

Gramáticas evolutivas

CARLOS CERVIGÓN



Evolución gramatical

- Evolución gramatical (GE) es una técnica evolutiva que puede hacer evolucionar programas en cualquier lenguaje, y se puede considerar una forma de programación genética basada en gramáticas.
- En vez de representar los programas en forma de árbol de análisis sintáctico (como en PG), se utiliza una representación lineal del genoma (array de enteros).

42 22 6 104 70 31 13 4 25 9 3 86 44 48 3 27 4 111 56 2

- Se emplea un mapeo genotipo-fenotipo: cada individuo usa cadenas 8 bits para representar los valores numéricos (codones).
- Los codones guardan la información para seleccionar una regla de producción en la representación BNF.

Evolución gramatical

- La gramática permite la generación de programas sintácticamente correctos de un lenguaje
- El usuario puede adaptar la gramática para producir simplemente soluciones sintácticamente válidas o puede incorporar conocimiento del dominio polarizando la gramática para producir frases muy específicas.
- La gramática se utiliza en el proceso de construcción de un programa mediante la aplicación de reglas de producción, comenzando desde el símbolo inicial de la gramática.

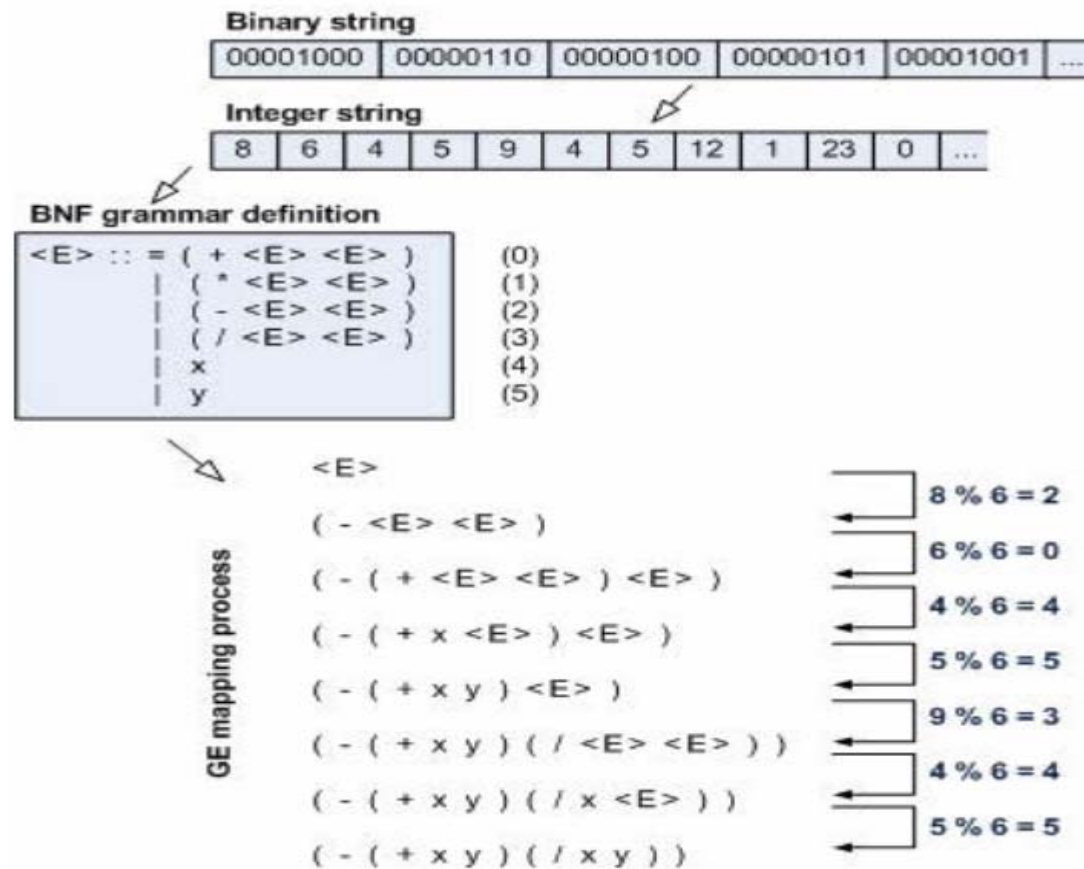
Evolución gramatical

Con el fin de seleccionar una regla de producción a aplicar, el siguiente valor codón en el genoma se lee e interpreta de izquierda a derecha y se obtiene según la siguiente fórmula:

$$\text{Regla} = c \% r$$

donde c es el valor entero codificado en el codón, r es el número de opciones de la regla para el no terminal actual y $\%$ es el operador módulo

