-op> <expr>
  | <var>
<op> ::= + | *
<pre-op> ::= sin | cos
<var> ::= 1.0 | x
```
Grammatical Evolution

Grammatical Genetic Programming

- Chromosomes:
  - Linear;
  - Binary/Integer;
  - Variable-length.

- Genotype-Phenotype Map;

- Bio-inspired.

Genotype-Phenotype Map

**Grammatical Evolution**

- Binary String
- Integer String
- Rules
- Program/Function
- Executed Program

**Biological System**

- DNA
- RNA
- Amino Acids
- Protein
- Phenotypic Effect

**Transcription**

**Translation**
## Genetic Code

<table>
<thead>
<tr>
<th></th>
<th>U</th>
<th>C</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>UUU - Phe</td>
<td>UCU - Ser</td>
<td>UAU - Tyr</td>
<td>UGU - Cys</td>
</tr>
<tr>
<td></td>
<td>UUC - Phe</td>
<td>UCC - Ser</td>
<td>UAC - Tyr</td>
<td>UGC - Cys</td>
</tr>
<tr>
<td></td>
<td>UUA - Leu</td>
<td>UCA - Ser</td>
<td>UAA - Stop</td>
<td>UGA - Stop</td>
</tr>
<tr>
<td></td>
<td>UUG - Leu</td>
<td>UCG - Ser</td>
<td>UAG - Stop</td>
<td>UGG - Trp</td>
</tr>
<tr>
<td>C</td>
<td>CUU - Leu</td>
<td>CCU - Pro</td>
<td>CAU - His</td>
<td>CGU - Arg</td>
</tr>
<tr>
<td></td>
<td>CUC - Leu</td>
<td>CCC - Pro</td>
<td>CAC - His</td>
<td>CGC - Arg</td>
</tr>
<tr>
<td></td>
<td>CUA - Leu</td>
<td>CCA - Pro</td>
<td>CAU - Gln</td>
<td>CGA - Arg</td>
</tr>
<tr>
<td></td>
<td>CUG - Leu</td>
<td>CCG - Pro</td>
<td>CAG - Gln</td>
<td>CGG - Arg</td>
</tr>
<tr>
<td>A</td>
<td>AUU - Ile</td>
<td>ACU - Thr</td>
<td>AAU - Asn</td>
<td>AGU - Ser</td>
</tr>
<tr>
<td></td>
<td>AUC - Ile</td>
<td>ACC - Thr</td>
<td>AAC - Asn</td>
<td>AGC - Ser</td>
</tr>
<tr>
<td></td>
<td>AUA - Ile</td>
<td>ACA - Thr</td>
<td>AAA - Lys</td>
<td>AGA - Arg</td>
</tr>
<tr>
<td></td>
<td>AUG - Met</td>
<td>ACG - Thr</td>
<td>AAG - Lys</td>
<td>AGG - Arg</td>
</tr>
<tr>
<td>G</td>
<td>GUU - Val</td>
<td>GCU - Ala</td>
<td>GAU - Asp</td>
<td>GGU - Gly</td>
</tr>
<tr>
<td></td>
<td>GUC - Val</td>
<td>GCC - Ala</td>
<td>GAC - Asp</td>
<td>GGC - Gly</td>
</tr>
<tr>
<td></td>
<td>GUA - Val</td>
<td>GCA - Ala</td>
<td>GAA - Glu</td>
<td>GGA - Gly</td>
</tr>
<tr>
<td></td>
<td>GUG - Val</td>
<td>GCG - Ala</td>
<td>GAG - Glu</td>
<td>GGG - Gly</td>
</tr>
</tbody>
</table>

### Code Name

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phe</td>
<td>Phenylalanine</td>
<td>Leu</td>
<td>Leucine</td>
</tr>
<tr>
<td>Tyr</td>
<td>Tyrosine</td>
<td>Cys</td>
<td>Cysteine</td>
</tr>
<tr>
<td>Trp</td>
<td>Tryptophan</td>
<td>Pro</td>
<td>Proline</td>
</tr>
<tr>
<td>His</td>
<td>Histidine</td>
<td>Glu</td>
<td>Glutamine</td>
</tr>
<tr>
<td>Arg</td>
<td>Arginine</td>
<td>Ile</td>
<td>Isoleucine</td>
</tr>
<tr>
<td>Met</td>
<td>Methionine</td>
<td>Thr</td>
<td>Thrreonine</td>
</tr>
<tr>
<td>Asn</td>
<td>Asparagine</td>
<td>Lys</td>
