UCD CASL

Complex & Adaptive Systems Laboratory

Genetic Programming: Syntax & Semantics

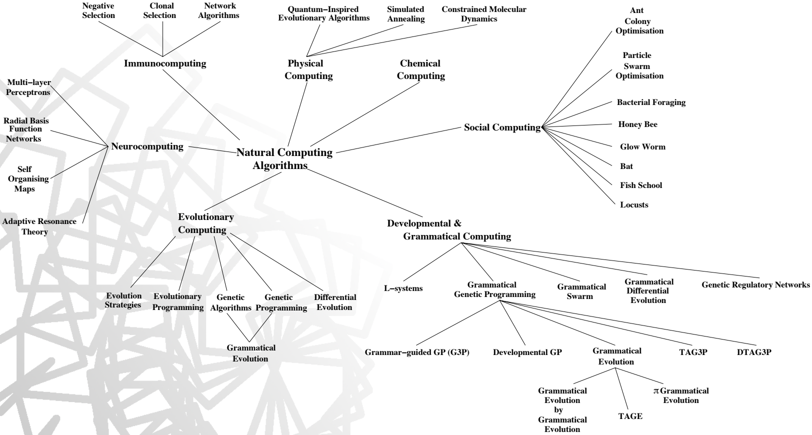
Michael O'Neill

Computational Toolbox Winterschool
Engelberg 8-12 Feb 2014

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


Negative Selection
Clonal Selection
Network Algorithms
Immunocomputing
Quantum−Inspired Evolutionary Algorithms
Simulated Annealing
Constrained Molecular Dynamics
Physical Computing
Chemical Computing
Ant Colony Optimisation
Particle Swarm Optimisation
Bacterial Foraging
Honey Bee
Glow Worm
Bat
Fish School
Locusts
Social Computing
Multi−layer Perceptrons
Radial Basis Function Networks
Self Organising Maps
Adaptive Resonance Theory
Neurocomputing
Natural Computing Algorithms
Evolutionary Computing
Evolution Strategies
Evolutionary Programming
Genetic Algorithms
Genetic Programming
Differential Evolution
Grammatical Evolution
Developmental & Grammatical Computing
L−systems
Grammatical Genetic Programming
Grammatical Swarm
Grammatical Differential Evolution
Genetic Regulatory Networks
Grammar−guided GP (G3P)
Developmental GP
Grammatical Evolution
TAG3P
DTAG3P
Grammatical Evolution by Grammatical Evolution
TAGE
π Grammatical Evolution

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

```
#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);
}
```

```
#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);
}
```

Representation

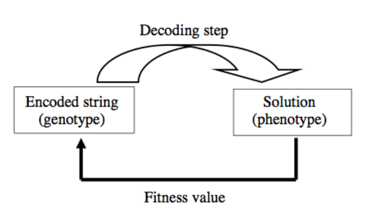
Recall GAs

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

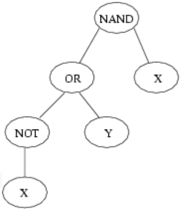
Representation

The Road to GP

- Historical Perspective on Representation
  - Friedberg (1958/1959);
  - Cramer (1985);
  - Koza (1989).
- Distinguishing features:
  - Variable-length;
  - GP operates on solution directly.



Representation

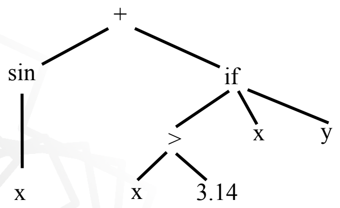


(NAND (OR (NOT X) Y) X)

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...

Representation



Different types of functions/terminals

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

Representation

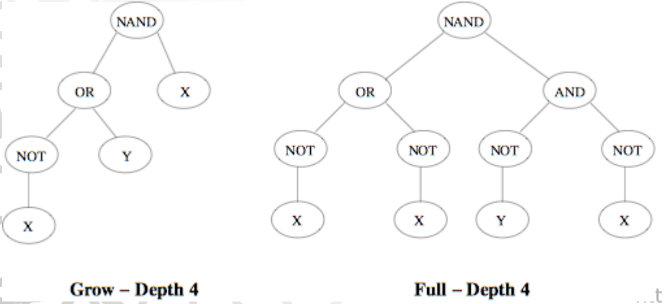
Features

- 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 )` ?!?

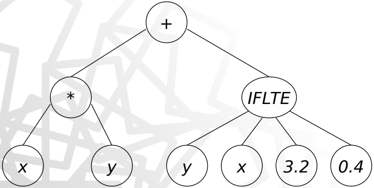
Initialisation

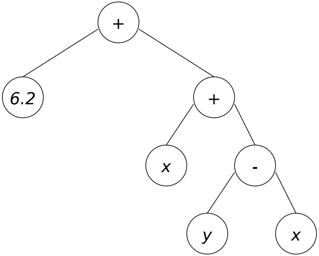
Tree Creation

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



Initialisation

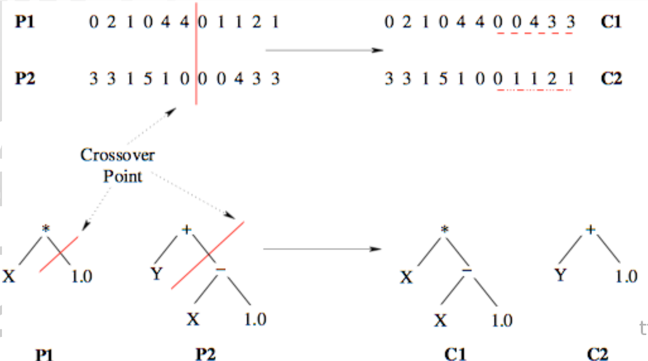




Genetic Operators

Operators

- Crossover;
- Mutation;
- Reproduction.



