Lecture 5: Linear Genetic Programming

CIU036 Artificial Intelligence 2, 2010

Krister Wolff, Ph.D.

Department of Applied Mechanics
Chalmers University of Technology
41296 Göteborg, Sweden

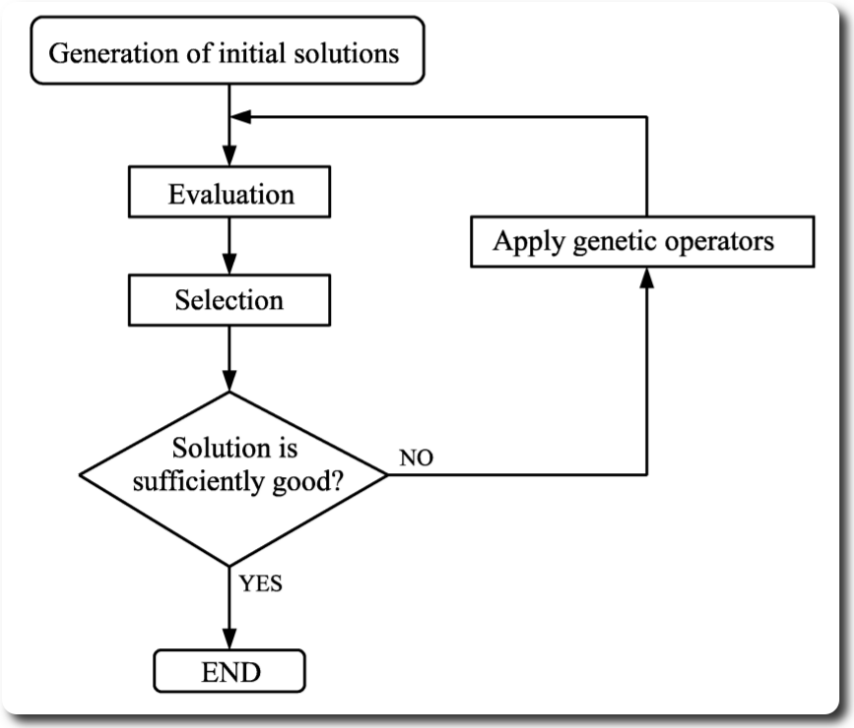
krister.wolff@chalmers.se

November 23, 2010

Table of Contents

Table of Contents

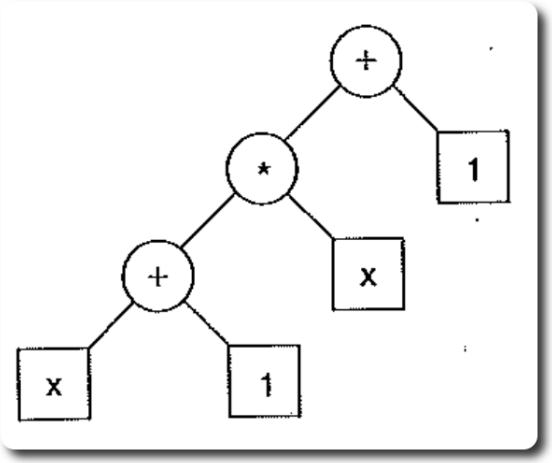
The flow of a generic EA



- The flow of the algorithm is pretty much the same as for GAs, but in **GP** the individuals of the population can be interpreted as *computer programs* (i.e. a list of instructions, written in a programming language, that is used to control the behavior of a machine).

Representation

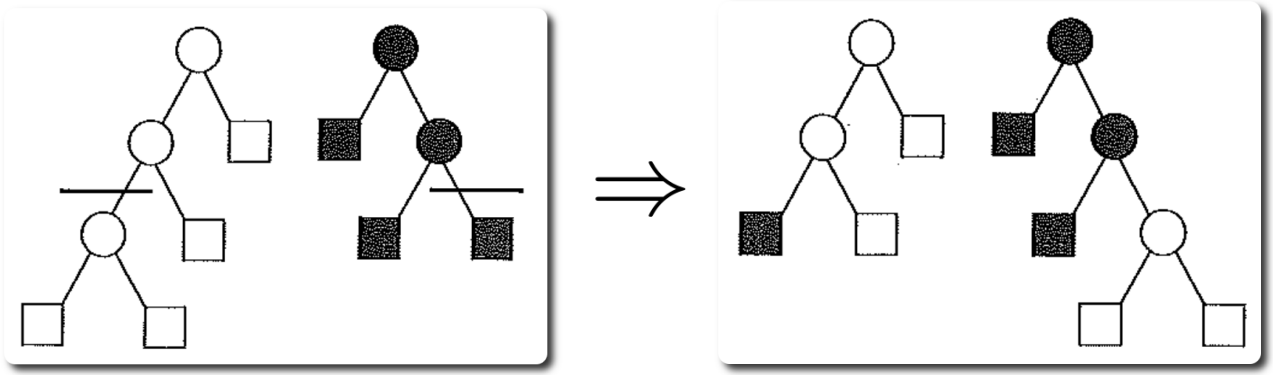
- In genetic programming the individuals are represented as **trees**:



- GP trees consist of **operators** (e.g. $+$, $*$) and **terminals** (e.g. x, 1).
- Evaluation proceeds repetedly from the left: $f(x) = (x + 1)x + 1$
- Here, the individual is representing a mathematical function. *Function fitting* is an important application area for GP.