<td>Lysine</td>
</tr>
<tr>
<td>Ser</td>
<td>Serine</td>
<td>Val</td>
<td>Valine</td>
</tr>
<tr>
<td>Ala</td>
<td>Alanine</td>
<td>Asp</td>
<td>Aspartic Acid</td>
</tr>
<tr>
<td>Glu</td>
<td>Glutamic Acid</td>
<td>Gly</td>
<td>Glycine</td>
</tr>
</tbody>
</table>
### Genetic Code

**GENETIC CODE**

<table>
<thead>
<tr>
<th>Codon</th>
<th>Amino Acid</th>
</tr>
</thead>
<tbody>
<tr>
<td>G G C</td>
<td>Glycine</td>
</tr>
<tr>
<td>G G A</td>
<td></td>
</tr>
<tr>
<td>G G G</td>
<td></td>
</tr>
</tbody>
</table>

(A group of 3 Nucleotides)

**PARTIAL PHENOTYPE**

<table>
<thead>
<tr>
<th>Codon</th>
<th>Amino Acid</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000010</td>
<td></td>
</tr>
<tr>
<td>00010010</td>
<td></td>
</tr>
<tr>
<td>00100010</td>
<td></td>
</tr>
</tbody>
</table>

**GE Codon**

<table>
<thead>
<tr>
<th>GE Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;line&gt;</td>
</tr>
</tbody>
</table>

For a rule with 2 choices, e.g.,

\[
\text{<code>} :: = \text{<line>} (0) \\
1 \text{<code>_<line>} (1)
\]

i.e. (GE Codon Integer Value) MOD 2 = Rule Number
Example Mapping

\[
\begin{align*}
\text{A} & : \langle \text{seq}\rangle ::= \langle \text{vowel}\rangle \\
& \quad | \langle \text{seq}\rangle\langle \text{vowel}\rangle \\
\text{B} & : \langle \text{vowel}\rangle ::= \text{a} \\
& \quad | \text{e} \\
& \quad | \text{i} \\
& \quad | \text{o} \\
& \quad | \text{u} \\
\end{align*}
\]

\[
\begin{align*}
00101011011101101000101100010001 & \\
\text{00101011011101101000101100010001} & \\
\text{43 118 144 17} & \\
\end{align*}
\]

\[
\begin{align*}
\langle \text{seq}\rangle & \quad 43 \mod 2 = 1 \\
\langle \text{seq}\rangle\langle \text{vowel}\rangle & \quad 118 \mod 2 = 0 \\
\langle \text{vowel}\rangle\langle \text{vowel}\rangle & \quad 144 \mod 5 = 4 \\
u\langle \text{vowel}\rangle & \quad 17 \mod 5 = 2 \\
\text{ui} & \\
\end{align*}
\]
Symbolic Regression Grammar

\[ S = <expr> \]

\[ P = <expr> ::= <expr> <op> <expr> \]
\[ \quad | ( <expr> <op> <expr> ) \]
\[ \quad | <pre-op> (<expr>) \]
\[ \quad | <var> \]

\[ <op> ::= + | - | / | * \]

\[ <pre-op> ::= \text{sin} | \text{cos} | \text{tan} | \text{log} \]

\[ <var> ::= x \]
Wrapper

\[
\begin{align*}
\text{<func> ::= <header> } \\
\text{<header> ::= float symbreg(float x) \{ <body> \} } \\
\text{<body> ::= <declarations> <code> <return> } \\
\text{<declarations> ::= float a; } \\
\text{<code> ::= a = <expr>; } \\
\text{<return> ::= return (a); } \\
\end{align*}
\]

float symbreg(float x){
    float a;
    a = <expr>;
    return(a);
}

\[
\begin{align*}
\text{<code> ::= <line>; } \\
\text{\quad | <line>; <code> } \\
\text{<line> ::= ...}
\end{align*}
\]
Example Mapping

\[
\begin{align*}
\text{<expr>} & ::= \text{<expr> <op> <expr>} \\