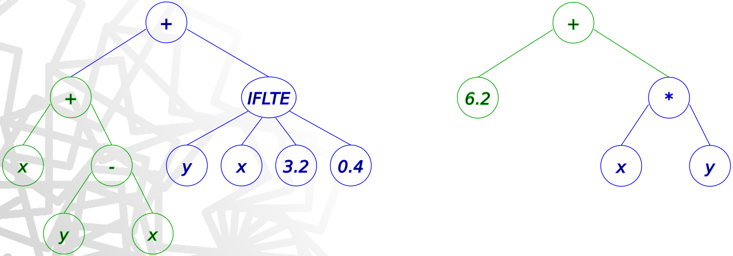
Genetic Operators

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

Crossover

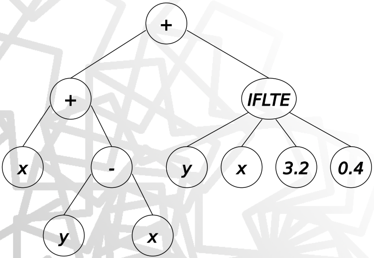


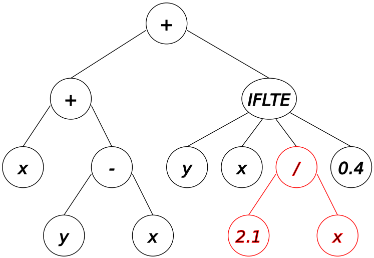
Genetic Operators

Mutation

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

Mutation





Parametrisation

$$x \longrightarrow \boxed{?} \longrightarrow y$$

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

case #	Input	Output
1	1	2
2	2	6
3	4	20
4	7	56
5	9	90

- 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

case #	Input	Output (o_i)	Prediction (p_i)	Error	Squared Error
1	1	2	1	1	2
2	2	6	4	2	4
3	4	20	16	4	16
4	7	56	49	7	49
5	9	90	81	9	81
				23	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.

GP - Example Problems

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;
- 20^{th} Century Patents;
- 21^{st} Century Patents;
- New Patented Inventions.

GP 1987-2002

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

Human-Competitive Results (non-Patent)

Transmembrane segment identification problem for proteins
Motifs for DEAD box family and manganese superoxide dismutase family of proteins
Cellular automata rule for Gacs-Kurdyumov-Levin (GKL) problem
Quantum algorithm for the Deutsch-Jozsa early promise problem
Quantum algorithm for Grovers database search problem
Quantum algorithm for the depth-two AND/OR query problem
Quantum algorithm for the depth-one OR query problem
Protocol for communicating information through a quantum gate
Quantum dense coding
Soccer-playing program that won its first two games in the 1997 Robo Cup competition
Soccer-playing program that ranked in the middle of field in 1998 Robo Cup competition
Antenna designed by NASA for use on spacecraft
Sallen-Key filter

20^{th} Century Patents

Campbell ladder topology for filters
Zobel M-derived half section and constant K filter sections
Crossover filter
Negative feedback
Cauer (elliptic) topology for filters
PID and PID-D2 controllers
Darlington emitter-follower section and voltage gain stage
Sorting network for seven items using only 16 steps
60 and 96 decibel amplifiers
Analog computational circuits
Real-time analog circuit for time-optimal robot control
Electronic thermometer
Voltage reference circuit
Philbrick circuit
NAND circuit
Simultaneous synthesis of topology, sizing, placement, and routing

21st Century Patents

Low-voltage balun circuit
Mixed analog-digital variable capacitor circuit
High-current load circuit
Voltage-current conversion circuit
Cubic function generator
Tunable integrated active filter

GP Literature

Sample of references...

- Koza J.R. (1992) Genetic Programming. MIT Press. Also books in 1994, 1999 and 2003.
- Koza J R (2010) Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines 11(3–4):251–284
- Poli R, Langdon W B, McPhee N F (2008) A Field Guide to Genetic Programming. `http://www.gp-field-guide.org.uk`
- Langdon W B, Poli R (2002) Foundations of Genetic Programming, Springer-Verlag
- Banzhaf W, Nordin P, Keller R E, Francone F D (1998) Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann
- Friedberg R. (1958). A learning machine: Part 1. IBM Journal of Research & Development 2(1):2-13
- Friedberg R. (1959). A learning machine: Part 2. IBM Journal of Research & Development pp.282-287.

