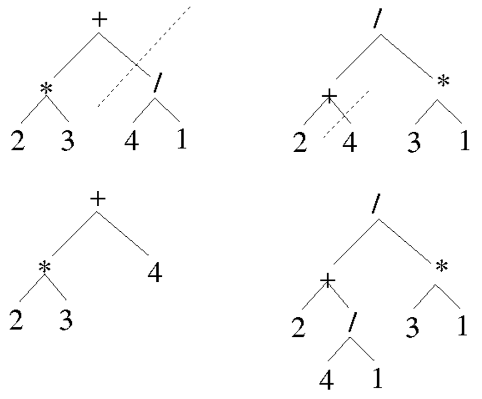
Gecco 2007 Grammatical Evolution Tutorial

Conor Ryan

Biocomputing and Developmental Systems Group
Department of Computer Science and Information Systems
University of Limerick

Issues with GP

- Function/terminal set must have "closure"



- Single types only
- Trees grow, or "bloat"

Biological Phenomena

- No simple one to one mapping
  - Genes produce proteins
  - Proteins combine to create phenotype
- Linear strings
  - Genomes are always held on strings
- Unconstrained search
  - Repair not performed

Grammatical Evolution

- Grammatical Evolution (GE)
  - GA to evolve programs
  - Morphogenetic Effect:
    - Genotype mapped to phenotype
  - Phenotype is a compilable program
- Genome governs mapping of a BNF/attribute grammar definition to the program

Grammatical Evolution

- Here genome (a binary string) is mapped to compilable C code
- Can potentially evolve programs in any language, with arbitrary complexity
- Any structure than be specified with a grammar, e.g. graphs, neural networks, etc.

Language Definition

- Backus Naur Form (BNF)
  - Notation for expressing a languages grammar as Production Rules
- BNF Grammar consists of the tuple $<$ T,N,P,S $>$ where
  - T is Terminals set
  - N is Non-Terminals set
  - P is Production Rules set
  - S is Start Symbol (a member of N)
- BNF Example

$$T = \{Sin, Cos, Tan, Log, +, -, /, *, X, (,)\}$$

$$S =< expr >$$

BNF Definition

- $$N = \{expr, op, pre\_op\}$$
- And P can be represented as:

```
(1) <expr> ::= <expr> <op> <expr> (A)
             | ( <expr> <op> <expr> ) (B)
             | <pre-op> ( <expr> ) (C)
             | <var> (D)

(2) <op> ::= + (A)
           | - (B)
           | / (C)
           | * (D)
```

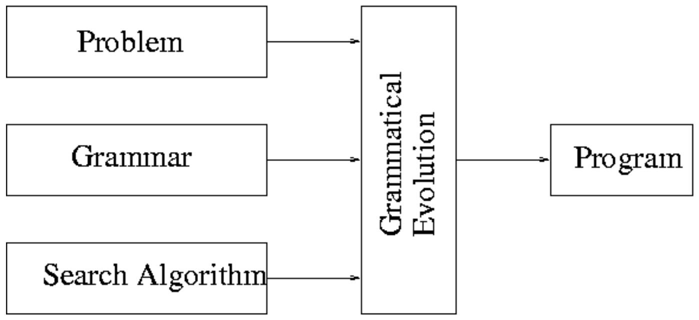
BNF Definition

```
(3) <pre-op> ::= Sin (A)
               | Cos (B)
               | Tan (C)

(4) <var> ::= X (A)
```

- A Genetic Algorithm is used to control choice of production rule

Architecture



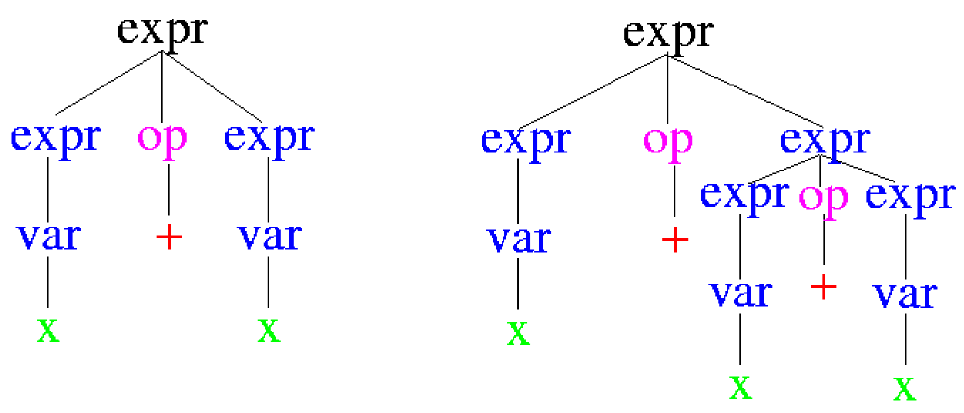
Related GP Systems

Name	Genome	Representation
Koza	Tree	Direct
Banzhaf et al	Linear	Direct
Gruau	Tree	Graph Grammar
Whigham	Tree	Derivation Tree
Wong & Leung	Tree	Logic Grammars
Paterson	Linear	Grammar

- Repair mechanisms..