& \quad | (\text{<expr> <op> <expr>}) \\
& \quad | \text{<pre-op>} (\text{<expr>}) \\
& \quad | \text{<var>}
\end{align*}
\]

\[
\begin{align*}
\text{<op>} & ::= + \\
& \quad | - \\
& \quad | / \\
& \quad | *
\end{align*}
\]

\[
\begin{align*}
\text{<pre-op>} & ::= \sin \\
& \quad | \cos \\
& \quad | \tan \\
& \quad | \log
\end{align*}
\]

\[
\begin{align*}
\text{<var>} & ::= x
\end{align*}
\]

\[
\begin{align*}
220 \mod 4 &= 0 \\
203 \mod 4 &= 3 \\
51 \mod 4 &= 3 \\
123 \mod 4 &= 3
\end{align*}
\]
Example Mapping

\[
\text{Example Mapping}
\]

\[
\begin{align*}
\text{<expr> ::= } & <\text{expr}> <\text{op}> <\text{expr}> \\
& (\text{<expr> <op> <expr>}) \\
& \text{<pre-op>} (<\text{expr}>) \\
& \text{<var>}
\end{align*}
\]

\[
\begin{align*}
\text{<op> ::= } & + \\
& - \\
& / \\
& *
\end{align*}
\]

\[
\begin{align*}
\text{<pre-op> ::= } & \text{sin} \\
& \text{cos} \\
& \text{tan} \\
& \text{log}
\end{align*}
\]

\[
\begin{align*}
\text{<var> ::= } & x
\end{align*}
\]

\[
\begin{align*}
220 \% 4 = & 0 \\
203 \% 4 = & 3 \\
51 \% 4 = & 3 \\
220 \% 4 = & 0 \\
203 \% 4 = & 3 \\
\end{align*}
\]

\[
\begin{align*}
\text{<expr><op><expr>} & \\
\text{x<op><expr>} & \\
\text{x*<expr>} & \\
\text{x*<expr><op><expr>} & \\
\text{x*<var><op><expr>} & \\
\end{align*}
\]

http://ncra.ucd.ie
Grammatical Evolution

Genetic Operators
- Binary/Integer String (variable-length);
- Bit/Codon Mutation;
- 1pt Crossover;
- Duplication;
- Tree-based Operators.
Symbolic Regression Grammar

\[ F(x) = x + x^2 + x^3 + x^4 \]
Santa Fe Ant Trail

Definition

- Instructions:
  - move();
  - left();
  - right();
  - if_food_ahead();

- Fitness: pieces of food eaten within 600 time steps.
Santa Fe Ant Trail Grammar

<code> ::= <line>
    | <code><line>
<line> ::= <if-statement>
    | <op>
<if-statement> ::= if(food_ahead()){
    <line>
        }
        else{
    <line>
        }
<op> ::= left();
    | right();
    | move();
Sorting Algorithm Grammar

```
<code> ::= <for> | <for><code>
<for> ::= "\n"i=0"\n"
        for a in x:"\n"i=1
        <for_a_in_x_line>"\n"
<for_a_in_x_line> ::= <for_a_in_x_setoutput> | <for_a_in_x_cond> | <for_b_in_x>
<for_a_in_x_setoutput> ::= guess[[<for_a_in_x_index>]] = <for_a_in_x_inputvar>"\n"
<for_a_in_x_index> ::= 1 | ((i <biop> <const>)%TOTAL)
<for_a_in_x_inputvar> ::= x[[<for_a_in_x_index>]]
<for_a_in_x_outputvar> ::= guess[[<for_a_in_x_index>]]
<for_a_in_x_cond> ::= "\n"if <for_a_in_x_expr><relop><for_a_in_x_expr> : <for_a_in_x_setoutput>
<for_a_in_x_expr> ::= <for_a_in_x_inputvar> | <for_a_in_x_outputvar>
<biop> ::= + | -
<relop> ::= < | >
<const> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<for_b_in_x> ::= j=0"\n"
        for b in x:"\n"j=1
        <for_b_in_x_line>"\n"
