Understanding GP Evolution using Entropy & Compression

Bob McKay

Structural Complexity Laboratory

sc.snu.ac.kr

School of Computer & Engineering
Seoul National University
Outline

• Entropy and Compression: Why?

• Detour
  – Expression Simplification

• Subtree Entropy Measurement
  – Results

• Measuring Regularity with Compression
  – Results
Thanks to...

- Daryl Essam
- Hao Hoang
- Moonyoung Kang
- Naoki Mori
- Xuan Nguyen
- Jungseok Shin

• (but I take full responsibility)
Evolution
Darwinian Evolution (1850s)

Multiple populations competing for limited resources

Dynamically changing populations with births and deaths

Inheritance: children are like their parents

Variation: children are not exactly the same as their parents

Fitness: different individuals have different probabilities to survive and reproduce
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
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
Some Issues in Understanding GP
Genetic Programming: Bloat

• Very fast growth of code size

• There is a range of explanations
  – A protection mechanism against the destructive effects of crossover
  – A result of selection pressure
    • With a linear genome and some reasonable choices of crossover operators, code actually shrinks in the absence of selection pressure
  – An equilibrium seeking behaviour within the population of fit programs
  – …

• Bloat often takes the form of introns - non-functional segments of code
  – ‘Not not’  ‘If true then …’
  – ‘* 1’  ‘+ 0’
Bloat

- GP solutions tend to grow unboundedly...
  - Heavily studied
    - In isolation from Diversity and Generalisation Studies
  - Measures to combat bloat generally reduce diversity
  - Bloat is also closely related to generalisation
Bloat Example: Can’t Understand

- IF \text{stand} = \text{outside\_study} \text{ THEN} \text{absent}

- IF (\text{WITHIN} 2 \text{ EXISTS} ((( NOT ((( NOT \text{quality} \geq \text{low} ) \text{ AND } (( NOT \text{floristic} \leq \text{medium} ) \text{ AND } \text{stand} < \text{high} ))) \text{ AND } (( \text{quality} = \text{medium} \text{ AND } ( \text{quality} > \text{outside\_study} \text{ AND } \text{floristic} < \text{medium} ))) \text{ OR } ((( NOT \text{floristic} \leq \text{medium} ) \text{ AND } \text{stand} < \text{high} ) \text{ AND } ( \text{slope} = \text{moderate} \text{ OR } \text{slope} \geq \text{moderate} ))) \text{ AND } (((\text{slope} > \text{steep} \text{ OR } \text{stream} > \text{no\_stream\_corridor} ) \text{ AND } (( NOT \text{stand} \leq \text{low} ) \text{ AND } \text{stream} > \text{no\_stream\_corridor} ))) \text{ AND } ( NOT ( NOT ( NOT ( \text{stream} > \text{no\_stream\_corridor} \text{ OR } \text{stand} = \text{low} ))) \text{ OR } ( \text{floristic} = \text{outside\_study} \text{ OR } \text{quality} \geq \text{low} ) \text{ AND } ( \text{floristic} < \text{high} \text{ OR } \text{stand} < \text{high} ))) \text{ OR } (\text{WITHIN} 2 \text{ EXISTS } \text{stand} = \text{regeneration} \text{ OR } ( \text{stream} = \text{no\_stream\_corridor} \text{ OR } \text{stand} = \text{low} ))) \text{ THEN} \text{rare}

- IF ( NOT (\text{WITHIN} 1 \text{ EXISTS} ((( \text{quality} < \text{medium} \text{ OR } \text{stand} = \text{regeneration} ) \text{ AND } (( NOT \text{floristic} < \text{outside\_study} ) \text{ OR } \text{stand} < \text{rock} ))) \text{ AND } ( NOT \text{WITHIN} 2 \text{ EXISTS } ((( \text{NOT} \text{dev} > \text{road\_corridor} ) \text{ AND } (( \text{slope} < \text{moderate} \text{ AND } ( \text{quality} \leq \text{outside\_study} \text{ AND } \text{floristic} < \text{medium} ))) \text{ OR } (( \text{slope} \leq \text{steep} \text{ AND } \text{stand} < \text{low} ) \text{ AND } ( \text{slope} > \text{moderate} \text{ OR } \text{floristic} \leq \text{medium} ))) \text{ )) ) \text{ THEN} \text{common}