- Koza - none needed
- Banzhaf - required for syntactically legal individuals
- Gruau - none needed
- Whigham - all crossovers subject to repair
- Wong & Leung - all crossovers subject to repair
- Paterson - under/overspecification.

Repair

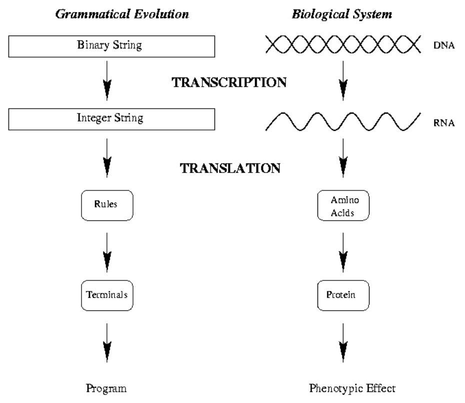


Grammatical Evolution

- In contrast GE uses
  - BNF - Paterson/Whigham/Wong etc.
  - Variable Length Linear Chromosomes - Koza/Gruau/Banzhaf
  - Genome encodes pseudo-random numbers
  - Degenerate Genetic Code
    - Several genes map to same phenotype
  - Wrap individuals
- Use 8 bit codons
  - Each codon represents at least one Production Rule
  - Gene contains many codons
- Pseudo-random numbers determine what production rule will be used

Grammatical Evolution

- Expression of a *Codon* results in an *Amino Acid* (choice in the derivation sequence)
  - *Amino acids* can combine to form a functional protein (i.e. Terminals such as $+$, *X* or *Sin*, can combine)



Example Individual

- To complete BNF definition for a function written in a subset of C we include.....

```
<func> ::= <header>
<header> ::= float symb(float X)  <body>
<body> ::= <declarations><code><return>
<declarations> ::= float a;
<code> ::= a = <expr>;
<return> ::= return (a);
```

- Note implementation details.....
  - Function is limited to a single line of code
    - If required can get GE to generate multi-line functions.....modify

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

Example Individual

- In this subset of C all individuals of the form

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

- Only $<$ *expr* $>$ will be evolved
- Each non-terminal is mapped to a terminal before any others undergo a mapping process

Example Individual

- Given the individual

220	203	51	123	2	45

....what will happen?

- <expr> has 4 production rules to choose from

```
(1) <expr> ::= <expr> <op> <expr> (A)
             | ( <expr> <op> <expr> ) (B)
             | <pre-op> ( <expr> ) (C)
             | <var> (D)
```

  - Taking first codon 220 we get 220 *MOD* $4 = 0$
  - Gives <expr>< *op* >< *expr* >
- Next choice for the first <expr>
  - Taking next codon 203 we get 203 *MOD* $4 = 3$
  - Gives <underline var>< *op* >< *expr* >

Example Individual

- <var> involves no choice
  - Mapped to X...only one production
  - Now have *X* <op>< *expr* >

220	203	51	123	2	45

- Read next codon to choose <op>
  - Next is third codon , value 51, so get 51 *MOD* $4 = 3$
  - Now have $X*$ <expr>
- Next choice for <expr>
  - Next codon is 123 so get 123 *MOD* $4 = 3$
  - Now have $X*$ <var>
- Again <var> involves no choice
  - Finally we get $X * X$
- The extra codons at end of genome are simply ignored in mapping the genotype to phenotype

Example Mapping Overview

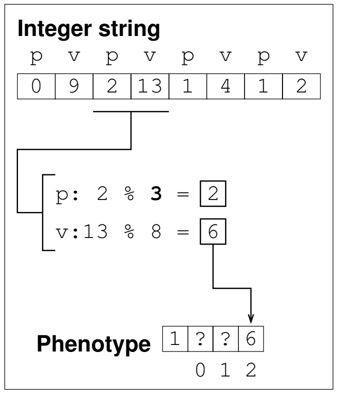
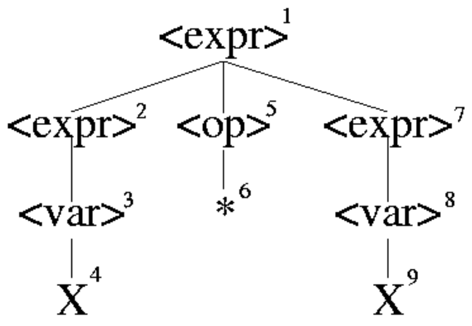


Figure: Example Mapping Outline

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

Derivation Tree Structure

1	2	5	7		
220	203	51	123	2	45



- Not all nodes require a choice!