<for_b_in_x_line> ::= <for_b_in_x_setoutput> | <for_b_in_x_cond>
<for_b_in_x_setoutput> ::= guess[[<for_b_in_x_index>]] = <for_b_in_x_inputvar>"\n"
<for_b_in_x_index> ::= (i <biop> <const>)%TOTAL) | j | (j <biop> <const>)%TOTAL
<for_b_in_x_inputvar> ::= x[[<for_b_in_x_index>]]
<for_b_in_x_outputvar> ::= guess[[<for_b_in_x_index>]]
<for_b_in_x_cond> ::= "\n"if <for_b_in_x_expr><relop><for_b_in_x_expr> : <for_b_in_x_setoutput>
<for_b_in_x_expr> ::= <for_b_in_x_inputvar> | <for_b_in_x_outputvar>

i=0
for a in x :
    j=0
    for b in x :
        guess[i] = x[j+1]
        j+1
    i+1

i=0
for a in x :
    if guess[i+1]>x[i]: guess[i] = x[i]
    i+1
```
Sorting Algorithm Results

```python
i=0
for a in x :
    j=0
    for b in x :
        if guess[j]>guess[i] : swap(((j - 0)%TOTAL),i)
            j+=1
    i+=1
```
Caching Algorithm Grammar

\[
\begin{align*}
\text{<stmt> ::= <stmt><stmt>}
\mid \text{if(<expr>{<stmt>};} \text{else{<stmt>}}
\mid \text{write_x(<expr>,<expr>);
\mid \text{victim = <expr>;<term> | <term>+<term> | <term>-<term>}
\mid \text{<term>*<term> | div(<term>,<term>)
\mid \text{rem(<term>,<term>)
\text{<term> ::= CACHESIZE | <num> | <fun> | (<expr>)
\text{<num> ::= <mant> | <mant><zeros>
\text{<mant> ::= 0 | 1 | 2 | 3 | 4 | 5
\text{<zeros> ::= 0 | 0<zeros>
\text{<fun> ::= counter()}
\mid \text{read_x(<expr>)
\mid \text{small_x()}
\mid \text{large_x()}
\mid \text{random_x()}
\end{align*}
\]

- Input data is Trace File
- Training vs Test Data
- Fitness = #hits/#misses OR #runs - #misses

**GE1:**
\[
\text{victim = counter() - CACHESIZE;}
\]

**GE2:**
\[
\text{write_x(CACHESIZE + counter(), CACHESIZE + counter())
\text{victim = CACHESIZE + counter();}
\]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ken2.00100</th>
<th>ken2.00200</th>
<th>Average of % Gain over LRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU(20)</td>
<td>374,596</td>
<td>380,041</td>
<td>-</td>
</tr>
<tr>
<td>LRU(200)</td>
<td>367,104</td>
<td>373,935</td>
<td>-</td>
</tr>
<tr>
<td>GE1(20)</td>
<td>300,569</td>
<td>318,444</td>
<td>17.97</td>
</tr>
<tr>
<td>GE1(200)</td>
<td>106,067</td>
<td>82,856</td>
<td>74.51</td>
</tr>
<tr>
<td>GE2(20)</td>
<td>300,569</td>
<td>318,444</td>
<td>17.97</td>
</tr>
<tr>
<td>GE2(200)</td>
<td>106,067</td>
<td>82,856</td>
<td>74.51</td>
</tr>
</tbody>
</table>
GE Components

- Grammar
- Search Engine
- Objective Function

Many important questions to address...

1 Neutral Evolution
Shape Grammars


Tree-adjunct Grammars

Grammar:
\[ e ::= e < o > e \mid v \]
\[ o ::= + \mid - \]
\[ v ::= X \mid Y \]

Chromosome:
12, 3, 7, 15, 9, 36, 14

Fig. 1. Sample GE grammar, chromosome and resulting derivation tree (edge labels indicating the order of expansion). <> denotes a non-terminal symbol.