Grammar-based Genetic Programming

Alternative GP Representations

Representations

- Various explored since trees;
- Graphs (PADO);
- Linear (Friedberg, Cramer, CGP and DGP);
- Grammars:
  - Tree-based (G^3P);
  - 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 = <expr>

N = {<expr>, <op>, <pre-op>, <var>}

P =

<expr> ::= (<op> <expr> <expr>)
| (<pre-op> <expr>)
| <var>

<op> ::= + | - | / | *

<pre-op> ::= sin | cos | tan | log

<var> ::= x

DGP

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.

DGP

```
<expr> ::= (<expr> <op> <expr>) | <var>
<op> ::= + | *
<var> ::= a | b
```

Developmental GP

Codon	Symbol
000	a
001	b
010	$+$
011	$*$
100	a
101	b
110	$+$
111	$*$

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

DGP

```
<expr> ::= (<expr> <op> <expr>) | <var>
<op> ::= + | *
<var> ::= a | b
```

Developmental GP

Codon	Symbol
000	a
001	b
010	$+$
011	$*$
100	a
101	b
110	$+$
111	$*$

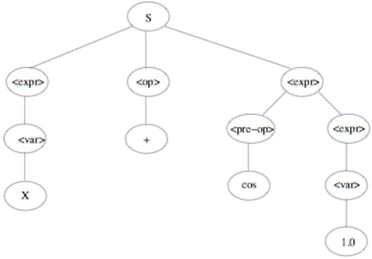
- 000 001 011 gives $ab*$;
- a is ok, b is illegal:
  - Look up <op> in grammar;
  - Nearest to b (001) is $*$ (011).
- $a * *$:
  - Second $*$ illegal;
  - Look up <var>;
  - Closest to 011 is b (001);
  - $a * b$.

G^3P

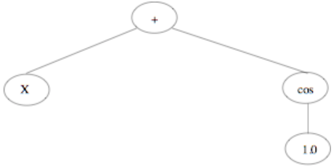
- Grammar Guided Genetic Programming;
- Use Derivation Trees:
  - Crossover: match NT symbol (no match, no XO);
  - Mutation: Replace with random derivation sequence.

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

Derivation Tree



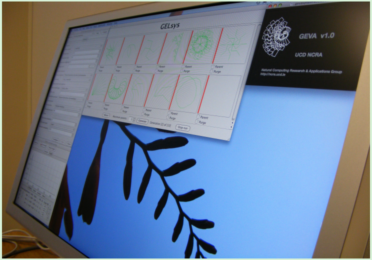
Syntax Tree



Grammatical Evolution

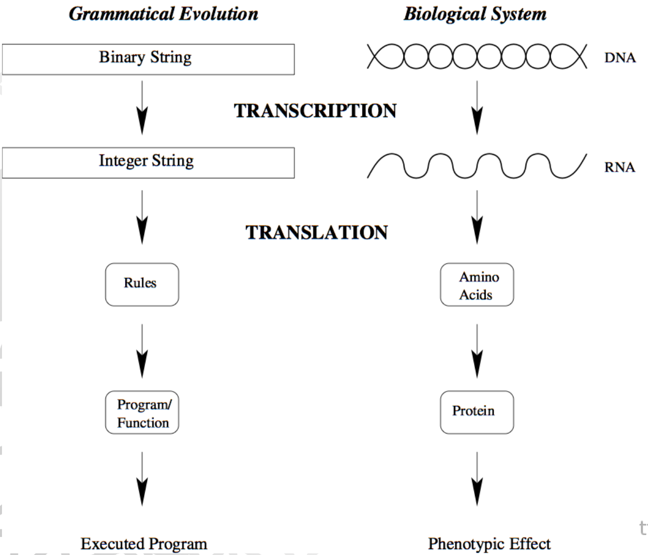
Grammatical Genetic Programming

- Chromosomes:
  - Linear;
  - Binary/Integer;
  - Variable-length.
- Genotype-Phenotype Map;
- Bio-inspired.



ONeill, Ryan. (2003). Grammatical Evolution. Kluwer Academic Press.
Brabazon, ONeill. (2006). Biologically Inspired Algorithms for Financial Modelling. Springer
Dempsey, ONeill, Brabazon. (2009). Foundations of GE in Dynamic Environments. Springer.

Genotype-Phenotype Map



Genetic Code

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

Code	Name	Code	Name
Phe	Phenylalanine	**Leu**	Leucine
Tyr	Tyrosine	**Cys**	Cysteine
Trp	Tryptophan	**Pro**	Proline
His	Histidine	**Gln**	Glutamine
Arg	Arginine	**Ile**	Isoleucine
Met	Methionine	**Thr**	Threonine
Asn	Asparagine	**Lys**	Lysine
Ser	Serine	**Val**	Valine
Ala	Alanine	**Asp**	Aspartic Acid
Glu	Glutamic Acid	**Gly**	Glycine

Genetic Code

GENETIC CODE — **PARTIAL PHENOTYPE**

Codon (A group of 3 Nucleotides) — **Amino Acid** (Protein Component)