Esquema



Gramática

Definimos una gramática en BNF (un subconjunto de un lenguaje)

Permite describir el lenguaje de salida que debe producir el sistema

- Terminales: elementos que pueden aparecer en el lenguaje
- No terminales: elementos que se expanden como otros terminales o no terminales

Una gramática se puede representar por la tupla $\{N, T, P, S\}$

Gramática

$N = \{ \langle \text{expr} \rangle, \langle \text{biop} \rangle, \langle \text{uop} \rangle, \langle \text{bool} \rangle \}$

$T = \{ \text{and}, \text{or}, \text{xor}, \text{nand}, \text{not}, \text{true}, \text{false}, (,) \}$

$S = \{ \langle \text{expr} \rangle \}$

y P se representa como:

(A) $\langle \text{expr} \rangle ::= (\langle \text{expr} \rangle \langle \text{biop} \rangle \langle \text{expr} \rangle) \mid \langle \text{uop} \rangle \langle \text{expr} \rangle \mid \langle \text{bool} \rangle$

(B) $\langle \text{biop} \rangle ::= \text{and} \mid \text{or} \mid \text{xor} \mid \text{nand}$

(C) $\langle \text{uop} \rangle ::= \text{not}$

(D) $\langle \text{bool} \rangle ::= \text{true} \mid \text{false}$

Gramática

- El código generado estará formado por elementos del conjunto T
- El proceso evolutivo hace evolucionar las reglas de producción que se van aplicando en cada etapa, empezando por el símbolo inicial S hasta llegar a un programa formado sólo por terminales
- Para la gramática BNF anterior, la siguiente tabla resume las reglas de producción y el número de opciones asociadas con cada una.

Regla	opciones
A	3
B	4
C	1
D	2

Mapeo genotipo-fenotipo

El genotipo se utiliza para mapear el símbolo de inicio en terminales mediante la lectura de los codones para generar un valor entero, a partir del cual se selecciona la regla de producción adecuada utilizando la función de mapeo:

$$\text{Regla} = c \bmod r$$

donde c es el valor entero del codón y r es el número de opciones de regla para el actual símbolo no terminal.

Mapeo genotipo-fenotipo

```
(B) <boolop> ::=      and (0)
                        | or      (1)
                        | xor     (2)
                        | nand    (3)
```

Si suponemos que el codón leído es 6 y nos toca derivar la regla B

$$6 \bmod 4 = 2$$

seleccionaría la regla (2) **xor** .

Es decir, **<boolop>** se reemplaza con **xor**. Cada vez que hay que seleccionar una regla de producción para transformar un no terminal, se lee otro codón, recorriendo el genoma.

wrapping

- Si agotamos los codones, se aplica el operado *wrap*, que pasa al primer codón del individuo, procediendo a reutilizarlo.
- El *wrapping* es opcional
- En GE cada vez que se usa el mismo codón siempre generará el mismo valor entero, pero dependiendo del no-terminal que se aplique se puede obtener una regla de producción diferente.

Sistema de comercio simplificado

`<S> ::= <tradingrule>`

`<tradingrule> ::= if(<signal>) {<trade>;} else {<trade>;}`

`<signal> ::= <value> <relop> <var>
 | (<signal>) AND (<signal>)
 | (<signal>) OR (<signal>)`

`<value> ::= <int-const> | <real-const>`

`<relop> ::= <= | >=`

`<trade> ::= buy
 | sell
 | do-nothing`

`<int-const> ::= <int-const><int-const>
 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

`<real-const> ::= 0.<int-const>`

`<var> ::= var0 | var1 | var2 | var3 | var4
 | var5 | var6 | var7 | var8 | var9`

Sistema de comercio simplificado

Considera el siguiente genotipo:

42 22 6 104 70 31 13 4 25 9 3 86 44 48 3 27 4 111 56 2

El primer codon es C = 42 y el símbolo no terminal es *<tradingrule>*.