(a) \( I = \{\alpha_0, \alpha_1\} \)

(b) \( A = \{\beta_0, \beta_1, \beta_2...\} \)

Fig. 2. The initial tree set \( I \) and a subset of the auxiliary tree set \( A \) of the TAG produced from the CFG in Fig. 1.
Tree-adjunct Grammars

Fig. 3. The derivation tree (left) and derived tree (right) throughout TAGE derivation. The shaded areas indicate new content added at each step.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GE</td>
<td>79</td>
<td>3</td>
<td>44</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>TAGE</td>
<td>88</td>
<td>12</td>
<td>76</td>
<td>63</td>
<td></td>
</tr>
</tbody>
</table>
Tree-adjunct Grammars

Figure 7: Frequency Maps: Symbolic Regression (a) (b); Six Multiplexer (c)
GE Components

- Grammar
- Search Engine
- Objective Function

Output Structure
Alternative Mapping

- $\pi$GE
- Artificial Genetic Regulatory Networks
Evolve the mapping order

Fig. 3. NT selection process in πGE

1. [(e)] <- (12%1=0)
2. [(e), o, e] <- (3%3=0)
3. [o, (e), v] <- (7%3=1)
4. [o, (v), e, o, e] <- (11%5=1)
5. [(o), e, o, e] <- (4%4=0)
6. [(e), o, e] <- (3%3=0)
7. [(o), e, v] <- (15%3=0)
8. [e, (v)] <- (9%2=1)
9. [(e)] <- (10%1=0)
10. [(v)] <- (7%1=0)

Fig. 4. Standard πGE Genotype to Phenotype Mapping
A distribution of orders
Artificial GRN's
The expression of a gene is regulated using the bit signatures of the enhancer and inhibitor sites and the concentration of each protein which currently exists in the model. The enhance \((e_i)\) and inhibit \((h_i)\) signals of gene \(i\) are calculated using:

\[
e_i = \frac{1}{N} \sum_{j=1}^{N} c_j \exp(\beta(u_j - u_{\text{max}})) \quad (1)
\]

\[
h_i = \frac{1}{N} \sum_{j=1}^{N} c_j \exp(\beta(u_j - u_{\text{max}})) \quad (2)
\]

where \(c_j\) is the concentration of protein \(j\), \(u_j\) is the number of complementary bits between the protein \(j\) and either the enhancer or inhibitor site of gene \(i\), and \(u_{\text{max}}\) is the maximum observed number of complementary bits between proteins and regulatory sites. \(N\) is the total number of proteins in the model, and \(\beta\) is a scaling factor which controls the significance of the protein/regulatory site match difference.

The expression of a protein at any point in time \((p_i)\) is then calculated according to (3):

\[
\frac{dc_i}{dt} = \delta(e_i - h_i)c_i - \phi(1.0) \quad (3)
\]

where \(\phi(1.0)\) is a term that scales protein production such that the sum of all protein concentrations is equal to 1.0, and \(\delta\) is a scaling factor.
