CSE/ECE 848 Introduction to Evolutionary Computation

Module 3 - Lecture 11 - Part 4

Genetic Programming - Linear Representation

Wolfgang Banzhaf, CSE
John R. Koza Chair in Genetic Programming

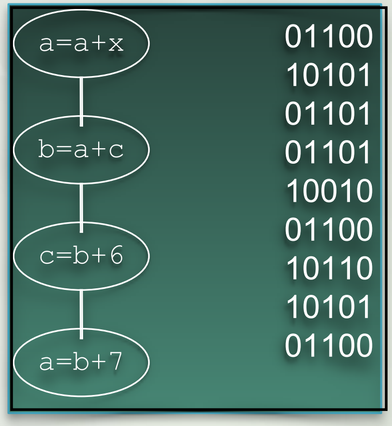
Representations

- Tree GP
- Linear GP
  - AIM GP
- Graph GP (PADO example)
- Cellular Encoding
- CF Grammar GP

Linear Genetic Programming (LGP)

Elements of LGP

- Follows principles of imperative languages
- Based on instruction sequences: Each instruction is a gene
- Each instruction contains the elements of operator and operand(s) and an assignment
- Bit sequences code for operators (op-code) and operands (register addresses)
- Close relatitionship between machine code and interpretation



Sample Program

- Registers and constants
- Operators like +,-, sin(), >
- lines with “//” are irrelevant for the behavior of the program
- r[i]:=r[j]+r[k] can be coded as (id(+),i,j,k) into 4 bytes

```
void gp(r)
  double r[8];
{  ...
   r[0] = r[5] + 71;
// r[7] = r[0] - 59;
   if (r[1] > 0)
   if (r[5] > 2)
     r[4] = r[2] * r[1];
// r[2] = r[5] + r[4];
   r[6] = r[4] * 13;
   r[1] = r[3] / 2;
// if (r[0] > r[1])
//   r[3] = r[5] * r[5];
   r[7] = r[6] - 2;
// r[5] = r[7] + 15;
   if (r[1] <= r[6])
     r[0] = sin(r[7]);
}
```

Instruction Sets

Instruction type	General notation	Input range
Arithmetic operations	$r_i := r_j + r_k$ $r_i := r_j - r_k$ $r_i := r_j \times r_k$ $r_i := r_j \;/\; r_k$	$r_i, r_j, r_k \in \mathbb{R}$
Exponential functions	$r_i := {r_j}^{(r_k)}$ $r_i := e^{r_j}$ $r_i := ln(r_j)$ $r_i := {r_j}^2$ $r_i := \sqrt{r_j}$	$r_i, r_j, r_k \in \mathbb{R}$
Trigonomic functions	$r_i := sin(r_j)$ $r_i := cos(r_j)$	$r_i, r_j, r_k \in \mathbb{R}$
Boolean operations	$r_i := r_j \wedge r_k$ $r_i := r_j \vee r_k$ $r_i := \neg\, r_j$	$r_i, r_j, r_k \in \mathbb{B}$
Conditional branches	$if\ (r_j > r_k)$ $if\ (r_j \leq r_k)$ $if\ (r_j)$	$r_j, r_k \in \mathbb{R}$ $r_j \in \mathbb{B}$

Protected instructions

Instruction	Protected definition			
$r_i := r_j \ / \ r_k$	$if \ (r_k \neq 0)$	$r_i := r_j \ / \ r_k$	$else$	$r_i := r_j + c_{undef}$
$r_i := {r_j}^{r_k}$	$if \ (\|r_k\| \leq 10)$	$r_i := \|r_j\|^{r_k}$	$else$	$r_i := r_j + r_k + c_{undef}$
$r_i := e^{r_j}$	$if \ (\|r_j\| \leq 32)$	$r_i := e^{r_j}$	$else$	$r_i := r_j + c_{undef}$
$r_i := ln(r_j)$	$if \ (r_j \neq 0)$	$r_i := ln(\|r_j\|)$	$else$	$r_i := r_j + c_{undef}$
$r_i := \sqrt{r_j}$	$r_i := \sqrt{\|r_j\|}$			

Branching

- If condition not fulfilled: Skip one instruction
- Nested conditions possible

  "AND"