Como solo tiene una regla ($42 \bmod 1 = 0$):

<tradingrule> ::= if(<signal>) {<trade>;} else {<trade>;}

Se reemplaza automáticamente por la parte derecha:

if(<signal>) {<trade>;} else {<trade>;}

Al seleccionar el No terminal más a la izquierda *<signal>* hay 3 posibles reemplazamientos

Sistema de comercio simplificado

Avanzamos un codón:

42 **22** 6 104 70 31 13 4 25 9 3 86 44 48 3 27 4 111 56 2

La segunda regla de producción es $22 \bmod 3 = 1$, por lo que obtenemos:

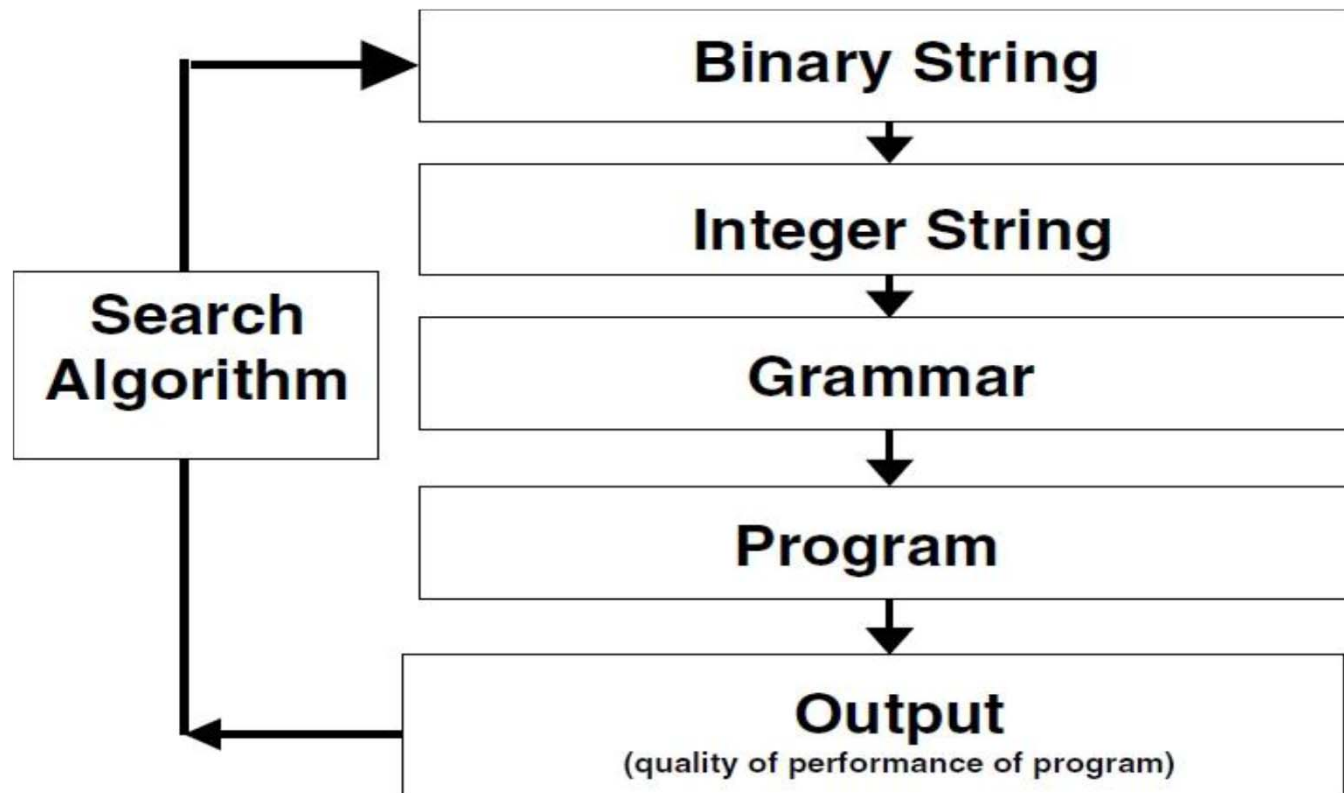
```
if((<signal>) AND (<signal>)) {<trade>;} else {<trade>;}
```