Algorithm 21.1: Artificial Genetic Regulatory Model for GP

1. initialise population
2. for each generation do
   1. for each individual do
      1. execute GRN model until it reaches a steady state;
      2. for each input do
         1. set values of input gene(s);
         2. execute GRN model for time period t;
         3. read values of output gene(s);
         4. update fitness;
      end
   end
   1. select parents;
   2. apply genetic operators;
   3. replacement;
end
GRN’s for Financial Modelling

\[ m_{Avg}(10) = \text{[values]} \]
\[ s_{Osc}(10) = \text{[values]} \]
\[ m_{Change}(5) = \text{[values]} \]
\[ m_{Change}(10) = \text{[values]} \]
### GRN’s for Financial Modelling

<table>
<thead>
<tr>
<th>FTSE market</th>
<th></th>
<th></th>
<th>Avg. daily inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period (days)</td>
<td>Buy &amp; Hold</td>
<td>Best-of-run</td>
<td></td>
</tr>
<tr>
<td>Train (75 to 439)</td>
<td>-1269.28</td>
<td>3275.96</td>
<td>5939.73</td>
</tr>
<tr>
<td>Test 1 (440 to 804)</td>
<td>4886.9</td>
<td>1083.58</td>
<td>2191.78</td>
</tr>
<tr>
<td>Test 2 (805 to 1169)</td>
<td>-1089.8</td>
<td>541.806</td>
<td>3709.59</td>
</tr>
<tr>
<td>Test 3 (1170 to 1534)</td>
<td>1908.53</td>
<td>500.949</td>
<td>2487.67</td>
</tr>
<tr>
<td>Total</td>
<td><strong>4436.35</strong></td>
<td><strong>5402.295</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nikkei market</th>
<th></th>
<th></th>
<th>Avg. daily inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period (days)</td>
<td>Buy &amp; Hold</td>
<td>Best-of-run</td>
<td></td>
</tr>
<tr>
<td>Train (75 to 439)</td>
<td>-6345.5</td>
<td>6163.38</td>
<td>5128.77</td>
</tr>
<tr>
<td>Test 1 (440 to 804)</td>
<td>1014.79</td>
<td>1125.6</td>
<td>1457.53</td>
</tr>
<tr>
<td>Test 2 (805 to 1169)</td>
<td>-5263.49</td>
<td>2144.71</td>
<td>3679.45</td>
</tr>
<tr>
<td>Test 3 (1170 to 1534)</td>
<td>4040.59</td>
<td>1331.56</td>
<td>961.644</td>
</tr>
<tr>
<td>Total</td>
<td><strong>-6553.61</strong></td>
<td><strong>8514.05</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dax market</th>
<th></th>
<th></th>
<th>Avg. daily inv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Period (days)</td>
<td>Buy &amp; Hold</td>
<td>Best-of-run</td>
<td></td>
</tr>
<tr>
<td>Train (75 to 439)</td>
<td>-882.241</td>
<td>2899.86</td>
<td>4586.3</td>
</tr>
<tr>
<td>Test 1 (440 to 804)</td>
<td>4047.63</td>
<td>952.689</td>
<td>3347.95</td>
</tr>
<tr>
<td>Test 2 (805 to 1169)</td>
<td>-551.995</td>
<td>608.161</td>
<td>1471.23</td>
</tr>
<tr>
<td>Test 3 (1170 to 1534)</td>
<td>2972.24</td>
<td>992.868</td>
<td>3495.89</td>
</tr>
<tr>
<td>Total</td>
<td><strong>5585.634</strong></td>
<td><strong>5453.578</strong></td>
<td></td>
</tr>
</tbody>
</table>
GRN’s for Financial Modelling

Fig. 3. Best evolved trader for the FTSE index. Blue dots indicating buy, do nothing and sell signals are plotted along with the raw market prices.
GRN’s + Tree-adjunct Grammars

The diagram illustrates the relationship between Gene Regulatory Networks (GRNs) and Tree-adjunct Grammars (TAGE). GRNs are represented by a series of genes (gene1, gene2, gene3, gene4, gene5, ..., geneN) with directed relationships indicating the flow of proteins (input-proteins to input signal, product-proteins). TAGE is depicted as a pole-balancing control program, with arrows indicating the flow from the GRN to TAGE and vice versa, highlighting the interaction between the two systems.
GE Components

Grammar \rightarrow \text{gpm} \rightarrow \text{Output Structure}

Search Engine \rightarrow \text{gpm}

Objective Function \rightarrow \text{gpm}
Alternative Search Engines

Social Programming

- Grammatical Swarm = PSO+GEMapper

++

- Grammatical Differential Evolution = DE+GEMapper
- Simulated Annealing...
GE Theses from our group...

Alternative GP Representations (Revisited)

**Evolve it!**

- Langdon’s evolution of data structures
- Spector’s “autoconstructive” evolution
- Banzhaf’s evolution of encoding
- O’Neill et al.’s evolution of grammars (use meta-grammars)
- O’Neill et al.’s evolution of mapping ($\pi$GE)
Grammer-based GP Literature

Sample of references...


Grammar-based GP Literature (continued)...

Sample of evolving representations references...