G G C
G G A $\longrightarrow$ Glycine
G G G

GE Codon — **GE Rule**

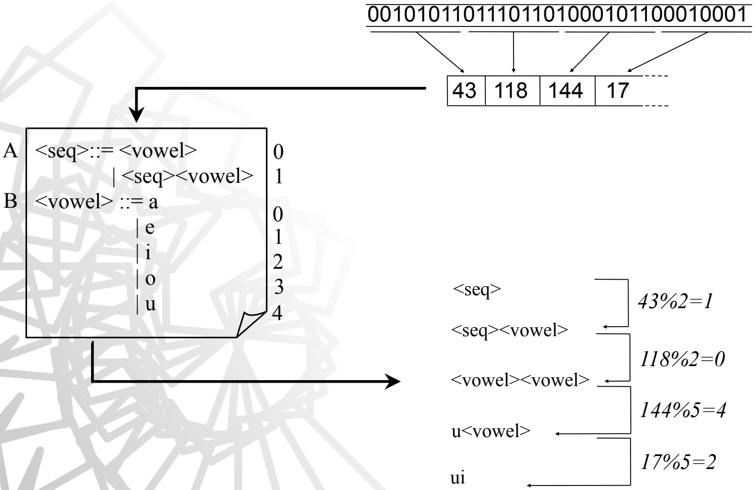
00000010
00010010 $\longrightarrow$ <line>
00100010

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

<code> :: = <line> (0)
| <code><line> (1)

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

Example Mapping



Symbolic Regression Grammar

S = <expr>

P = <expr> ::= <expr> <op> <expr>

| (<expr> <op> <expr>)

| <pre-op> (<expr>)

| <var>

<op> ::= + | - | / | *

<pre-op> ::= sin | cos | tan | log

<var> ::= x

Wrapper

```
<func> ::= <header>
<header> ::= float symbreg(float x) { <body> }
<body> ::= <declarations><code><return>
<declarations> ::= float a;
<code> ::= a = <expr>;
<return> ::= return (a);
```

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

```
<code> ::= <line>;
        | <line>; <code>
<line> ::= …
```

Example Mapping

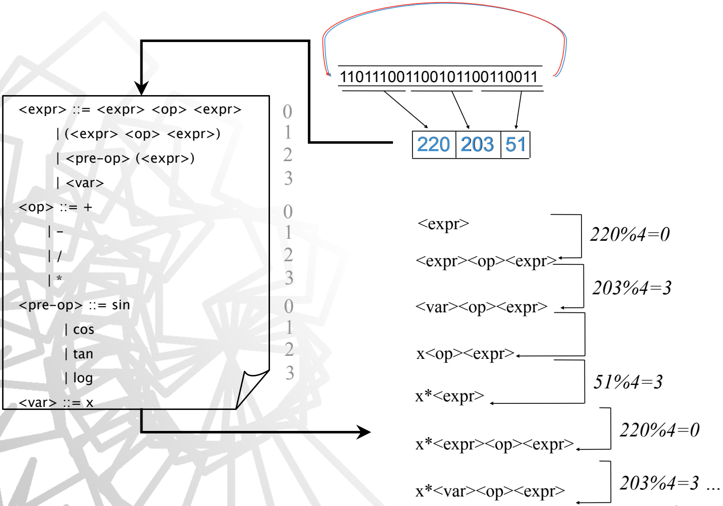
```
<expr> ::= <expr> <op> <expr>      0
         | (<expr> <op> <expr>)    1
         | <pre-op> (<expr>)       2
         | <var>                   3
<op> ::= +                         0
       | -                         1
       | /                         2
       | *                         3
<pre-op> ::= sin                   0
           | cos                   1
           | tan                   2
           | log                   3
<var> ::= x
```

110111001100101100110011011110110000001000101101

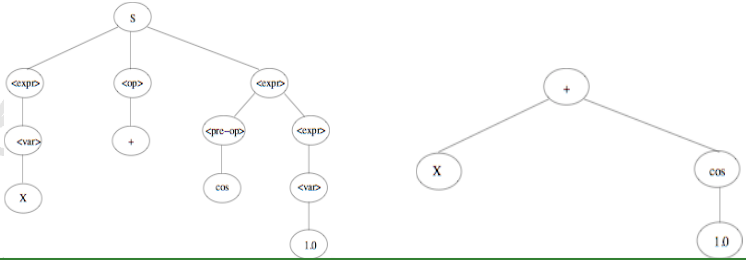
220	203	51	123	2	45

```
<expr>
                    220%4=0
<expr><op><expr>
                    203%4=3
<var><op><expr>

x<op><expr>
                    51%4=3
x*<expr>
                    123%4=3
x*<var>

x*x
```