Codons are polymorphic

- When mapping $< expr >$, we calculate

$$220 \mod 4$$

- However, if we were mapping $< pre-op >$ with 220, we would calculate

$$220 \mod 3$$

because there are just three choices

- Meaning of a codon depends on its *context*

Mapping Process

- No simple one to one mapping in GE
- Mapping Process to generate programs
  - Separate Search and Solution Spaces
  - Ensure validity of individuals
  - Remove language dependency
  - Maintain diversity

Genetic Code Degeneracy

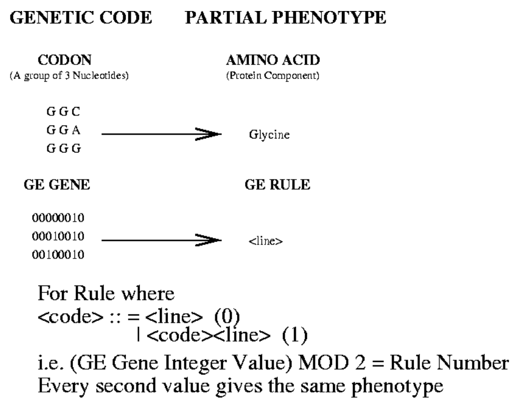


Figure: The Degenerate Genetic Code

Genetic Code Degeneracy

- Neutral Mutations
  - Mutations having no effect on Phenotype Fitness
- Help preserve individual validity
- Gradual accumulation of mutations without harming functionality
  - Revisit later

Initialisation

- Individuals are strings of random numbers
  - No guarantee that they will terminate
  - Individuals can be *very* short.

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

- Production

  `<expr>-><var>`

  always leads to termination
- `<expr>`
  is the start symbol
  - On average, a quarter of all individuals are just one point

Sensible Initialisation

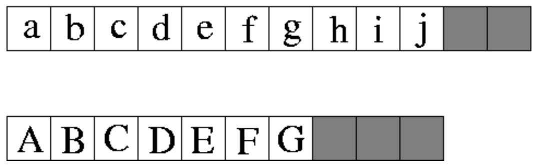
- Generate a spread of individual sizes.
  - Based on *Ramped Half and Half* initialisation in GP
    - For all tree depths from 2 to maximum size
    - Generate an equal number of trees of that size
    - Use *full* for 50%
    - Use *grow* for 50%
- Similar in GE, but generate *derivation trees* of equivalent size

Sensible Initialisation - 2

- Record which number choice was made for each step
- Perform an "unmod" on list of choices
  - Produce a number between 0 and 255 that produces the original number when moded by the number of choices for that productionrule
- Ensures that *all* individuals are valid
- Reduces the number of clones (easier to detect)
- Eliminates single point individuals (if desired)

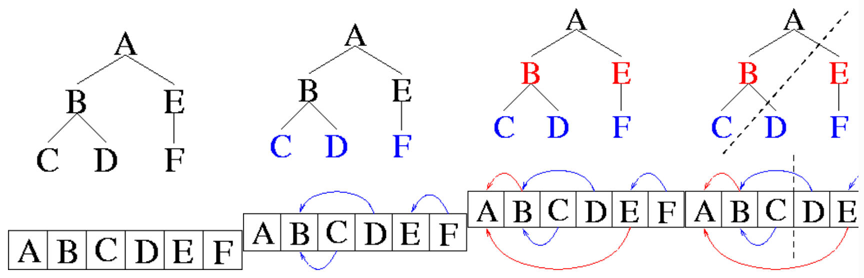
Genetic Operators

- Perform unconstrained Evolutionary Search
- GE employs standard operators of Genetic Algorithms
  - Point mutation, one-point crossover etc.
- Sometimes modified version of one-point crossover, Sensible Crossover, is used:
  - Effective length
  - Actual length



Crossover

- What actually happens in crossover?
- Preliminary : Visualisation.



- Crossover is performed at *genotypic* level

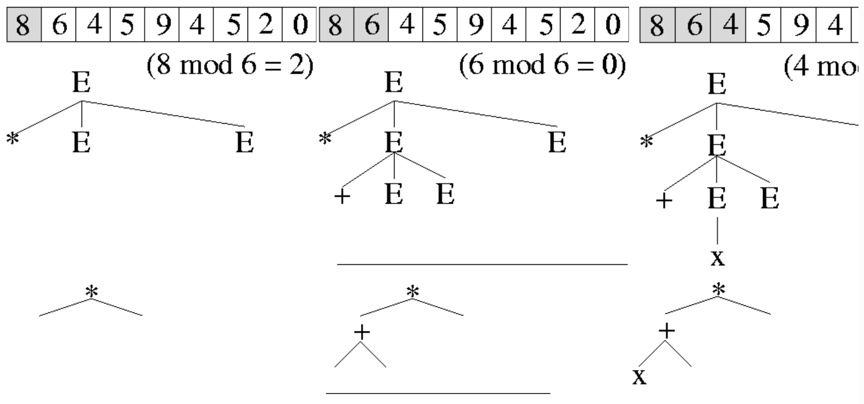
Ripple Crossover

- Analyse 1-point crossover in terms of derivation & syntax trees
- Use a *closed* grammar

```
E::= (+ E E)  {0}
    |(- E E)  {1}
    |(- E E)  {2}
    |(- E E)  {3}
    |X        {4}
    |Y        {5}
```

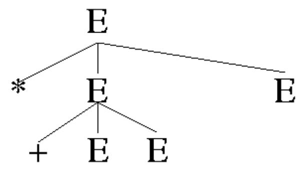
- No polymorphism, because there is only one non-terminal, i.e. one *context*