- IF ((( \text{stand} \geq \text{regeneration} \text{ AND } \text{stand} \geq \text{outside\_study} ) \text{ AND } ( NOT ( \text{stream} \geq \text{no\_stream\_corridor} \text{ AND } \text{floristic} \leq \text{medium} ))) \text{ OR } ((( \text{floristic} = \text{outside\_study} \text{ OR } \text{slope} = \text{steep} ) \text{ AND } ((( \text{slope} \leq \text{steep} \text{ AND } \text{stand} < \text{low} ) \text{ AND } ( \text{slope} > \text{moderate} \text{ OR } \text{floristic} \leq \text{medium} ))) \text{ OR } \text{floristic} \geq \text{low} ))) \text{ THEN} \text{abundant}

- ELSE \text{DEFAULT} \text{ abundant}
Example with Bloat Removed

- IF stand = outside_study  THEN absent
- IF (WITHIN 2 EXISTS
  - ((quality ≥ low AND floristic = high AND stand < high AND slope > low)
  - OR ( quality = medium AND floristic = low))
  - AND ((stand = medium AND stream = stream_corridor AND floristic = high)
  - OR (WITHIN 2 EXISTS
  - stand = regeneration OR stream = no_stream_corridor OR stand = low)))
  - THEN rare
- IF (NOT WITHIN 1 EXISTS (quality = low OR stand = regeneration )
- OR (WITHIN 2 EXISTS(dev ~= pine_plantation AND stand < low
  - AND ( slope =steep OR floristic ≤ medium))))))
- THEN common
- ELSE DEFAULT abundant

Issues
Can bloat be controlled?

• Many attempts to control bloat by eliminating introns

• It is possible to eliminate introns in some simple domains (eg Boolean)

• Generally not possible in more complex domains
  – In continuous domains, evolution just replaces exact introns with approximate introns
    • code which is very nearly ineffective
  – Or effective code is placed into non-operational code segments
  – In general, it is un-computable whether code is ineffective

– Eliminating bloat may (generally does) conflict with the goal of preserving diversity
Training Accuracy vs Generalisation

- There is a well-known trade-off between model complexity (and accuracy) and generalisation to new data
  - More complex (and accurate) models generalise less well

- Little studied in GP
- Bloat puts pressure on complexity of effective code
  - because it uses up code space
- Gives some parsimony pressure
  - But accidental and uncontrolled
Size Limits and Introns

• with a bounded genome size
  – Introns occupy code space
    • Without adding to information to the genome
  – Hence introns reduce the degrees of freedom of the genotype
  – Provide an indirect parsimony pressure
  – Improve generalisation
    • An ad-hoc but effective, mechanism
  – But the degree of parsimony pressure is accidental and uncontrolled
Parsimony Pressure and Genetic Programming

• Wide range of methods attempted to impose parsimony pressure on the genome.... But...
  – When fitness is low (early evolution), parsimony pressure dominates
  – Results in convergence to population of very small individuals - loss of genetic diversity

• Requires changing parsimony pressure
  – Difficult to tune parameters for parsimony pressure
Diversity

- GP systems may converge before they find good solutions
  - Most study derived from Genetic Algorithms
  - But Genetic Algorithms don’t have bloat interactions…
  - GP algorithms may be converged
    - Same effective solutions
  - But still diverse in their tree forms
Possible Solution: Parsimony and Diversity

• Impose parsimony pressure continuously

• Directly oppose convergence pressure with a diversity-promoting mechanism
  – Fitness sharing
    • Difficult to combine with parsimony pressure
      – Should the parsimony fitness be shared? How?
  – Anticorrelation penalties
    • Difficult to find anticorrelation penalties with compatible dimensionality with parsimony
  – Information based accuracy, parsimony and anticorrelation measures