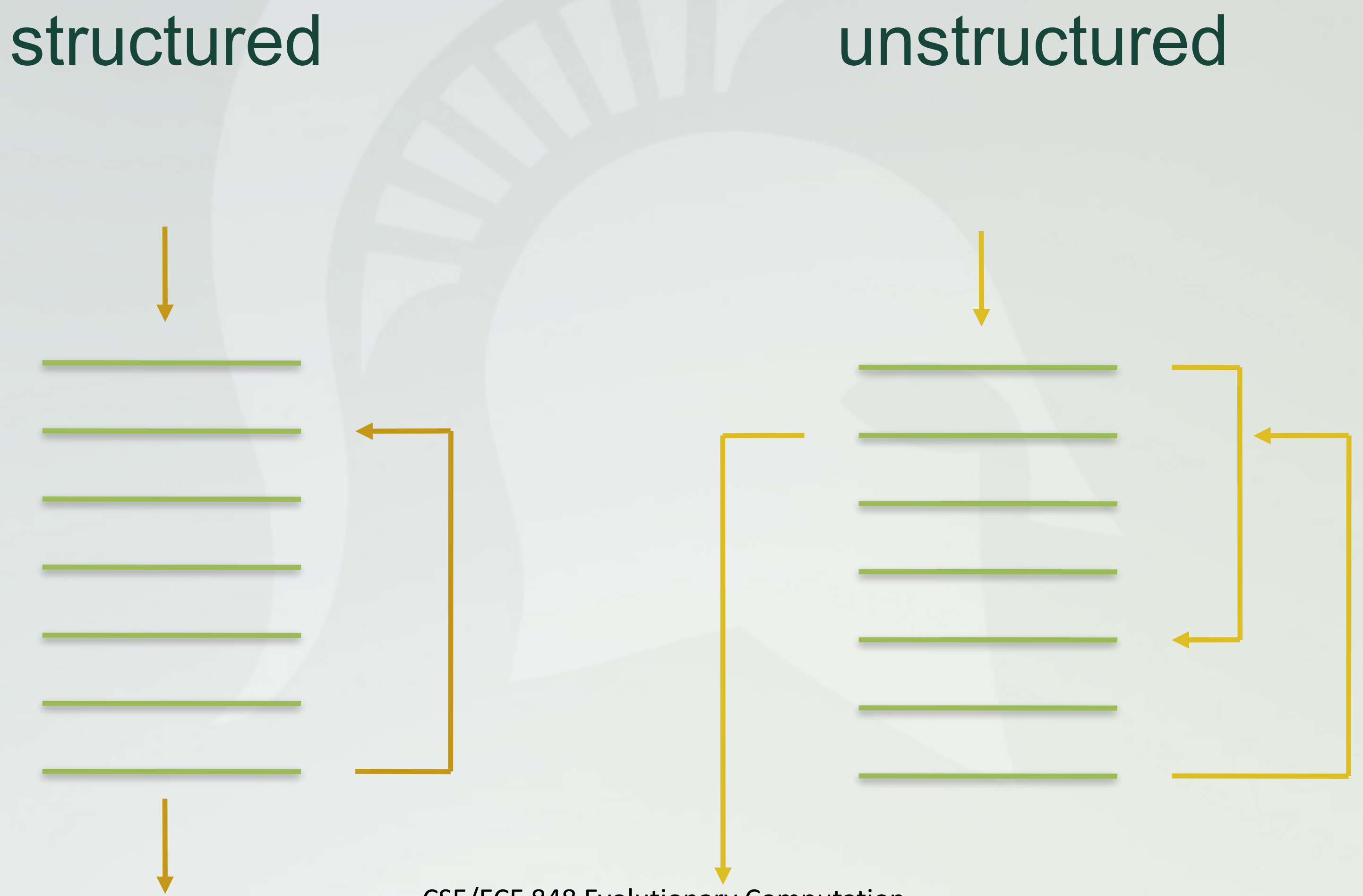
```
if (<cond1>)
if (<cond2>)
<oper>;
```

- Sequence of instructions

  "OR"

```
if (<cond1>)
<oper>;
if (<cond2>)
<oper>;
```

Loops



Initialization

Full

Length of sequences L_l

Choose until L_l:

Instructions $\in$ {Instruction set}

Operands $\in$ {Terminals (registers, constants)}

~"Grow" Random

Max. Length $L_{l_{max}}$; min. Length $L_{l_{min}}$

Choose random length $L_Z \in [L_{l_{min}} : L_{l_{max}}]$

Choose until L_Z:

Instructions $\in$ {Instruction set}

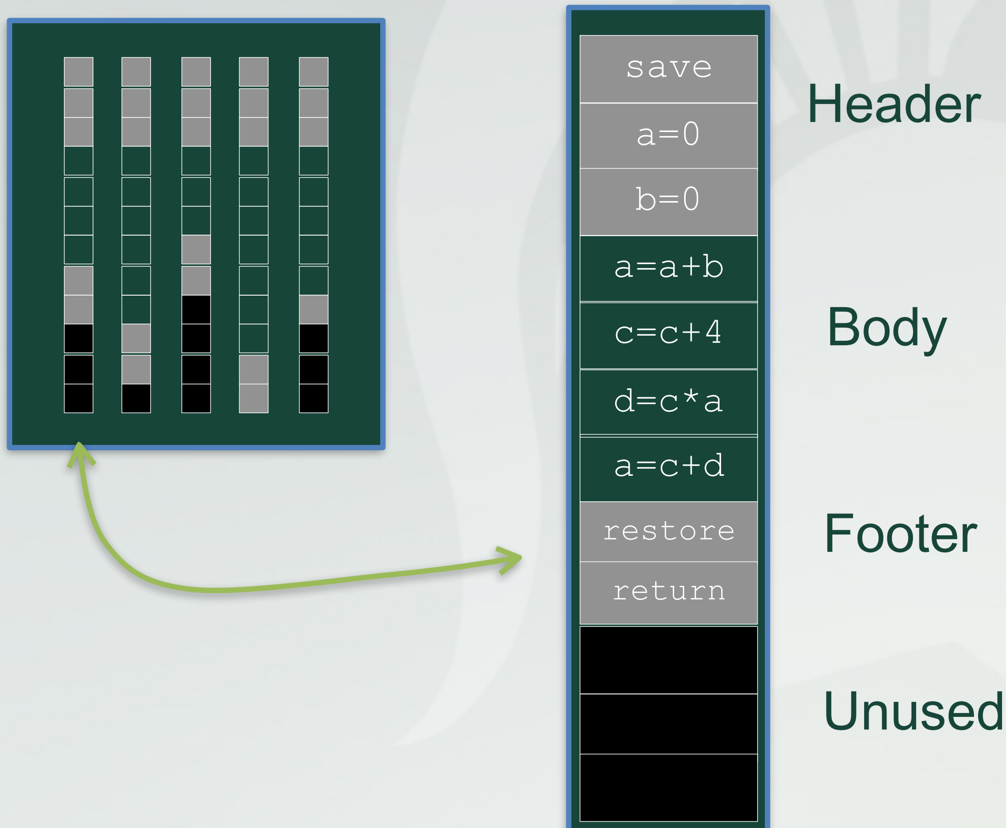
Operands $\in$ {Terminals}

AIMGP

Machine Language GP

(Automatic Induction of Machine Code GP)

AIMGP Population



Registers: a, b, c, d, e, f, g, ...
Constants
Arithmetic/Logic operations

Tournament algorithm:
Choose N individuals
Evaluate program fitness
Substitute worst programs
  through offspring of the
  better
Repeat

Mutation in AIMGP

A: Mutation in Operands

Point mutation

Op-code	Operand 1	Operand 2
01110010011	10001001011	110111010101

32-bit Instruction

Register address permitted?
Constant value permitted?
Op-code permitted?

B: Mutation in Op-code

Point mutation

Op-code	Operand 1	Operand 2
01110010011	10001001011	110111010101

32-bit Instruction

Crossover in AIMGP

A: Crossover between instructions

Crossover point

32 bit Instruction

011100001100110101101100011010110101	011100101110101100101100110110111010	110110100110101010101011

B: Crossover within instructions

Crossover point

= protected area

32 bit Instruction

011100001100110101101100011010110101	011100101110101100101100110110111010	110110100110101010101011

Two sample programs for a regression of y=x^2/2

```
unsigned int fn0 (i0, i1)
unsigned int i1
unsigned int i1;
{ i1 = 0;
   ⋮
  return i0
}
```