Example Mapping



Grammatical Evolution



Genetic Operators

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

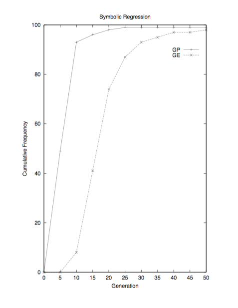
Symbolic Regression Grammar

<expr> ::= <expr> <op> <expr>
| (<expr> <op> <expr>)
| <pre-op> (<expr>)
| <var>

<op> ::= + | - | / | *

<pre-op> ::= sin | cos | exp | log

<var> ::= x | 1.0



$$F(x) = x + x^2 + x^3 + x^4$$

Santa Fe Ant Trail

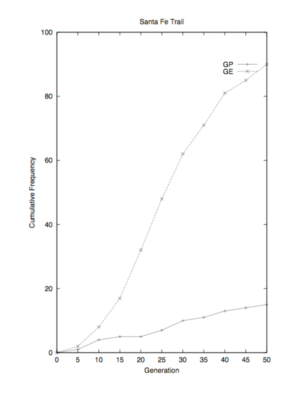
Definition

- Instructions:
  - move();
  - left();
  - right();
  - if_food_ahead();
- Fitness: pieces of food eaten within 600 time steps.

```
.111............................
...1............................
...1....................011100..
...1....................1....1..
...1....................1....1..
...1111011111.......01100....0..
............1.......0........1..
............1.......1........0..
............1.......1........0..
............1.......1........1..
............0.......1........0..
............1.......0........0..
............1.......0........1..
............1.......1........0..
............1.......1..0001110..
............0...01000..1........
............0...0......0........
............1...0......0........
............1...1......01000....
............1...1..........1....
............1...1..........0....
............1...1..........0....
............1...0......00010....
............1...0......1........
.001100111110...1...............
.1..............1...............
.1..............1...............
.1.....0111111100...............
.1.....1........................
.0.....1........................
.0111100........................
................................
```

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

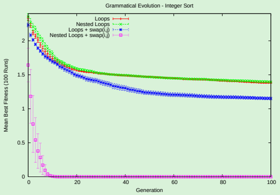
```
<code> ::= <for>
         | <for><code>

<for> ::= "\n"i=0"\n"
              for a in x :"\n\t"
                   <for_a_in_x_line>"\n\t"
                   i+=1

<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> ::= i
                     | ((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\t"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\t"
                 for b in x :"\n\t\t"
                      <for_b_in_x_line>"\n\t\t"
                      j+=1

<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
                     | ((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\t\t"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

```
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

```
<stmts> ::= <stmt>
          | <stmt>;<stmts>
<stmt> ::= if(<expr>){<stmts>;} else{<stmts>;}
          | write_x(<expr>,<expr>);
          | victim = <expr>;
<expr> ::= <term> | <term>+<term> | <term>-<term>
          | <term>*<term> | div(<term>,<term>)
          | rem(<term>,<term>)
<term> ::= CACHESIZE | <num> | <fun> | (<expr>)
<num> ::= <mant> | <mant><zeros>
<mant> ::= 0 | 1 | 2 | 3 | 4 | 5
<zeros> ::= 0 | 0<zeros>
<fun> ::= counter()
       | read_x(<expr>)
       | small_x()
       | large_x()
       | random_x()
```

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

GE1:
```
victim = counter() - CACHESIZE;
```

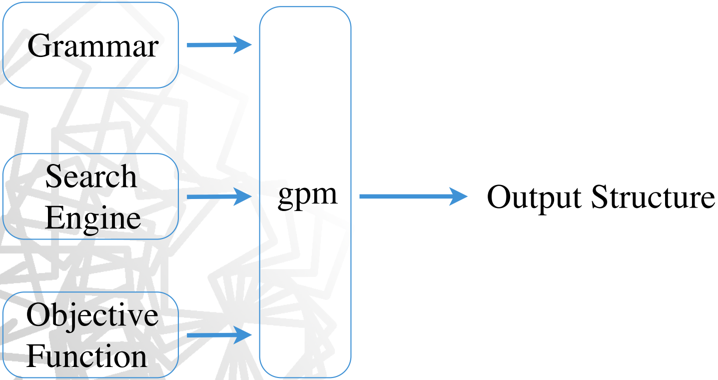
- Force use of info[]:

```
<stmts> ::= write_x(<expr>,<expr>); victim = <expr>;
```

GE2:
```
write_x(CACHESIZE + counter(), CACHESIZE + counter())
victim = CACHESIZE + counter();
```

Algorithm (CACHESIZE)	ken2.00100 (Misses)	ken2.00200 (Misses)	Average of % Gain over LRU
LRU(20)	374,596	380,041	-
LRU(200)	367,104	373,935	-
GE1(20)	300,569	318,444	17.97
GE1(200)	106,067	82,856	74.51
GE2(20)	300,569	318,444	17.97
GE2(200)	106,067	82,856	74.51

GE Components



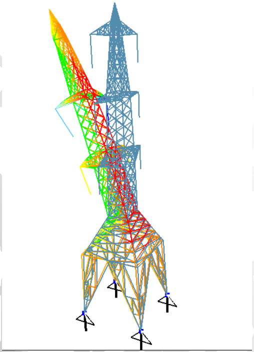
[1]

- Many important questions to address...