Al recorrer todo el genotipo obtenemos la fexpresión final:

```
if((13 <= var5) AND (0.64 <= var3)) {buy();}  
else {sell();}
```

Las variables (*var0 a var9*) podrían ser una selección de elementos de información, por ejemplo, *var5* podría ser una tasa *P/E* y *var3* podría representar el crecimiento de ventas de los últimos 3 años.

Esquema



Algoritmo GE

- 1) Definir la sintaxis BNF para traducir cadena de bits a un programa
- 2) Definir una población inicial con individuos generados aleatoriamente.
- 3) Traducir cadena de bits a un programa de acuerdo a la sintaxis BNF.
- 4) Estimar la función de aptitud del individuo.
- 5) Actualizar la población.
- 6) Terminar el proceso cuando sea necesario.
- 7) Ir al paso 3).

Algoritmo GE

Para traducir la cadena de números a un programa:

- Recorrer los valores del genoma para decidir qué regla aplicar en cada momento según la formula

Regla= codón % número de alternativas de la regla actual.

- Reemplazar el símbolo elegido entre los candidatos posibles.

Ejemplo

(1) $\langle \text{expr} \rangle$::= $\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$ (A)
 | $(\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle)$ (B)
 | $\langle \text{pre-op} \rangle (\langle \text{expr} \rangle)$ (C)
 | $\langle \text{var} \rangle ;$ (D)

(2) $\langle \text{op} \rangle$::= $+$ (A)
 | $-$ (B)
 | $/$ (C)
 | $* ;$ (D)

(3) $\langle \text{pre-op} \rangle$::= $+$ (A)
 | $- ;$ (B)

(4) $\langle \text{var} \rangle$::= $X ;$ (A)

Ejemplo

Genoma de partida:

220	203	17	3	109	215	104	30
-----	-----	----	---	-----	-----	-----	----

$S \rightarrow \langle \text{expr} \rangle$ con 4 alternativas : $220 \bmod 4 = 0$

Se reemplaza $\langle \text{exp} \rangle$ por $\langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

220	203	17	3	109	215	104	30
-----	-----	----	---	-----	-----	-----	----

$S \rightarrow \langle \text{expr} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$ con 4 alternativas : $203 \bmod 4 = 3$

Se reemplaza $\langle \text{expr} \rangle$ por $\langle \text{var} \rangle$

$S \rightarrow \langle \text{var} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

Ejemplo

$S \rightarrow \langle \text{var} \rangle \langle \text{op} \rangle \langle \text{expr} \rangle$

var solo tiene una alternativa:

$S \rightarrow X \langle \text{op} \rangle \langle \text{expr} \rangle$

220	203	17	3	109	215	104	30
-----	-----	----	---	-----	-----	-----	----

$S \rightarrow X \langle \text{op} \rangle \langle \text{expr} \rangle$ con 4 alternativas : $17 \bmod 4 = 1$

$S \rightarrow X - \langle \text{expr} \rangle$

220	203	17	3	109	215	104	30
-----	-----	----	---	-----	-----	-----	----

$S \rightarrow X - \langle \text{expr} \rangle$ con 4 alternativas : $3 \bmod 4 = 3$

$S \rightarrow X - \langle \text{var} \rangle$ $S \rightarrow X - X$

220	203	17	3	109	215	104	30
-----	-----	----	---	-----	-----	-----	----

Solución candidata

intrones

Ejemplos de gramáticas

Gramáticas para generar constantes reales:

$\langle \text{real} \rangle ::= \langle \text{entero} \rangle \langle \text{punto} \rangle \langle \text{entero} \rangle \mid$
 $\langle \text{entero} \rangle \langle \text{entero} \rangle ::= \langle \text{entero} \rangle \langle \text{digito} \rangle \mid$
 $\langle \text{digito} \rangle \langle \text{digito} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
 $\langle \text{punto} \rangle ::= .$