• Disappointing results so far
Modularity and Hierarchy in GP

- Modularity rarely arises naturally in GP systems
  - It is usually ‘built in’
    - Automatically defined functions (ADF)
    - Automatically defined macros (ADM)
- Regular structure virtually never emerges in GP genotypes
Regularity and Scalability

• **GP Does not find** regularly structured solutions:

  • IF stand = outside_study THEN absent

  • IF (WITHIN 2 EXISTS (( NOT (( NOT quality ≥ low ) AND (( NOT floristic ≤ medium ) AND stand < high )))) AND (( quality = medium AND ( quality > outside_study AND floristic < medium )) OR ((( NOT floristic ≤ medium ) AND stand < high ) AND ( slope = moderate OR slope ≥ moderate )))) AND ((( slope > steep OR stream > no_stream_corridor ) AND (( NOT stand ≤ low ) AND stream > no_stream_corridor )) AND ( NOT ( NOT ( NOT ( stream ≥ no_stream_corridor AND floristic ≤ medium ))))) AND ((( stream = no_stream_corridor OR stand = low ) AND ( NOT slope ≤ moderate )) OR (( floristic = outside_study OR quality ≥ low ) AND ( floristic < high OR stand < high ))) OR (WITHIN 2 EXISTS stand = regeneration OR ( stream = no_stream_corridor OR stand = low ))) THEN rare

  • IF ( NOT (WITHIN 1 EXISTS (( quality < medium OR stand = regeneration ) AND (( NOT floristic < outside_study ) OR stand < rock )) AND ( NOT WITHIN 2 EXISTS (( NOT dev > road_corridor ) AND (( slope < moderate AND ( quality ≤ outside_study AND floristic < medium ))OR (( slope ≤ steep AND stand < low ) AND ( slope > moderate OR floristic ≤ medium )))))) THEN common

  • IF ((( stand ≥ regeneration AND stand ≥ outside_study ) AND ( NOT ( stream ≥ no_stream_corridor AND floristic ≤ medium )))OR (( floristic = outside_study OR slope = steep ) AND ((( slope ≤ steep AND stand < low ) AND ( slope > moderate OR floristic ≤ medium ))) OR ( floristic ≥ low ))) THEN abundant

  • ELSE DEFAULT abundant
Regularity and Scalability

• **Simplified version:**
  - IF stand = outside_study THEN absent
  - IF (WITHIN 2 EXISTS
    - ((quality ≥ low AND floristic = high AND stand < high AND slope > low)
    - OR ( quality = medium AND floristic = low))
    - AND ((stand = medium AND stream = stream_corridor AND floristic = high)
    - OR (WITHIN 2 EXISTS
      - stand = regeneration OR stream = no_stream_corridor OR stand = low)))
  - THEN rare
  - IF (NOT WITHIN 1 EXISTS (quality = low OR stand = regeneration )
  - OR (WITHIN 2 EXISTS(dev ~= pine_plantation AND stand < low
    - AND ( slope =steep OR floristic ≤ medium)))))
  - THEN common
  - ELSE DEFAULT abundant

• **This excess complexity makes it difficult for GP to learn incrementally tougher problems**
We Need to Understand:

- Individual complexity
- Diversity
- Generalisation
- Regularity
What’s Wrong with Previous Analyses?

