Using Human Speech Structures to Model Reality: Grammars in Genetic Programming

Bob McKay

School of Information Technology & Elec. Eng.
University of New South Wales @
The Australian Defence Force Academy
Outline

• Introduction
  – Background: Genetic Programming
    • Typical Applications
  – Background: Grammars

• Grammar Guided Genetic Programming

• Grammatical Evolution

• Tree Adjunct Grammars and Genetic Programming

• Estimation of Distribution Algorithms, Grammars and Genetic Programming

• Developmental Evaluation and Genetic Programming
Thanks to...

- Hussein Abbass
- Daryl Essam
- Hao Hoang
- Xuan Nguyen
- Brian Ross
- Yin Shan
- Peter Whigham
- Xin Yao

- All of whom helped to form these ideas

• (but I take full responsibility)
Aim

• Genetic Programming is now 15 - 20 years old
• Formal grammars derive from the work of Chomsky and others in the 1950s
• They were first combined in 1995
• Today, grammar-based GP is an important part of GP research
• This talk aims to describe some research directions in this area
• Introduction
  – **Background: Genetic Programming**
    • Typical Applications
  – **Background: Grammars**

• Grammar Guided Genetic Programming

• Grammatical Evolution

• Tree Adjunct Grammars and Genetic Programming

• Estimation of Distribution Algorithms, Grammars and Genetic Programming

• Developmental Evaluation and Genetic Programming
Evolutionary Computation
Underlying Idea

If Darwinian evolution can create solutions to complex problems of survival in the natural World…. 

….Why not apply it to creating solutions to problems of interest to us?
Generate the initial population $P(0)$ stochastically

**REPEAT**

- Evaluate the fitness of each individual in $P(t)$
- Select parents from $P(t)$ based on their fitness
- Use stochastic variation operators to get $P(t+1)$

**UNTIL** termination conditions are satisfied
Tree Based Genetic Programming

- Original Idea:
  - Evolve populations of trees representing problem solutions
  - Cramer (1985); Schmidhuber (1987); Koza (1992)
    - Closure assumption: any function can apply to any argument
• Typical: ramped half-and-half initialisation

  – Ramped:
    • Choose a lower and upper bound for tree depth
    • Generate trees with maximum depths distributed uniformly between these bounds

  – Half and half:
    • 50% full trees
      – At depth bound, nodes chosen uniformly randomly from constant symbols
      – Elsewhere, nodes chosen randomly from the function symbols
    • 50% grow trees
      – At depth bound, nodes chosen randomly from constant symbols
      – Elsewhere, nodes chosen randomly from all symbols
GP Selection

• **Truncation selection**
  – Select the best k% of the population
    • Generally too eager

• **Fitness proportionate selection**
  – Probability of selection proportionate to fitness

• **Tournament selection**
  – Choose k individuals uniformly randomly
  – Select the best of those individuals
    • Eagerness tunable by k
      – Larger k = more eager algorithm
  – The most commonly used today
Stochastic Variation Operator: Mutation

- Randomly choose a node in the parent tree
- Delete the sub-tree below that node
- Generate a new random sub-tree
Stochastic Variation Operator: Crossover

- Randomly choose a node in each parent tree
- Exchange the sub-trees rooted at those points
What’s the Difference? (between GP and GA)

• Naïve: Tree vs linear representation

• More important: fixed complexity vs unknown complexity
• Introduction
  – Background: Genetic Programming
    • Typical Applications
  – Background: Grammars

• Grammar Guided Genetic Programming

• Grammatical Evolution

• Tree Adjunct Grammars and Genetic Programming

• Estimation of Distribution Algorithms, Grammars and Genetic Programming

• Developmental Evaluation and Genetic Programming
Electronic Design

- Koza et al: Zobel filter
Quantum Algorithms

• Barnum, Bernstein and Spector Depth One OR Query

\[ \theta = 5.96143477 \]
Control System Parameters

• Koza et al:
  – Parameter Equations for Proportional Integral Derivative (PID) Controller
Bioinformatics

• Wide variety of applications

• Well known: Motif detection for gene families
  – D-E-A-D
  – manganese superoxide dismutase
  – Koza et al 1999
Antenna Design

  - Design of wire antenna for NASA spacecraft
Chemical Dynamics Modelling

- Evolving systems of differential equations
  - Predicting discharge behaviour of a battery
  - Cao et al 2000
Ecological Modelling

Inexpensive Layers

Expensive Layer
• Introduction
  – Background: Genetic Programming
    • Typical Applications
  – Background: Grammars