Different Views of Crossover



Rebuilding individuals

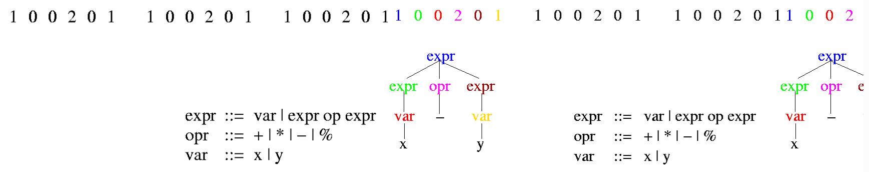
- Parent left with “spine”



- *Tail* swapped with other parent
  4 5 9 4 5 2 0 5 2 2
- Unmapped *E* terms must be mapped
- Use tail from other parent

Intrinsic Polymorphism

- With more than one non-terminal, a codon could be used differently in the offspring



Effects of Ripple Crossover

- Symbolic Regression Grammars

  Closed Grammar

```
E ::= x
    | (+ E E) | (* E E)
    | (- E E) | (/ E E)
```

  And the context free grammar:

```
Exp ::= Var | Exp Op Exp
Var ::= x
Op ::= + | * | - | /
```

Effects (contd.)

- Santa Fe ant trail grammars

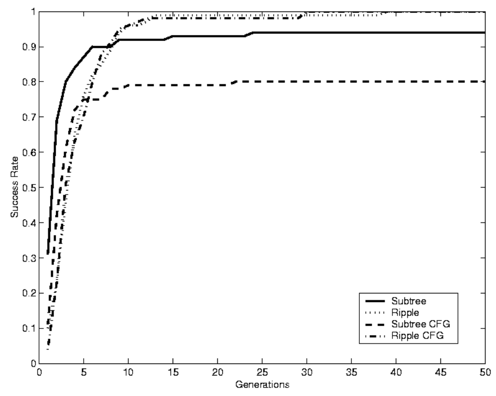
  Closed grammar

```
E ::= move() | left() | right()
    | iffoodahead(E E) | prog2(E, E)
```

  Context free grammar:

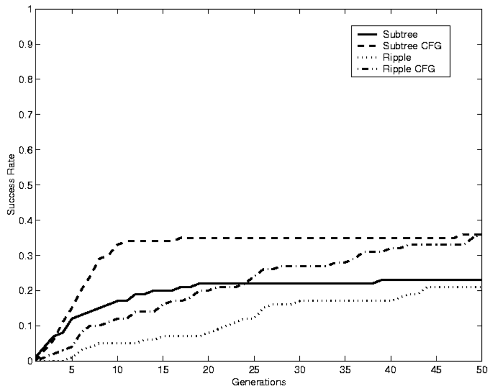
```
Code ::= Line | prog2(Line, Code)
Line ::= Condition | Action
Action ::= move() | right() | left()
Condition ::= iffoodahead(Code, Code)
```

Symbolic Regresssion Success Rates



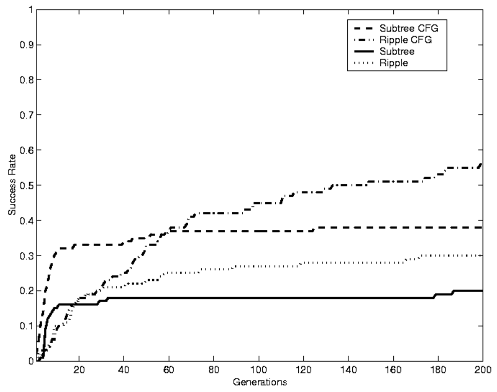
Both ripple crossovers start more slowly, but reach higher fitness.

Santa Fe Success Rates



Both ripple crossovers again start more slowly, but reach similar fitness.

Santa Fe - Extended Run



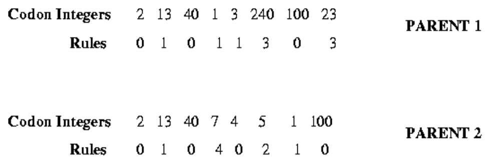
Success rates on the Santa Fe ant trail problem, averaged over 100 runs, for 250 generations. Ripple crossovers start slowly, but reach higher fitness.

Other types of Crossover?

- Homologous Crossover
  - Try not to cross in identical areas
- Uniform
- Same size homologous
- Same size two point

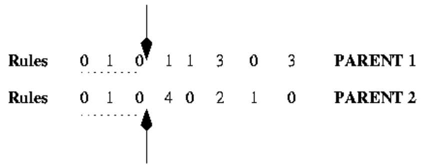
Homologous Crossover - First point

- Record rule histories for each individual



Codon Integers	2	13	40	1	3	240	100	23	**PARENT 1**
Rules	0	1	0	1	1	3	0	3	
Codon Integers	2	13	40	7	4	5	1	100	**PARENT 2**
Rules	0	1	0	4	0	2	1	0	