Header And Footer

* Non-expressed code

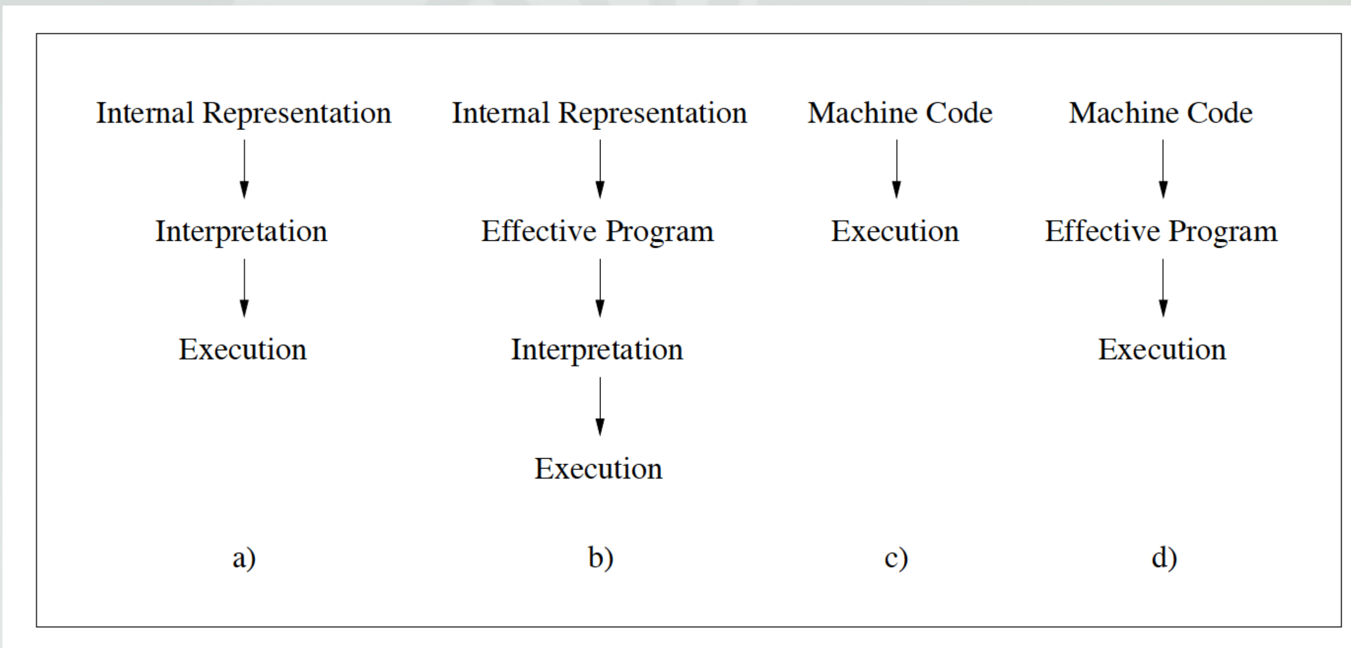
```
i1 = i1 + 849;   *
i1 = i1 + 2277;  *
i1 = i0 + i1;    *
i1 = i0 - i0;    *
i1 = i0 + i1;    *
i1 = i0 + 922;   *
i1 = i0 >> 1;    i1=i0/2
i0 = i0 * i1;    i0:=i0²/2
```

Individual A

```
i1 = i1 << 7;    *
i0 = i0 << 2;    i0:=i0*2²
i1 = i1 >> i0;   *
i0 = i0 + i0;    i0:=i0*2   (x*2³)
i1 = i1 << 21;   *
i1 = i1 >> i0;   *
i0 = i0 * i0;    i0:=i0²    (x²*2⁶)
i1 = i1 << i0;   *
i0 = i0 >> i0;   *
i0 = i0 >> 7;    i0:=i0/2⁷  (x²/2)
i1 = i1 - 4002;  *
```

Individual B

Execution Concepts



LGP

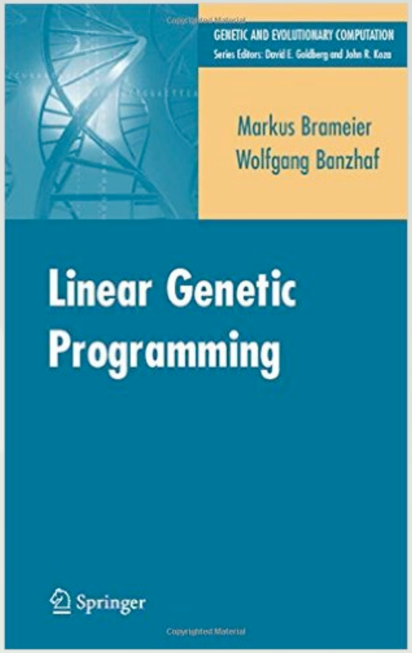
LGP optimized

AIMGP

AIMGP optimized

Linear GP

- Linear in sequence of instructions
- Natural way of coding
- Efficiency gains over Tree GP



2007