Target Constant

5.67

24.35

20021.11501

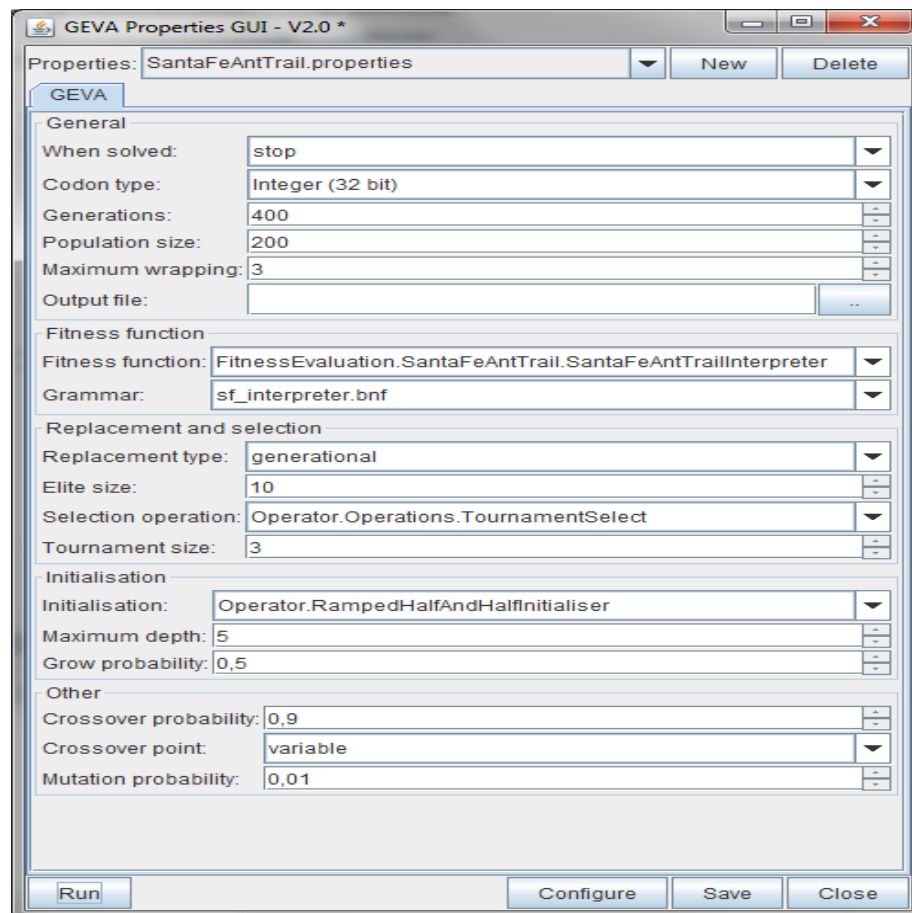
$\langle \text{value} \rangle ::= \langle \text{value} \rangle \langle \text{op} \rangle \langle \text{value} \rangle \mid (\langle \text{value} \rangle) \mid \langle \text{number} \rangle \mid \langle \text{func} \rangle (\langle \text{number} \rangle)$
 $\langle \text{op} \rangle ::= + \mid - \mid / \mid *$
 $\langle \text{number} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
 $\langle \text{func} \rangle ::= \text{Math.sin} \mid \text{Math.cos} \mid \text{Math.tan}$
 $\langle \text{func} \rangle ::= \text{Math.sin} \mid \text{Math.cos} \mid \text{Math.tan}$
 $\langle \text{value} \rangle ::= \langle \text{value} \rangle \langle \text{op} \rangle \langle \text{value} \rangle \mid (\langle \text{value} \rangle) \mid \langle \text{number} \rangle \mid$