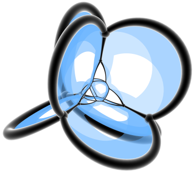
[1] Neutral Evolution

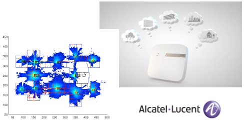
Smart & Creative Systems

UCD Natural Computing Research and Applications (NCRA) Group

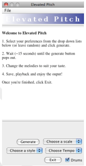


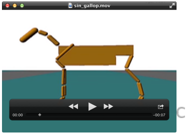
FMC²
Financial Mathematics and Computation Research Cluster



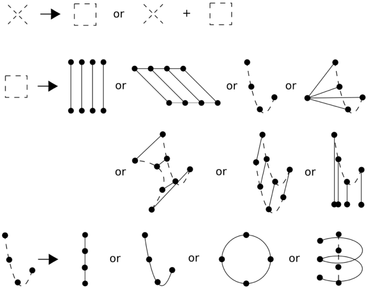


Alcatel·Lucent





Shape Grammars



- O'Neill M., McDermott J., Swafford J.M., Byrne J., Hemberg E., Shotton E., McNally C., Brabazon A., Hemberg M. (2010). Evolutionary Design using Grammatical Evolution and Shape Grammars: Designing a Shelter. International Journal of Design Engineering, 3(1):4-24;
- McDermott J., Swafford J.M., Hemberg M., Byrne J., Hemberg E., Fenton M., McNally C., Shotton E., O'Neill M. (2012). An Assessment of String-Rewriting Grammars for Evolutionary Architectural Design. Environment and Planning B, 39(4):713-731.

Tree-adjunct Grammars

```
Grammar:
<e>:= <e><o><e> | <v>
<o>:= + | -
<v>:= X | Y

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

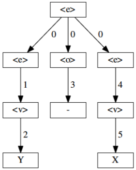
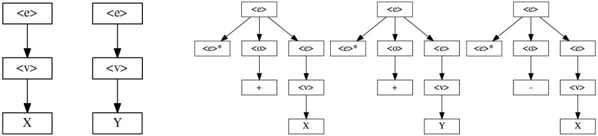


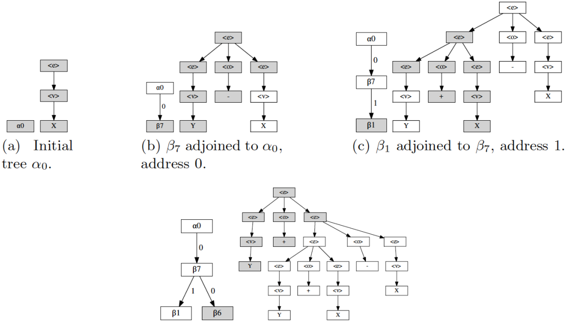
Fig. 1. Sample GE grammar, chromosome and resulting derivation tree (edge labels indicating the order of expansion). <> denotes a non-terminal symbol.



(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



(a) Initial tree α_0.

(b) β_7 adjoined to α_0, address 0.

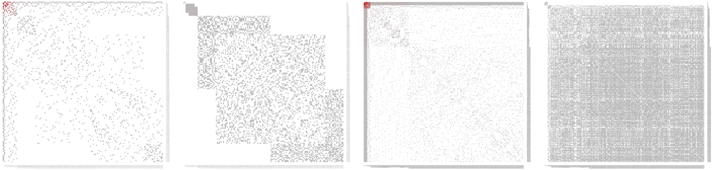
(c) β_1 adjoined to β_7, address 1.

(d) β_6 adjoined to β_7, address 0.

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

	Even 5	Santa Fe	Sym. Reg.	Six Multi.
GE	79	3	44	6
TAGE	88	12	76	63

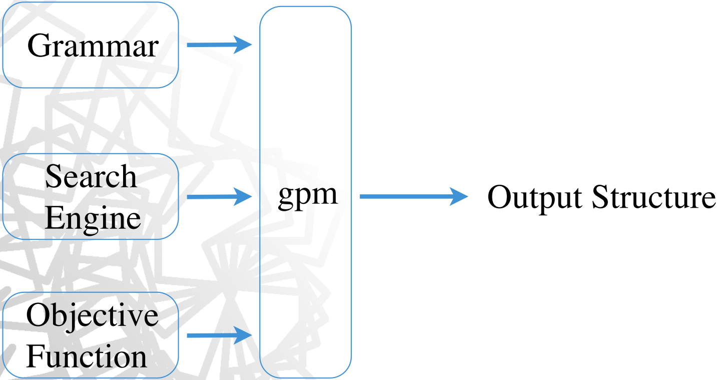
Tree-adjunct Grammars



(a) SR CFG-GE (b) SR TAGE (c) SM CFG-GE (d) SM TAGE

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

GE Components



Alternative Mapping

- πGE
- Artificial Genetic Regulatory Networks

πGE

Evolve the mapping order

```
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. 3. NT selection process in πGE