- Study separately
  - Individual complexity
    - Usually measured as size, depth
  - Diversity
    - Variety of different metrics
  - Generalisation
    - Little studied (Kushchu - late 1990s)
  - Regularity
    - Little studied
      - Hornby 2006: studied modularity in ADFs and similar systems
        » Metrics require explicit modularity, not useful for studying emergence
Our Aims

• Use related methods to study
  – Individual complexity
  – Population complexity
  – Generalisation and error
  – Solution regularity

• Study both
  – Complexity of raw individuals
  – Complexity of effective code within individuals

• Intuition:
  – These aspects of GP interact
  – So we can’t understand them in isolation
Preliminary: Expression Simplification
Redundancy in GP Arithmetic Expressions

- Neutral part: $0 \cdot f(x)$
- Redundant part: $1 \cdot f(x)$

• Aim
  - To simplify expressions
Previous Work on Redundancy

- Algebraic simplification
  - Rules such as $1*f(x) \rightarrow f(x)$

- Hard to define universal rules
  - $(f(x)-1)+(1-f(x))$
  - Functions may only be redundant given data instances
    - $\sin nx$ when $n$ instances evenly spaced in $-\pi \ldots \pi$
  - Equivalences may be inexact
    - $\pi \approx 4 \sin(\sin(2))$
Equivalent Decision Simplification

• Identify a suitable set of ‘simple’ trees

• (Recursively) check every subtree for equivalence (on fitness instances) to a ‘simple’ subtree, and replace
  – E.g. if constants are in ‘simple’ set and given appropriate instances
    • Replace \( \sin nx \) by 0

• Advantages
  – We don’t need to pre-define equivalences
  – We can handle equivalences only on the sample points
  – We can handle approximate equivalences
Sample Problem

- **Symbolic Regression**
  - 1000 runs, 500 population, 200 generations
  - Target function $\cos 2x$, range $-\pi \ldots \pi$, randomly selected target instances
  - Function set $+ - * / \sin$

- **Analysis divided into three sets of 100 runs**
  - Runs which find a solution within 20 generations
    - Highly successful
  - Runs which find a solution between 50-69 generations
    - Moderately successful
  - Runs which find no solutions
    - Unsuccessful
Equivalent Decision Effectiveness

- Typical GP population from symbolic regression
  - Solutions found by the ‘high success’ runs
  - 102,534 different solution genotypes reduced to three solutions of size 12
  - Average prior size 138.2
    - So over 90% of individuals was redundant code
  - Evidence (see later) we have found effectively all simplifications

- Comparison with typical algebraic simplification
  - Average size 15.1
  - Only 39% reduce to size 12
  - 57 genotypes retain over 100 nodes
  - Worst case 214 nodes
  - Seriously affects analyses based on algebraic simplification
Entropy Measurement
Entropy

Complicated Solutions

Loss of Diversity

Poor Accuracy

Information Theory to Understand Genetic Programming
Entropy

- Given a set of events $x_1 \ldots x_n$ with probabilities $p(x_i)$

- Entropy is $\sum_{i=1,n} - p(x_i) \log p(x_i)$

- The minimum number of bits required to transmit a message describing the probability distribution

- Entropy provides a means to measure
  - Population complexity
  - Individual complexity
  - Model accuracy
  - Between-runs complexity

- Problem:
  - Not clear how to measure entropy of GP trees
  - Approximation required
Entropy Estimates

• Entropy of single nodes
  – Effectively, entropy of node content
  – Ignores structure

• Entropy of subtrees
  – Size n=2, 3, 4,…
  – Large n gives too many possible events
  – Most bins empty, estimate of entropy inaccurate
  – 4 is largest feasible size
Subtree templates

- Initial work estimated entropies of different templates
Template Results

- Subtree Population entropy by generation
- Higher order templates generally similar
- Rest of work uses template 3a
What we could analyse before...

- Individual size
  - Left: Original
  - Right: Simplified
What we could analyse before...

- Phenotype Entropy
New Analyses: Simplified Entropy

- Left: Population Entropy (original trees)
- Right: Population Entropy (simplified)
  - Note similarity to phenotype entropy
New Analyses: Subtree Entropy

- **Original**
- **Simplified**

- **Individual**

- **Population**
Subtree Entropy: What did we learn New?