Framework GEVA

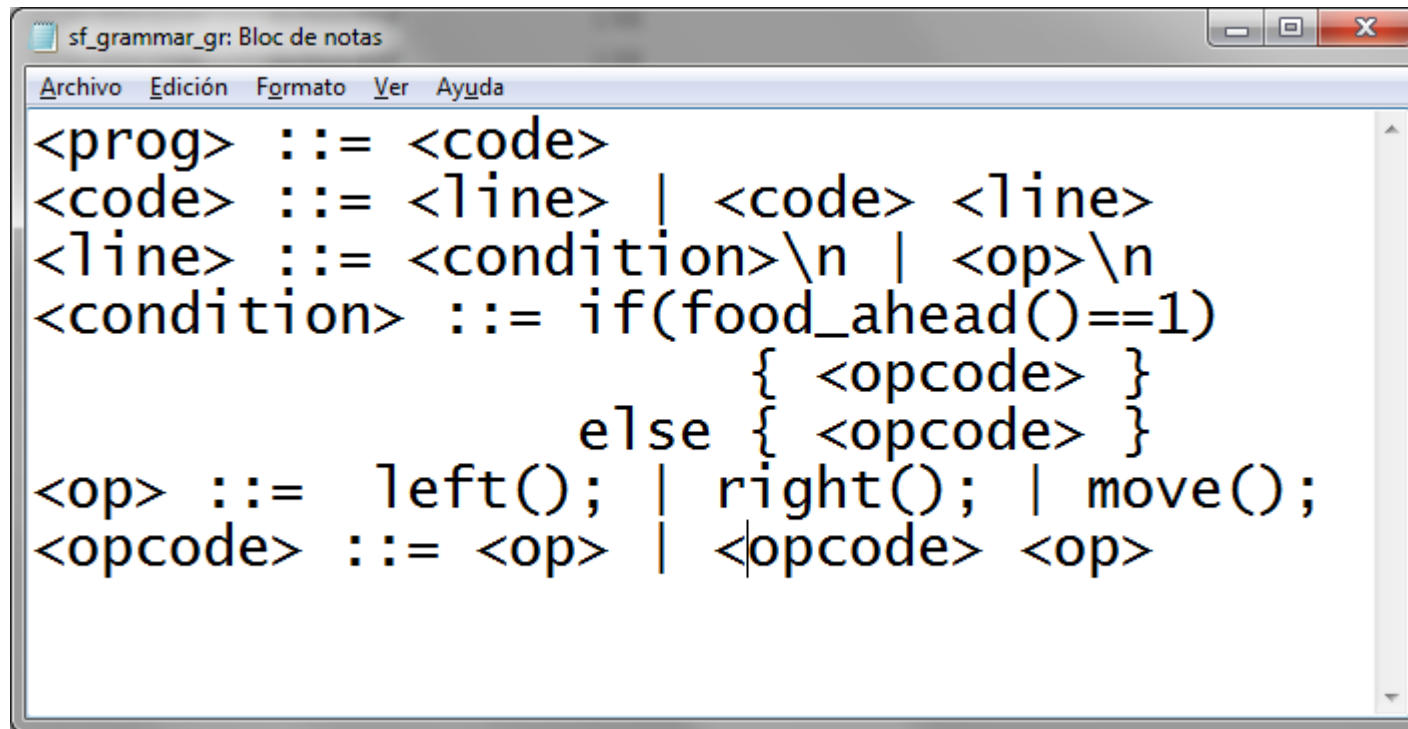
<http://ncra.ucd.ie/GEVA.html>

- GEVA is an implementation of Grammatical Evolution in Java developed at UCD's Natural Computing Research & Applications group.
- As well as providing the characteristic genotype-phenotype mapper of GE a search algorithm engine, and GUI are also provided.
- Currently a variable-length integer encoding Evolutionary Algorithm search engine is provided.