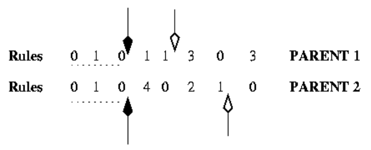
- Align rule histories of parents



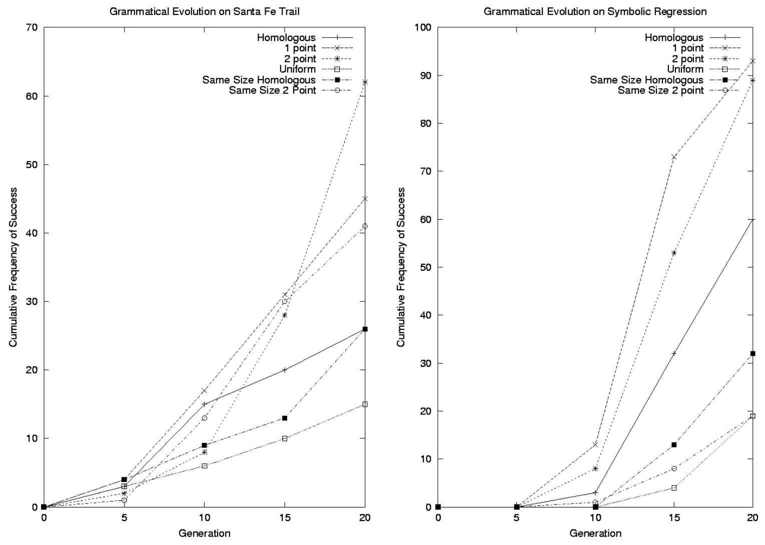
Rules	0	1	0	1	1	3	0	3	**PARENT 1**
Rules	0	1	0	4	0	2	1	0	**PARENT 2**

Homologous Crossover - Second Point

- Choose second point outside of area of similarity

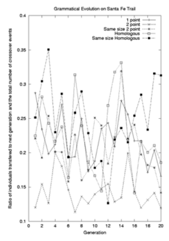
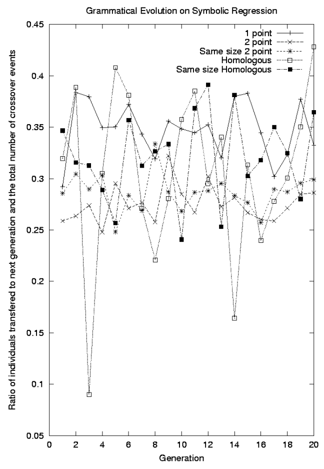

Rules 0 1 0 1 1 3 0 3 PARENT 1
Rules 0 1 0 4 0 2 1 0 PARENT 2

Crossover comparisons (Cumulative Freq. Success)



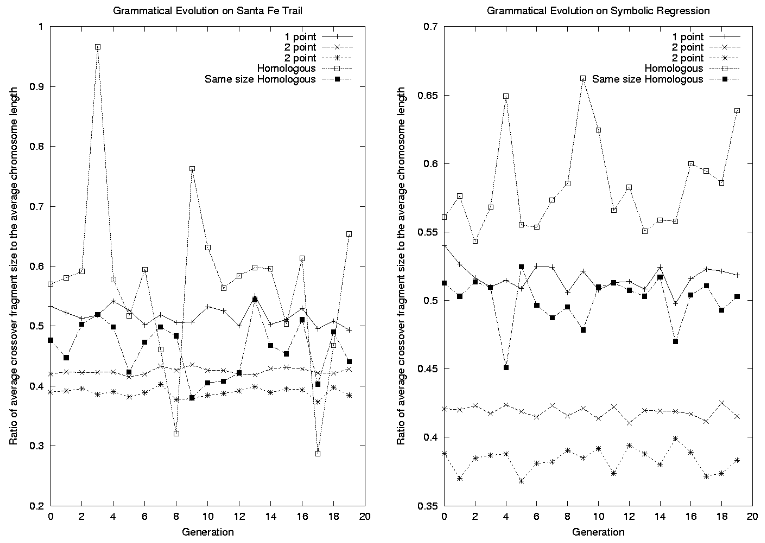
1pt/2pt best, uniform worst.

Productivity of Operators

Ratio of the number of individuals undergoing crossover that have been propagated to the next generation and the total number of crossover events occurring in that generation

Relative size of crossover fragments



Ratio of the average fragment size being swapped and the average chromosome length at each generation averaged over 20 runs.

Headless Chicken - Crossover or Macromutation

- Appears Crossover works
- 50% material exchange with 1-point over entire runs
- If useful material exchanged then swapping random fragments should degrade performance?

Headless Chicken Comparison

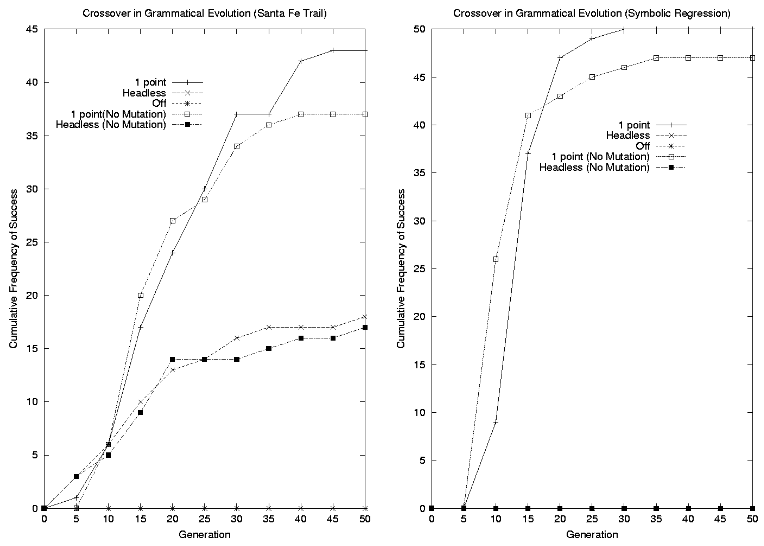


Figure: The Performance of Headless Chicken Crossover on Benchmark Problems.