Crossover and mutation

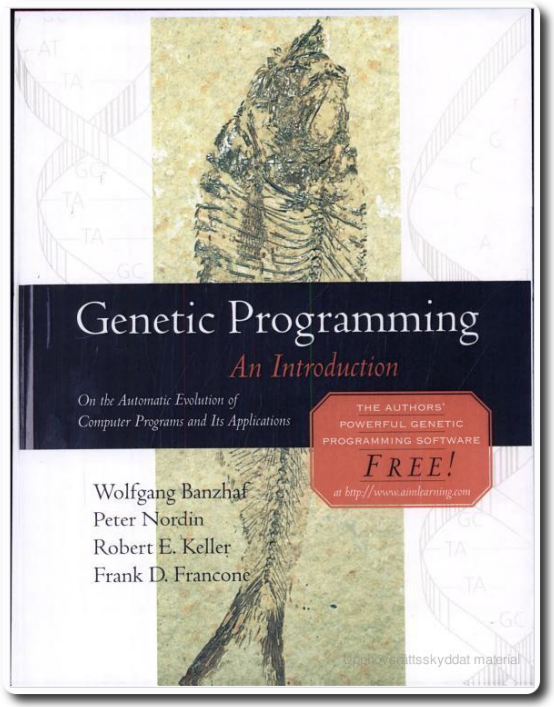
- **Crossover** in tree-based GP:



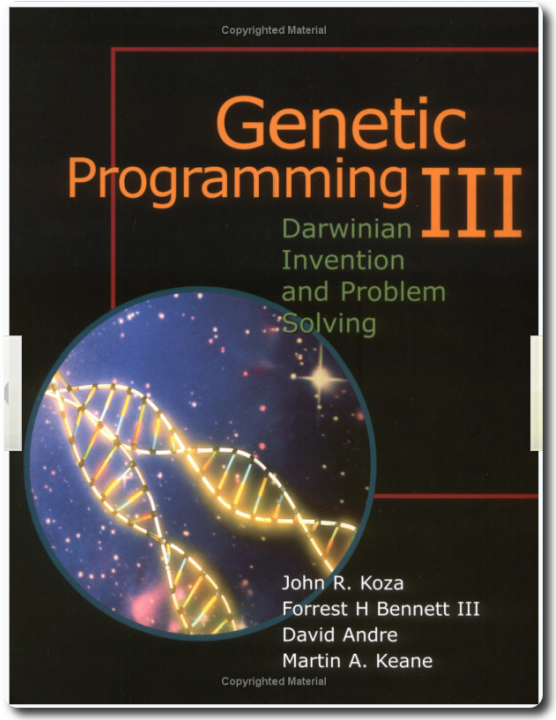
- **Mutation** in tree-based GP:
  - Randomly select a node in the tree and replace the existing tree at that point with a new subtree.
  - The new subtree is created in the same way as the initial population.

GP literature

- Books on **Genetic programming** (only two examples):



- Wolfgang Banzhaf *EtAl*; Genetic programming An introduction.



- John R. Koza *EtAl*; Genetic Programming III.

Table of Contents

The atoms of linear genetic programming

- Linear genetic programming (LGP) is a different "flavor" of GP. The individuals are represented by *linear structures* built up of **registers** and **instructions**:
- Registers
  - **Variable registers**; (denoted r_i) are utilized for *input*, *manipulating data*, and storing the resulting *output* from a calculation.
  - **Constant registers**; (denoted c_i) are *write protected* once they have been *initialized* with a value.
- Instructions
  - An **instruction** consists of *operator* (e.g. $+$, $-$, $\times$, $\div$) and *operand(s)* (i.e. the arguments): $r_3 := c_2 + r_1$
- Note that LGP facilitates the possibilty of using **multiple program outputs**, whereas tree-based GP provides only one program output.

LGP programs

- **Example**: A simple LGP program that consist of five *instructions*:

$$
\begin{aligned}
1: &\quad r_2 := r_1 + c_1 \\
2: &\quad r_3 := r_2 + r_2 \\
3: &\quad r_1 := r_2 \times r_3 \\
4: &\quad \mathit{if}\,(r_1 > c_2) \\
5: &\qquad r_1 := r_1 + c_1
\end{aligned}
$$

- LGP **program execution** starts with the *topmost* instruction, and proceeds *down* the list. It stops when the *last* instruction have been executed.
- **Conditional branching** (i.e. instruction 4): If `true`, the subsequent instruction (i.e. instruction 5) is executed, otherwise that instruction is skipped.