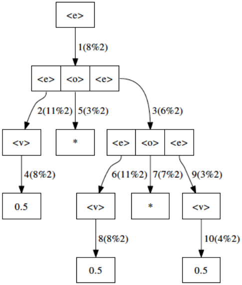
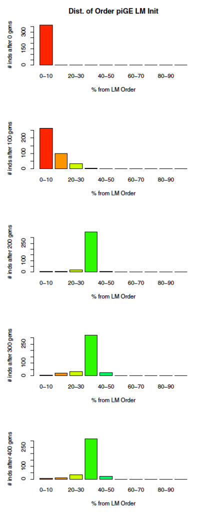
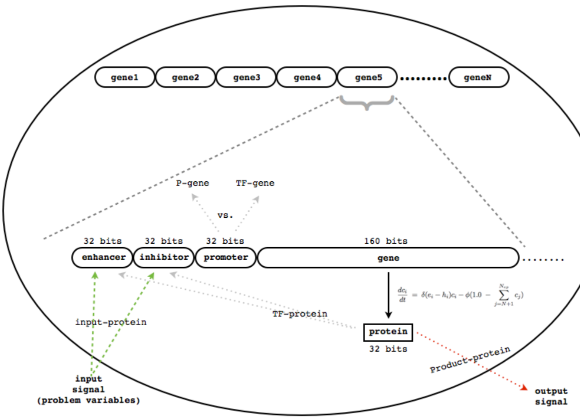


Fig. 4. Standard πGE Genotype to Phenotype Mapping

A distribution of orders



Artificial GRN's



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:

$$e_i = \frac{1}{N}\sum_{j=1}^{N} c_j \exp(\beta(u_j - u_{max})) \tag{1}$$

$$h_i = \frac{1}{N}\sum_{j=1}^{N} c_j \exp(\beta(u_j - u_{max})) \tag{2}$$

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_{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 β 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):

$$\frac{dc_i}{dt} = \delta(e_i - h_i)c_i - \phi(1.0) \tag{3}$$

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 δ is a scaling factor.

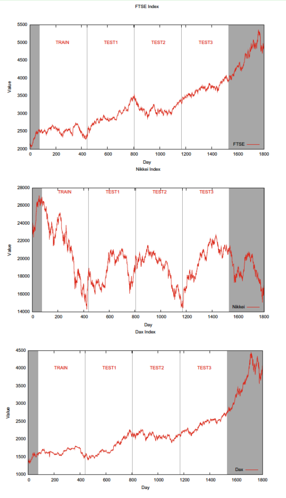
Artificial GRN's

Algorithm 21.1: Artificial Genetic Regulatory Model for GP

```
initialise population
for each generation do
    for each individual do
        execute GRN model until it reaches a steady state;
        for each input do
            set values of input gene(s);
            execute GRN model for time period t;
            read values of output gene(s);
            update fitness;
        end
    end
    select parents;
    apply genetic operators;
    replacement;
end
```

GRN's for Financial Modelling

$mAvg(10)$:	00000000000000000000000000000000
$sOsc(10)$:	00000000000000001111111111111111
$mChange(5)$:	11111111111111110000000000000000
$mChange(10)$:	11111111111111111111111111111111



GRN's for Financial Modelling

FTSE market

Period (days)	Buy & Hold	Best-of-run	Avg. daily inv.
Train (75 to 439)	-1269.28	3275.96	5939.73
Test 1 (440 to 804)	4886.9	1083.58	2191.78
Test 2 (805 to 1169)	-1089.8	541.806	3709.59
Test 3 (1170 to 1534)	1908.53	500.949	2487.67
Total	**4436.35**	**5402.295**	

Nikkei market

Period (days)	Buy & Hold	Best-of-run	Avg. daily inv.
Train (75 to 439)	-6345.5	6163.38	5128.77
Test 1 (440 to 804)	1014.79	1125.6	1457.53
Test 2 (805 to 1169)	-5263.49	2144.71	3679.45
Test 3 (1170 to 1534)	4040.59	1331.56	961.644
Total	**-6553.61**	**8514.05**	

Dax market

Period (days)	Buy & Hold	Best-of-run	Avg. daily inv.
Train (75 to 439)	-882.241	2899.86	4586.3
Test 1 (440 to 804)	4047.63	952.689	3347.95
Test 2 (805 to 1169)	-551.995	608.161	1471.23
Test 3 (1170 to 1534)	2972.24	992.868	3495.89
Total	**5585.634**	**5453.578**	

GRN's for Financial Modelling

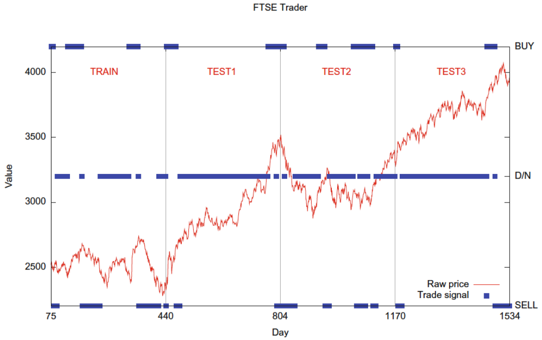
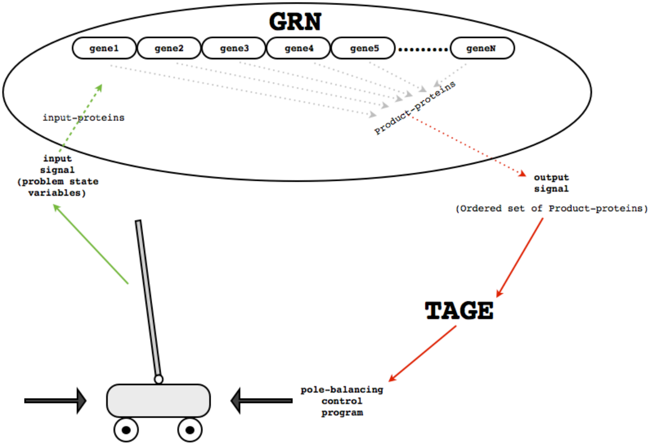
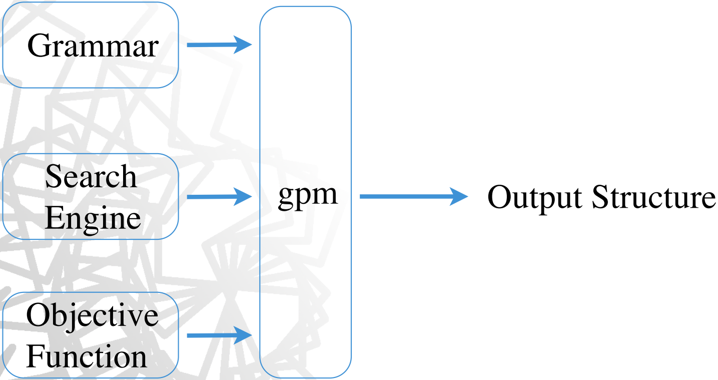