The GAuGE System

Genetic Algorithms using Grammatical Evolution

Purpose:

- Position independent genetic algorithm;
- No under- or over-specification;
- Independent of search engine.

Based on mapping process (similar to GE):

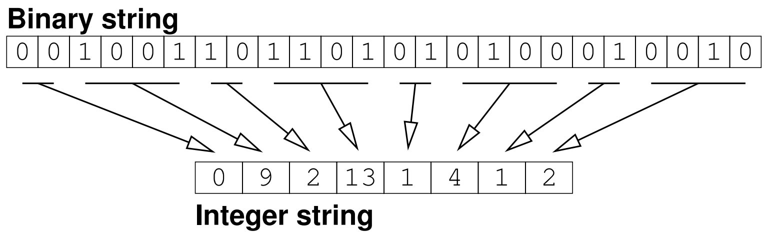
- Specify position and value of each variable at genotypic level;
- Map genotype strings into functional phenotype strings.

Mapping in the GAuGE System

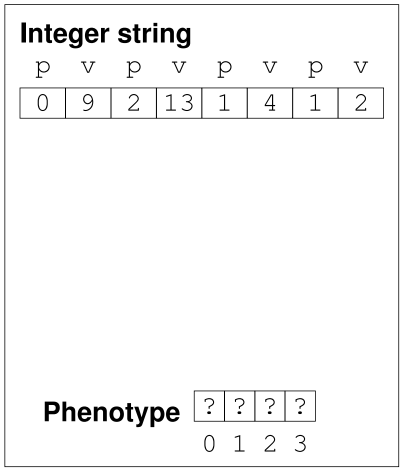
Transform binary string into integer string:

- Problem has 4 variables ($\ell = 4$), with range $0 \ldots 7$;
- Choose *position field size* ($pfs = 2$);
- Choose *value field size* ($vfs = 4$);
- Calculate binary string length:

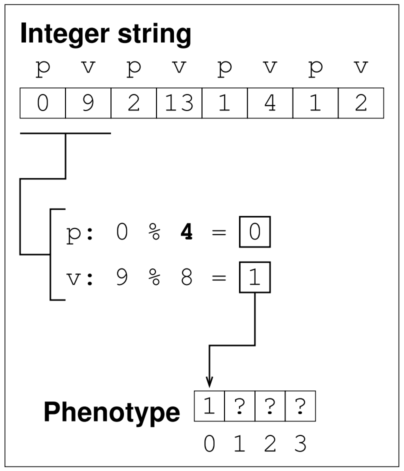
$$L = (pfs + vfs) \times \ell = (2+4) \times 4 = 24 \text{ bits}$$



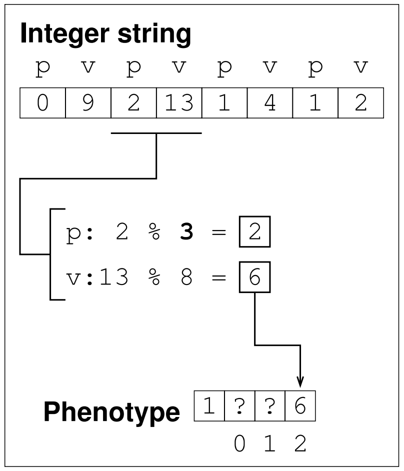
Calculating Phenotype



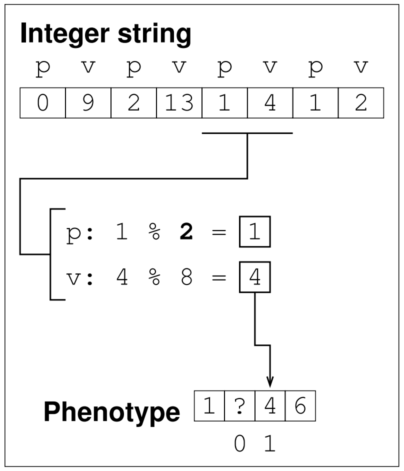
Calculating Phenotype



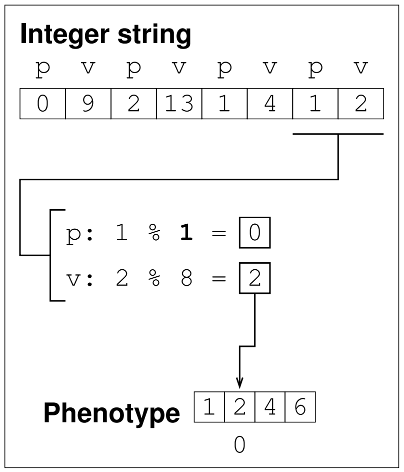
Calculating Phenotype



Calculating Phenotype



The GAuGE System



Where is Gauge useful?