• Grammar Guided Genetic Programming
• Grammatical Evolution
• Tree Adjunct Grammars and Genetic Programming
• Estimation of Distribution Algorithms, Grammars and Genetic Programming
• Developmental Evaluation and Genetic Programming
Grammars and Natural Language

• Formal Grammars developed in the 1950s to represent natural and computing languages

• Hierarchy of grammars:
  – Regular Grammars
  – Context Free Grammars (CFG)
  – Context Sensitive Grammars (CSG)

• CFGs are most widely used in GP
Context Free Grammars

- Sentences are made up from simpler sub-structures:

  - Sent → Sub Pred
  - Sub → NP
  - Pred → V PP
  - PP → P NP-
  - NP → Det N
  - V → sat | stood | …
  - Det → the | a | …
  - N → cat | mat | …

Derivation Tree
The Idea....

• Why not combine the ideas of Genetic Programming and Context Free Grammars?
  – Wong and Leung 1995
  – Whigham 1995
  – Schultz 1995
  – Gruau 1996

• Evolve populations of
  – Grammar derivation trees
    Instead of
  – Expression trees
• Introduction
  – Background: Genetic Programming
    • Typical Applications
  – Background: Grammars

• Grammar Guided Genetic Programming

• Grammatical Evolution

• Tree Adjunct Grammars and Genetic Programming

• Estimation of Distribution Algorithms, Grammars and Genetic Programming

• Developmental Evaluation and Genetic Programming
Grammar-Based Representations

• The chromosome is a derivation tree in a predefined grammar

  - $S \rightarrow B$
  - $B \rightarrow B \text{ or } B$
  - $B \rightarrow B \text{ and } B$
  - $B \rightarrow \neg B$
  - $B \rightarrow \text{if } B B B$
  - $B \rightarrow a0 \text{ | } a1$
  - $B \rightarrow d0 \text{ | } d1 \text{ | } d2 \text{ | } d3$

![Derivation Tree Diagram]
Grammar Guided Genetic Programming (GGGP)

• Problem space represented by a Context Free Grammar G
  – Individuals are derivation trees in G
  – Crossover uses sub-tree crossover as in GP
    • But the root nodes must have the same label
  – Mutation uses sub-tree mutation as in GP
    • But the generated sub-tree must be consistent with the grammar
What do Grammars offer to Genetic Programming?

• **Declarative Search Space Restriction**
  – Grammars offer a systematic, declarative way to incorporate user knowledge of restrictions on the search space

• **Incremental Model Exploration**
  – Supports an interactive style of model development, in which the user gradually refines the grammar describing the search space, until it is sufficiently small for effective GP solutions to emerge
What do Grammars offer to Genetic Programming (2)?

- Solution Models and Incremental Learning
  - The incremental refinement process may be automated
    - Grammars describing progressively smaller sub-spaces of the solution space

- Homologous Operators
  - In GGGP crossover and mutation, a subtree is replaced by another with the same syntactic function
    - And hopefully, similar semantic function
• **Introduction**
  – Background: Genetic Programming
    • Typical Applications
  – Background: Grammars

• **Grammar Guided Genetic Programming**

• **Grammatical Evolution**

• **Tree Adjunct Grammars and Genetic Programming**

• **Estimation of Distribution Algorithms, Grammars and Genetic Programming**

• **Developmental Evaluation and Genetic Programming**
Motivation of Grammatical Evolution (GE)

- Generate a linear representation for Genetic Programming search spaces
  - So that the well-developed theory from Genetic Algorithms can be applied

- But preserve the semantic relationships built into tree-based representations

- The aim is to find a well-behaved transformation from a tree representation to a linear representation
The ‘Grammatical Evolution’ Transformation

• Inorder traversal of numbered productions:

1. \( S \rightarrow B \)
2. \( B \rightarrow B \text{ or } B \)
3. \( B \rightarrow \text{not } B \)
4. \( B \rightarrow \text{if } B \text{ and } B \)
5. \( B \rightarrow a_0 \)
6. \( B \rightarrow a_1 \)
7. \( B \rightarrow d_0 \)
8. \( B \rightarrow d_1 \)
9. \( B \rightarrow d_2 \)
10. \( B \rightarrow d_3 \)
Grammatical Evolution

- Problem space represented by linear strings of integers
  - Can apply normal GA-style operators
  - Genotype-phenotype mapping uses the GE transformation
    - Using modular arithmetic to guarantee feasibility

- Possibly the most widely-used GP system in existence today
Grammatical Evolution: Advantages and Disadvantages