GEVA

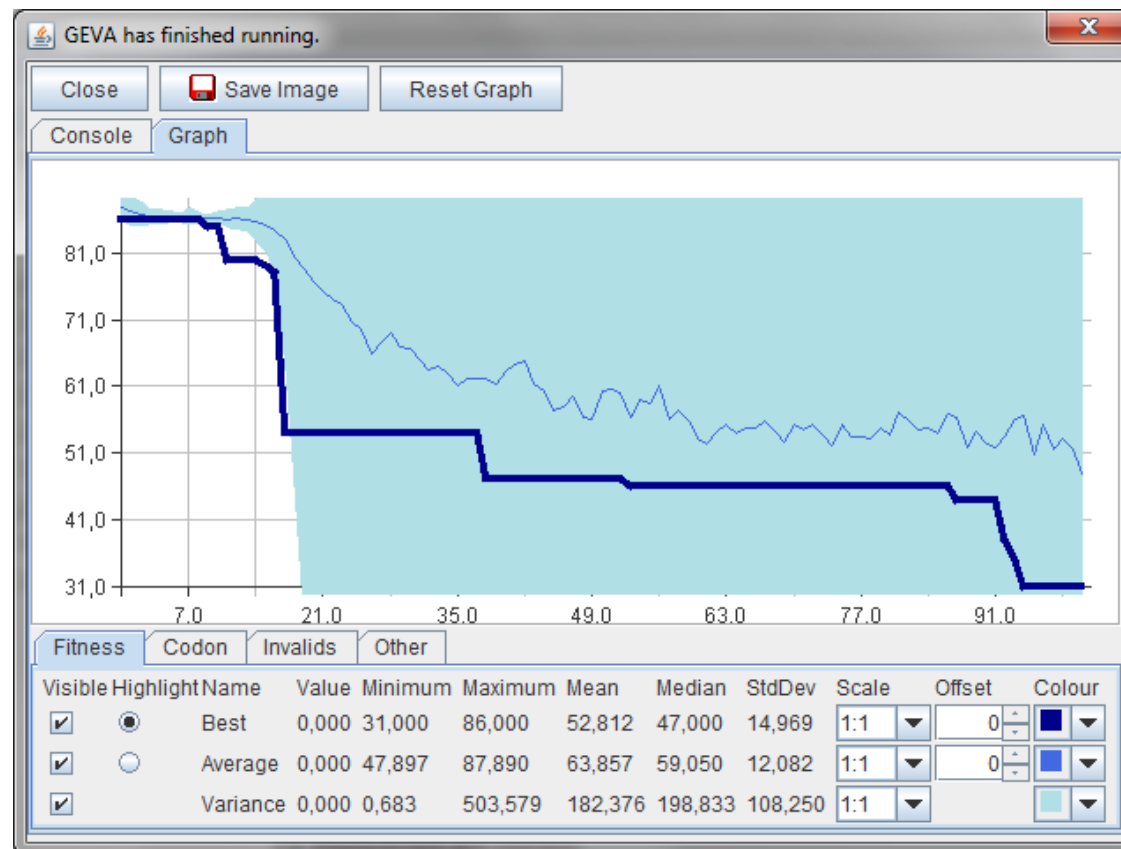


GEVA

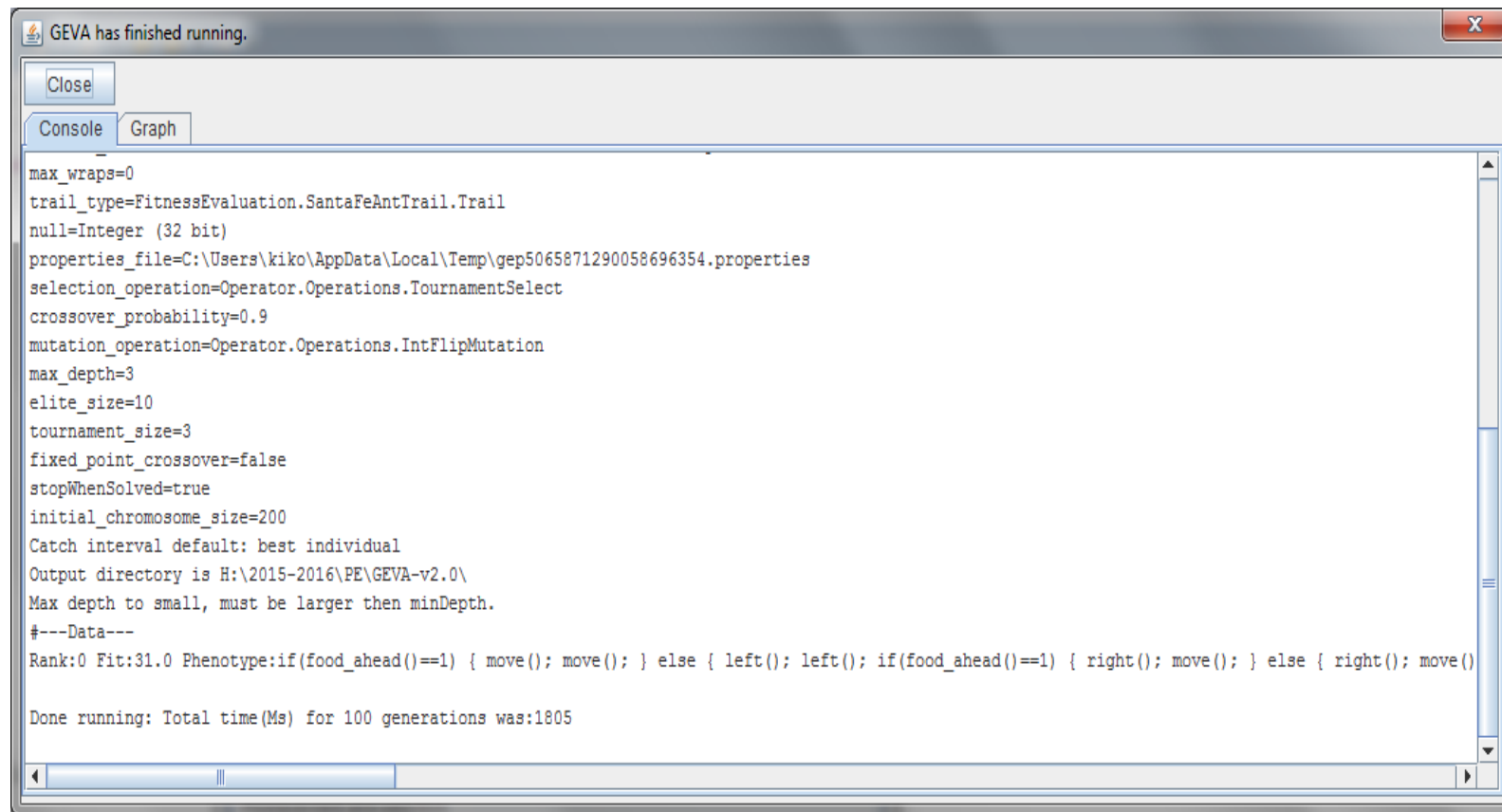


```
<prog> ::= <code>
<code> ::= <line> | <code> <line>
<line>  ::= <condition>\n | <op>\n
<condition> ::= if(food_ahead()==1)
                    { <opcode> }
                    else { <opcode> }
<op> ::= left(); | right(); | move();
<opcode> ::= <op> | <opcode> <op>
```


GEVA



GEVA



The screenshot shows a window titled "GEVA has finished running." with a "Close" button. Below the title bar are two tabs: "Console" (selected) and "Graph". The console displays the following text:

```
max_wraps=0
trail_type=FitnessEvaluation.SantaFeAntTrail.Trail
null=Integer (32 bit)
properties_file=C:\Users\kiko\AppData\Local\Temp\gep5065871290058696354.properties
selection_operation=Operator.Operations.TournamentSelect
crossover_probability=0.9
mutation_operation=Operator.Operations.IntFlipMutation
max_depth=3
elite_size=10
tournament_size=3
fixed_point_crossover=false
stopWhenSolved=true
initial_chromosome_size=200
Catch interval default: best individual
Output directory is H:\2015-2016\PE\GEVA-v2.0\
Max depth to small, must be larger then minDepth.
#---Data---
Rank:0 Fit:31.0 Phenotype:if(food_ahead()==1) { move(); move(); } else { left(); left(); if(food_ahead()==1) { right(); move(); } else { right(); move(); }
Done running: Total time(Ms) for 100 generations was:1805
```

PG vs GE

<http://www.tc33.org/genetic-programming/genetic-programming-software-comparison/>