- GAuGE adapts the representation to the problem
  - Useful where interactions between genes not known
- GAuGE is cheap
  - Far less complicated than algorithms that try to model gene interactions/relationships
- GAuGE discovers saliency
  - Most important genes end up on left side of strings

Chorus

- Mapping Independent Codons - no ripple effect
- Codon % *Total* number of rules in the grammar
- Competition between the Genes
- Concentration Table
- Variable length binary strings
- 8 bit codons

Grammar specification

```
S= <expr>
(0) <expr> ::= <expr> <op> <expr>
(1) | ( <expr> <op> <expr> )
(2) | <pre-op> ( <expr> )
(3) | <var>
(4) <op> ::= +
(5) | -
(6) | *
(7) | /
(8) <pre-op>::= Sin
(9) | Cos
(A) | Exp
(B) | Log
(C) <var> ::= 1.0
(D) | X
```

Mapping

Four non-terminals:

- <expr> 0..3, <op> 4..7, <pre-op> 8..B, <var> C..D

```
209 102 190 55 65 15 255 87
D 4 8 D 9 1 3 3
```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D
<e>	0	0	0	0	0	0	0	0	0	0	0	0	0	0
<e><o><e>	0	*0*	0	0	1	0	0	0	1	1	0	0	0	2
<v><o><e>	0	0	0	*0*	1	0	0	0	1	1	0	0	0	2
X<o><e>	0	0	0	0	1	0	0	0	1	1	0	0	0	*1*
X+<e>	0	0	0	0	*0*	0	0	0	1	1	0	0	0	1
X+<v>	0	0	0	*0*	0	0	0	0	1	1	0	0	0	1
X+X	0	0	0	0	0	0	0	0	1	1	0	0	0	*0*

Number of individuals wrapped

- Wrap Count & Invalid Individuals

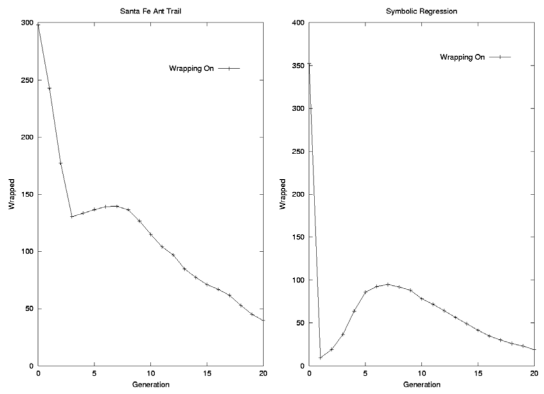


Figure: Number of individuals wrapped on the symbolic regression and Santa Fe trail problems.

Wrapping and Invalid Individuals

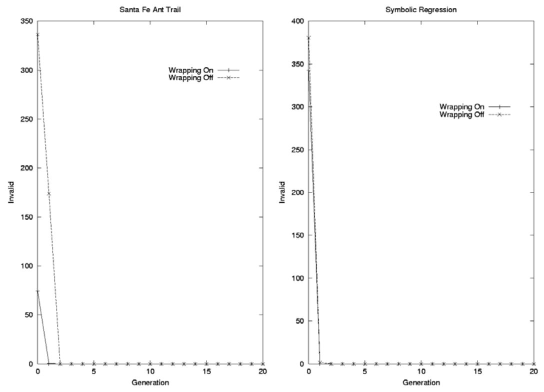


Figure: The number of invalid individuals for each generation in the presence and absence of wrappinge.

Performance

- Freq. of Success

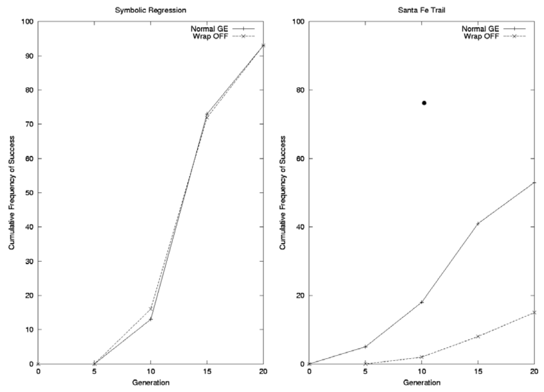


Figure: Figure shows the cumulative frequency of success measures on both problems with and without the presence of wrapping.

Definitions

- Actual length
  - Entire length of individual
- Effective length
  - Number of codons used
  - (Note! Can be less than or greater than actual length)

Genome Lengths

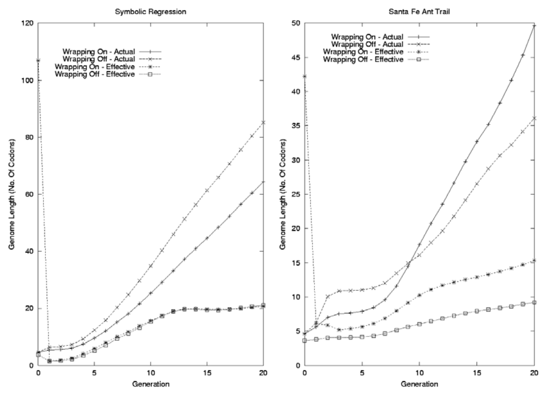


Figure: The figure shows the actual versus effective genome length for symbolic regression and the Santa Fe trail in the presence and absence of wrapping.