• **Good**
  
  – Linear Chromosome
    • Benefits from the whole range of GA theoretical research
    • Can use a wide range of GA-derived operators

• **Bad?**
  
  – The GE transformation preserves distance poorly
    • Very similar genotypes can represent completely different phenotypes

• **Finding a different (more continuous) transformation is currently an active goal of GE research**
• Introduction
  – Background: Genetic Programming
    • Typical Applications
  – Background: Grammars
• Grammar Guided Genetic Programming
• Grammatical Evolution
• Tree Adjunct Grammars and Genetic Programming
• Estimation of Distribution Algorithms, Grammars and Genetic Programming
• Developmental Evaluation and Genetic Programming
Tree Adjunct Grammars (TAGs)

- Arise from more modern efforts to represent natural language
  - Joshi et al 1975
    - Traditional Grammars do not properly represent the relationship between
      - The cat sat on the mat
      - The big black cat sat lazily on the comfortable mat which it had commandeered
  - Insertion of a meaningful sub-structure into a meaningful unit is an important component of natural language
Tree Adjunct Grammars (TAGs)

- Built on two types of elementary trees
  - $\alpha$ trees
    - Represent complete syntactic units

\[ \begin{array}{c}
\alpha_1: S \\
NP \quad VP \\
Hoai \quad V \quad NP \\
likes \quad peanut
\end{array} \]

\[ \begin{array}{c}
\alpha_2: \text{Adj} \\
small
\end{array} \]

\[ \begin{array}{c}
\alpha_3: \text{Adj} \\
red
\end{array} \]
TAGs (continued)

– $\beta$ trees
  • Represent insertable elements
  • Must have an identical non-terminal at root and at frontier
    – The ‘foot’ node
TAG Operations

- **Adjunction**
  \[ \alpha_1: \]
  \[ \beta_1: \]

- **Substitution**
  \[ \alpha_3: \]
• Derivation Tree
  • Lists the sequence of operations applied in generating a given sentence

• Derived Tree
  • The CFG tree
TAG Genetic Programming (TAG3P)

• Basic Form:

• Problem space represented by a TAG Grammar G
  – Individuals are derivation trees in G
  – Crossover uses sub-tree crossover
    • But the root nodes must have the same label
  – Mutation uses sub-tree mutation
    • But the generated sub-tree must be consistent with the grammar
TAG3P vs GGGP

• Both tree-based representations
  – What have we gained?

• GGGP trees have **fixed arity**
  – Each production determines a fixed number of children

• TAG trees have **flexible arity**
  – Any sub-tree may be deleted without affecting tree validity

• This ‘non-fixed-arity’ buys us flexibility
TAG3P Properties

- Good
  - New operators
    - Point insertion and deletion
      - Powerful local search capabilities (in contrast with most GP representations)
      - Solve Daida’s structural difficulty problem
• **Replication**
  - Effective in problems with repeated structure

• **Relocation**
  - Effective in problems requiring structural exploration
TAG3P Properties (cont)

• Good?
  – Long-distance relationships
    • TAG3P building blocks can contain long-distance relationships
      – So far, only demonstrated useful in artificial problems
      – We hope to show that they exist also in some real problems

• Bad?
  – In theory, the number of $\beta$ trees may be exponential in the number of non-terminals
    • It is possible to construct CFGs for which the corresponding TAG is very large
  – Fortunately, this does not seem to happen in practical problems we have tried
    • TAG sizes are usually comparable to the size of the corresponding CFG
• Introduction
  – Background: Genetic Programming
    • Typical Applications
  – Background: Grammars
• Grammar Guided Genetic Programming
• Grammatical Evolution
• Tree Adjunct Grammars and Genetic Programming
• Estimation of Distribution Algorithms, Grammars and Genetic Programming
• Developmental Evaluation and Genetic Programming
Estimation of Distribution Algorithms

- Estimation of Distribution Algorithms maintain a probability model from which populations are generated
  - The model is updated to reflect fit individuals
  - The hope is that it converges to a problem solution

- In the simplest (independent probabilities) version, each chromosome location contains a probability table for the possible alleles at that location
EDA Algorithm

Initialise the EDA probability model
Repeat
   Generate population from probability model
   Evaluate population fitness
   Optionally, generate a new probability model
   *Update the probabilities in the model based on population fitness
Until stopping criteria are satisfied

(*GP algorithms to date use truncation selection, and update the probability tables to increase the probability of generating the truncated population)
Estimation of Distribution Algorithms