Fig. 3. Best evolved trader for the FTSE index. Blue dots indicating buy, do nothing and sell signals are plotted along with the raw market prices.

GRN's + Tree-adjunct Grammars



GE Components



Alternative Search Engines

Social Programming

- Grammatical Swarm = PSO+GEMapper

++

- Grammatical Differential Evolution = DE+GEMapper
- Simulated Annealing...

GE Theses from our group...

- Fagan D. (2014). An analysis of genotype-phenotype mapping in GE, PhD Thesis, UCD;
- Murphy E. (2014). An Exploration of Tree-Adjoining Grammars for GE, PhD Thesis, UCD;
- Swafford J.M. (2013). Analyzing the Discovery and Use of Modules in GE, PhD Thesis, UCD;
- Byrne J. (2012). Approaches to Evolutionary Architectural Design Exploration Using Grammatical Evolution, PhD Thesis, University College Dublin;
- Cui W. (2012). An Empirical Investigation of Price Impact: An Agent-based Modelling Approach, PhD Thesis, University College Dublin;
- Murphy James E. (2011). Applications of Evolutionary Computation to Quadrupedal Animal Animation, PhD Thesis, University College Dublin;
- Hemberg E. (2010). An Exploration of Grammars in Grammatical Evolution, PhD Thesis, UCD;
- Dempsey I. (2007). Grammatical Evolution in Dynamic Environments, PhD Thesis, UCD.
- Cleary R. (2005). Extending Grammatical Evolution with Attribute Grammars: An Application to Knapsack Problems, Masters Thesis;
- Amarteifio S. (2005). Interpreting a Genotype-Phenotype Map with Rich Representations in XMLGE, Masters Thesis.
- Leahy F. (2005). Grammatical Swarm, Masters Thesis.

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 (πGE)

Grammar-based GP Literature

Sample of references...

- McKay R.I., Nguyen X.H., Whigham P.A., Shan Y., O'Neill M. (2010). Grammar-based Genetic Programming - A Survey. Genetic Programming and Evolvable Machines 11(3-4):365-396
- Whigham P.A. (1996). Grammatical bias for evolutionary learning. PhD thesis. University of New South Wales, ADFA.
- Wong M.L., Leung K.S. (2000). Data Mining Using Grammar Based Genetic Programming and Applications. Kluwer Academic Publishers.
- Paterson N. (2002). Genetic programming with context-sensitive grammars. PhD thesis, Saint Andrew's University.
- O'Neill M. (2001). Evolutionary automatic programming in an arbitrary language: evolving programs with Grammatical Evolution. PhD thesis, University of Limerick.
- ONeill M., Ryan C. (2003). Grammatical Evolution. Kluwer Academic Press.
- Brabazon A., ONeill M. (2006). Biologically Inspired Algorithms for Financial Modelling. Springer
- Dempsey I., ONeill M., Brabazon A. (2009). Foundations of GE in Dynamic Environments. Springer.

Grammar-based GP Literature (continued)...

Sample of evolving representations references...

- O'Neill M., Brabazon A. (2005). mGGA: The meta-Grammar Genetic Algorithm. European Conference on Genetic Programming EuroGP 2005 pp.311-320, LNCS 3447, Lausanne.
- O'Neill M., Ryan C. (2004). Grammatical Evolution by Grammatical Evolution: The Evolution of Grammar and Genetic Code. European Conference on Genetic Programming EuroGP 2004, pp.138-149, LNCS 3003, Coimbra, Springer.
- Spector L., Robinson A. (2002). GP and autoconstructive evolution with the Push programming language. Genetic Programming & Evolvable Machines 3(1):7-40.
- Langdon W.B. (1998). Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming! Kluwer.
- Keller R.E., Banzhaf W. (1996). Genetic Programming using Genotype-Phenotype Mapping from Linear Genomes into Linear Phenotypes. Genetic Programming 1996: Proceedings of the First Annual Conference, pp.116-122, Stanford University. MIT Press.