- **Individual entropy:**
  - Very different courses of original and simplified entropies
  - Much lower simplified entropy

- **Population entropy**
  - Far lower simplified entropy

- **Comparison**
  - Ability to numerically compare individual and population entropies
  - Potential for theoretical analysis and explanation
Entropy: future work

- Current work relates population entropy (diversity) and individual entropy (complexity, bloat)
- Future work aims to explore inclusion of entropy-based accuracy measurements in learning problems
Compression and Regularity
Homeobox Genes

• Control the segmentation of animals
  – (even in less apparently segmented animals such as ourselves)

• Manipulating homeobox gene control permits the growth of legs on heads, or eyes on body segments
Homeobox Genes as Subprograms

• Homeobox genes act as subroutines
  – Molecular gradients take the role of parameters, controlling detailed structure of the segment

• Homeobox genes may activate further homeobox genes, permitting a hierarchical cascade of organisation
Underlying issue

• Natural systems generate genotypes far more regular than artificial
  – In fact, natural evolution proceeds largely by copying genes and then varying them

• We believe this regularity is important for the scalability of natural evolution
  – (compared with poor scalability of GP)

• We’ve generated a couple of GP systems aimed at evolving more regular solutions
  – Perform well
  – But we’d like to argue the good performance/scalability result from the regularity bias
  – Need to at least show that the solutions are more regular

• How to measure?
• The more regular the structure, the more the item can be compressed

• The more the item can be compressed, the more regular the structure
Compression to Measure Regularity

• Use compressibility to measure complexity
  – Compress the data
    • Individual trees
    • Populations
    • Populations across runs
    • …
  – Measure the compression ratio
  – Use the compression ratio to estimate regularity
    • The more compressible, the more regular

• Of course, we need lossless (reversible) compression algorithms
Issues (1)

- Standard compression algorithms are designed for string compression
  - If we use a string representation of the tree, the compression are biased toward particular regularities
    - Eg preorder representation biases regularities in left children

- Solution: use tree compression algorithms
  - Fortunately, XML compression is currently a hot issue
  - We use XMLPPM, an excellent tree compression algorithm
  - Tree version of Predict-by-Partial-Match
  - Statistical compressor
    - Slower but substantially better compressors than dictionary based
      - LZ, gif
Issues (2)

• Originally, we just directly used compression ratio

• Problem: it is more difficult to compress small trees
  – Led to bias in compression
    • Algorithms with smaller trees give smaller compression ratios
Fixing Issue 2

• For XMLPPM
  – Random ‘grow’ trees are minimally compressible
  – Linear trees are maximally compressible
  – Measure maximal and minimal compression at a given size
  – For a given tree, population or whatever
    • Report the ratio between this tree, linear and random trees of the same size
Sample Problem

• Symbolic regression, $1+x+x^2+x^3+\ldots+x^9$
  - Find a function fitting 20 points
  - Expected to favour regular, structured solutions

• Algorithms
  - GP, TAG-GP, TAG plus development, TAG plus evaluation during development
    • Expected to promote genotype regularity
• Background only, not the important point for this talk
Regularity of Individuals

- Effectively, measures how self-similar individuals are

- Unsimplified

- Simplified
Regularity of Populations

- Measures how self-similar populations are
  - A way of measuring building block diversity

- Un simplified

- Simplified
Regularity between Runs

- Measures how much different runs discover the same building blocks

- Ununsimplified

- Simplified
**Further Work**

- Modify XMLPPM
  - Generate the compression model from one population
    - E.g. generation
  - Compress the population from a different population
  - Permits study of the inter-generational preservation of building blocks
  - How much do the building blocks in generation $k$ still exist in generation $k+1$
Conclusions
Conclusions

• Equivalent Decision Simplification allows us to study the behaviour of effective code

• Subtree entropy allows us to understand the interaction of individual and population complexity (bloat and diversity)
  – Long-term belief: understanding GP systems requires an understanding of the information flows between individual complexity, population complexity and data

• Compression allows us to understand the behaviour of building blocks within and between populations, and even between run
  – Maybe even preservation of building blocks
The End

谢谢

감사합니다