- EDAs have been extremely successful in fixed-complexity search spaces
  - Can they extend to GP representations?
  - EDA for GP must explicitly distinguish between
    - Structure learning
    - Content learning
  - In EDA terms, explicitly distinguish
    - Probability Model
    - Probabilities
  - Two primary strands
    - Prototype Tree
    - Grammar-based
Prototype Tree EDAs

- The underlying model is a full tree of maximum arity
  - Each node holds a probability table for the content of the node
    - Original version (PIPE, Salustowicz & Schmidhuber 1997) has the node probabilities independent
    - More recent versions learn dependent probabilities
PIPE Prototype Tree

+ - * / 0 1 x #

+ - * / 0 1 x #
+ - * / 0 1 x #
+ - * / 0 1 x #
+ - * / 0 1 x #
Problems with Prototype Tree EDAs

- Prototype tree gives position-dependent probabilities
  - Cannot learn position-independent building blocks
- If the same code appears in different positions in the tree, it must be learnt separately each time
Grammar-Based EDAs

• The underlying model is a stochastic Context-Free Grammar:

  • B → B or B 0.6
  • B → B and B 0.3
  • B → not B 0.1

  – Supports position-independent building blocks

    • B → C or D 0.9
    • C → C and C 0.95
    • D → not D 0.95

  – B generates ((? and ?) or (not ?)) with high probability

    • Wherever B occurs
Learning Building Blocks

- Learning building blocks in a grammar-based EDA is (relatively) easy if the grammar already records the building block
  - Just a matter of learning the probabilities
  - Very like the learning in PIPE
Learning New Building Blocks

• But what if the right building blocks don’t occur in the original grammar?

• Learning new building blocks requires learning new, more specific, grammar models:
  – Start with
    • B → B or B
    • B → B and B 0
    • B → not B
  – Learn
    • B → C or D
    • C → C and C
    • D → not D
Learning New Building Blocks

• Use Grammar Learning methods from natural language processing
  – Grammar Learning is extraordinarily computationally intensive
  – Current methods have been developed for noise-free single-shot learning
  – There is a need to develop efficient incremental grammar-learning methods for multi-shot learning in noisy environments
Learning New Grammars

- Grammar learning can be
  - Top-down or bottom-up (or inside-out)
  - Specific to general or general to specific (or both)
  - We have experimented with
    - Inside-out, general to specific
      - PEEL system, 2003
    - Bottom-up, specific to general
      - GMPE system, 2004
    - Clearly many other possibilities are possible
      - But may require identifying large repeated sub-trees
Grammar-Based EDA-GP Properties

• **Good**
  – Very fast convergence (in number of evaluations)
  – Explicit representation of building blocks
    • Can help to understand the structure of GP problems
• **Bad?**
  – Very slow (cost of learning dominates cost of evaluation)
    • Currently only practical for problems with very high evaluation cost
    • Potential for robust, approximate learning methods to dramatically reduce cost
• Introduction
  – Background: Genetic Programming
    • Typical Applications
  – Background: Grammars
• Grammar Guided Genetic Programming
• Grammatical Evolution
• Tree Adjunct Grammars and Genetic Programming
• Estimation of Distribution Algorithms, Grammars and Genetic Programming
• Developmental Evaluation and Genetic Programming
Developmental Genetic Programming

- Standard GP genotypes are highly disorganised and poorly structured (bloat etc)
- Biological genotypes exhibit a greater degree of organisation
  - Why?
Developmental Genetic Programming

- Most biological organisms go through a developmental process
  - The adult differs from the embryo
- Many have argued that this developmental process is the source of the structure which emerges in biological systems
  - Leads to the idea of Developmental Genetic Programming
    - The genotype is a program for creating the solution
- Nevertheless, evolution of modular, hierarchical structure has not been clearly demonstrated
Evaluation during Development

• How do biological and current developmental GP systems differ?
  – Biological systems are evaluated during development
    • A human doesn’t have to be fit just in adulthood
      – We must also survive pregnancy, birth, childhood
    – Current developmental GP systems are not evaluated during development
      • The incomplete individual is an incomplete program
        – Evaluation is undefined

• Hypothesis (Wuhan, 2002)
  – Evaluation during development is necessary for modular structure to emerge
    • But how to evaluate during development?
    • Recall that every rooted sub-tree of a TAG tree is a valid TAG tree
Feasibility Study: Developmental Evaluation

- Preliminary experiment only (proof of concept)
- Based on TAG3P: uses TAG3P crossover, mutation
- Evaluation process is special
  - Multiple evaluation stages (corresponding to development)
  - Trivial developmental process
    - Reveal more of the genotype at each stage
    - But tackle increasingly difficult fitness problems
First Experiment: Developmental Evaluation