Summary

- For SR (left) wrapping off has the longest actual length
- Effective length virtually the same
- For SF (right) wrapping on longer in both cases.
- Conclusions:
  - Wrapping improves frequency of success on Santa Fe ant trail
  - No effect on Symbolic Regression cumulative frequency
  - Provides some constraint on genome lengths

Cumulative Freq. with and without degeneracy

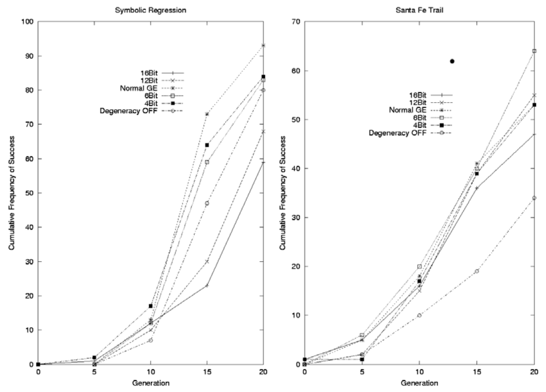


Figure: Cumulative frequencies of success for both problem domains in the presence and absence of genetic code degeneracy over 50 generations.

Variety

- No huge difference...
  - Normal, 4- and 6-bit top three in both
  - No degeneracy fourth in SR, last in SF
- Mean variety

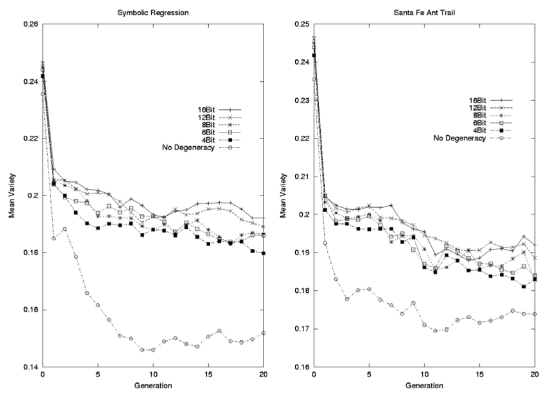


Figure: The figure shows the genetic code degeneracy and mean variety on symbolic regression and Santa Fe trail problems.

Unique Individuals

- Unique individuals

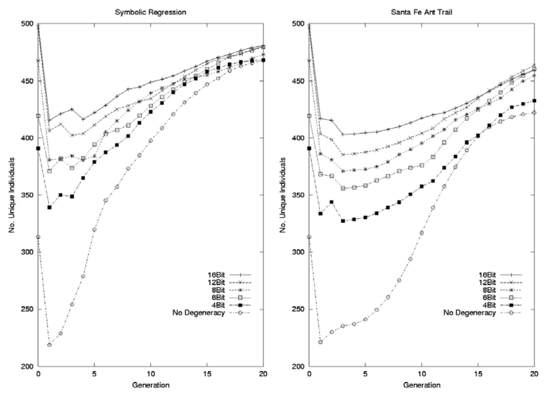


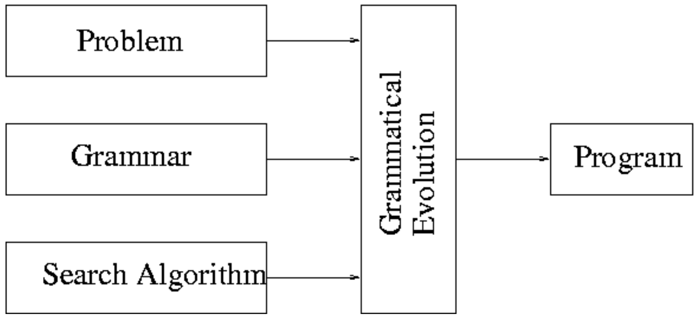
Figure: The figure shows genetic code degeneracy and unique individuals (for actual genome) on both problem domains.

- Conclusions:
  - Improves genetic diversity
  - Improves frequency of success on Santa Fe ant trail

Wrapping & Degeneracy

- Removing both....
  - Cumulative frequency of success degrades
  - Genome lengths increase over 60% on Symbolic Regression
  - Genetic diversity no worse than without degeneracy alone

Search Techniques



- Other techniques
  - Simulated Annealing
  - Hill Climbing
  - Random Search

Comparison

- Three standard GP problems
  - Santa Fe trail
  - Symbolic Integration (integrate Cos(x) + 2x + 1)
  - Symbolic regression $x^4 + x^3 + x^2 + x$

	Metaheuristic			
Problem	RS	HC	SA	GA
Santa Fe	54%	7%	14%	81%
Symbolic Integration	66%	4%	3%	100%
Symbolic Regression	0%	0%	0%	59%

The Future

- The Grammar (Attribute Grammars)
- Search & Evolutionary Dynamics
- Applications
- Newest Code Release
  - http://waldo.csisdmz.ul.ie/libGE/