• Symbolic regression on a well-known series of increasingly difficult functions

  – \( x, x^2+x, x^3+x^2+x \ldots x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x \)

• Increase portion of individual used

  – Depth 2 for function \( x \)
  – Depth 4 for function \( x^2+x \)
  – ….
  – Depth 18 for function \( x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x \)
First Experiment: Developmental Evaluation

- Mainly relatively standard GP settings
  - Crossover 90%, mutation 10%, tournament size 3
- Budget of 229,500 function evaluations (= 9 * 51 * 500)
  - May be any of \(x, x^2+x, x^3+x^2+x \ldots\)
    \(x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x\)
  - 20 random points in region \(0 < x < 1\)
- Success: error on \(x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x < 10^{-4}\)
Control Experiment: Direct Evaluation

- Standard GP symbolic regression on single function
  - $x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x$

- Same GP settings
  - Crossover 90%, mutation 10%, tournament size 3
  - Population size 500, 459 generations
  - 229,500 function evaluations on
    - $x^9+x^8+x^7+x^6+x^5+x^4+x^3+x^2+x$
    - 20 random points in region $0 < x < 1$

- Result:
  - Success rate < 10% (3` success in 30 runs)
Experiments:
Developmental Evaluation

• Each individual evaluated 9 times
  – Depth 2 for function $x$
  – ….
  – Depth 18 for function $x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$

• Same GP settings
  – Crossover 90%, mutation 10%, tournament size 3
  – Three settings (population size vs generations):
    • Population 100, 255 generations
    • Population 500, 51 generations
    • Population 500, 201 generations (ie exceeds budget)
First Results: Developmental Evaluation

• Results:
  – Success rate 100_255: 43% (13 success in 30 runs)
  – Success rate 500_51: 40% (12 success in 30 runs)
  – Success rate 500_201: 73% (22 success in 30 runs)
Discussion

- Success rate is roughly 4 times that of standard approach (for the same number of evaluations) with no sign of premature convergence
  - Even better in computation cost
    - Evaluation of $x^2 + x$ much cheaper than $x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$
    - We also have solutions for $x$, ..., $x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x$ for free

- Certainly incremental learning
  - In final generation, every individual which has learnt polynomial of degree $k$ has also learnt polynomials of degree $1$...$k-1$
Discussion

• Have we learnt modular solutions?
  – They look nicer than the standard approach on visual inspection
  – But we need to develop metrics (and code) for modular and hierarchical structure before we can definitely say we have achieved this

• Artificial Problem
  – More complex real-World problems will almost certainly require a more sophisticated approach
Developmental Evaluation - Properties

• Good
  – Effective implementation of incremental learning
  – Very fast learning for well-structured problems

• Good?
  – Perhaps able to evolve modular, hierarchical structures

• Bad?
  – Current approach requires extension to more sophisticated developmental processes
  – Our first problem sequence is extremely well-structured
    • The relationship between successive problems is simple
    • Can it extend to less well-structured problem sequences?
Future Work

• This work is only a feasibility study
  – For PhD proposal of Mr Hoang Tuan Hao

• Developmental process is minimal
  – Future work will extend to more interesting developmental processes
    • Requires a TAG analogue to CFG-based L-systems
• Introduction
  – Background: Genetic Programming
    • Typical Applications
  – Background: Grammars
• Grammar Guided Genetic Programming
• Grammatical Evolution
• Tree Adjunct Grammars and Genetic Programming
• Estimation of Distribution Algorithms, Grammars and Genetic Programming
• Developmental Evaluation and Genetic Programming
• Summary
Summary

• Standard Grammar Guided Genetic Programming is now fairly well understood (a wide range of applications)

• Grammatical Evolution, transforming tree evolution to a Genetic Algorithm problem, has been highly successful
  – But there is great potential for improving the GE transformation

• Tree Adjunct Grammars have major advantages for GP
  – Wide range of biologically inspired operators
  – Without the distance distortion imposed by the GE transformation
  – TAGs can also compactly represent long-distance relationships in the genome
    • Can this be useful in Genetic Programming?
Summary (cont)

- Grammars have major advantages for EDA
  - Clean support for evolution of structure as well as content
  - Open field of research in incremental, noise-tolerant grammar learning methods

- Developmental evaluation supports incremental learning
  - (and hopefully, modular, hierarchical structures)
  - Requires the feasibility properties of TAG for incremental evaluation
  - Future research directions
    - Metrics for modular and hierarchical structure
    - More sophisticated developmental processes
      - Requires a TAG analogue of CFG-based L-systems