Examples of typical LGP operators

- A basic LGP **instruction set** (i.e. set of allowed instructions):

Instruction	**Description**	**Instruction**	**Description**
Addition	$r_i := r_j + r_k$	Sine	$r_i := \sin r_j$
Subtraction	$r_i := r_j - r_k$	Cosine	$r_i := \cos r_j$
Multiplication	$r_i := r_j \times r_k$	Square	$r_i := {r_j}^2$
Division[1]	$r_i := r_j / r_k$	Square root	$r_i := \sqrt{r_j}$
Exponentiation	$r_i := \mathrm{e}^{r_j}$	Conditional branch	`if` $r_i > r_j$
Logarithm	$r_i := \ln r_j$	Conditional branch	`if` $r_i \leq r_j$

- The **operand**(**s**) (r_j, r_k) , can either be variable registers or constant registers, but the **destination** (r_i) *must* be a variable register.

[1] see next slide

Semantic correctness of LGP programs

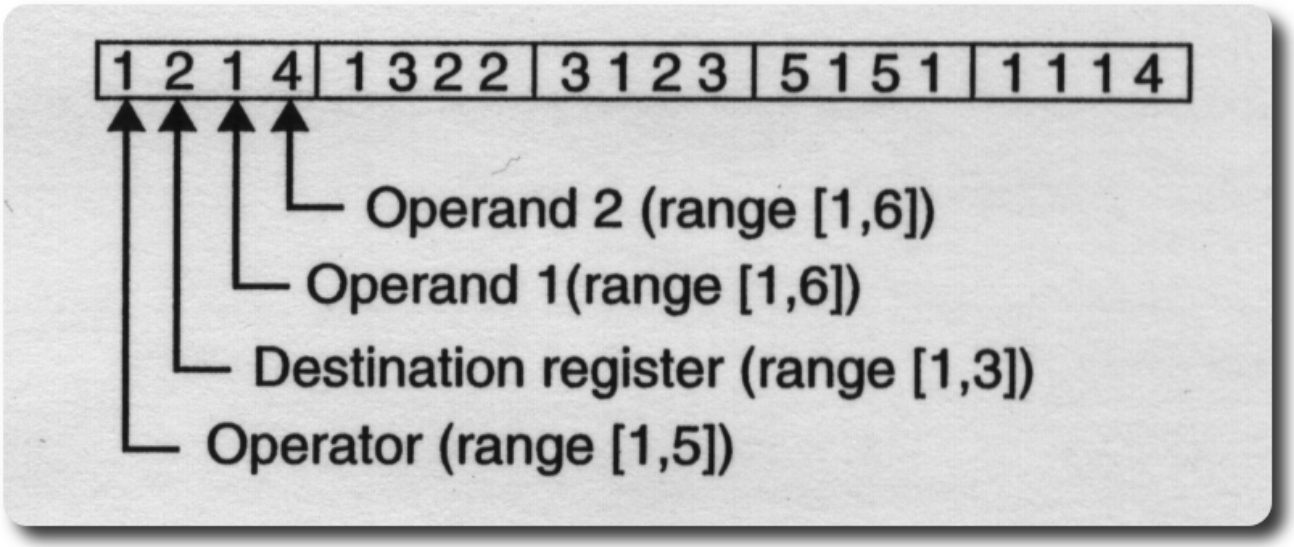
- An *invalid* instruction of an LGP program may cause a program crash. Examples of such problems are:
  - Illegal jump instruction (may arise from augmented branching instructions).
  - Division by zero.
- For the latter case a **protected division** operator is used:

$$r_i := \begin{cases} r_j / r_k & \text{if } r_k \neq 0 \\ c_{\max} & \text{otherwise} \end{cases}$$

  where $c_{\max}$ is a large pre-specified constant.
- Newly generate programs must be subject to *screening* in order to ensure semantic correctness.

LGP chromosomes

- Example of LGP encoding scheme. Let the **operator**, **operands**, and **destination** be represented as a sequence of integers[2]:



- Each **gene** consists of four integers, e.g. 1 2 1 4.
- In the case of **three number instructions** (e.g. Sine, Cosine) the fourt number of the gene should be ignored!

[2] **Misprint!** 5 1 5 1 should be 5 1 1 5

LGP encoding

- Consider the registers:
  - The set of **variable registers**: $\mathcal{R} = \{r_1, r_2, r_3\}$
  - The set of **constant registers**: $\mathcal{C} = \{c_1, c_2, c_3\}$
  - The union of these sets, $\mathcal{R} \cup \mathcal{C}$, is denoted $\mathcal{A} = \{r_1, r_2, r_3, c_1, c_2, c_3\}$.
- Consider the operators:
  - The set of **operators** *addition*, *subtraction*, *multiplication*, *division*, and the *conditional branching* $r_i > r_j$: $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5\}$
- Consider the first **gene** of the chromosome: 1 2 1 4.
  - 1: Operator $\in \mathcal{O}$ (addition)
  - 2: Destination $\in \mathcal{R}$ (variable register)
  - 1: Operand 1 $\in \mathcal{A}$ (any register)
  - 4: Operand 2 $\in \mathcal{A}$ (any register)
- Thus, the decoded **instruction** is: $r_2 := r_1 + c_1$.

Evaluation of LGP chromosome

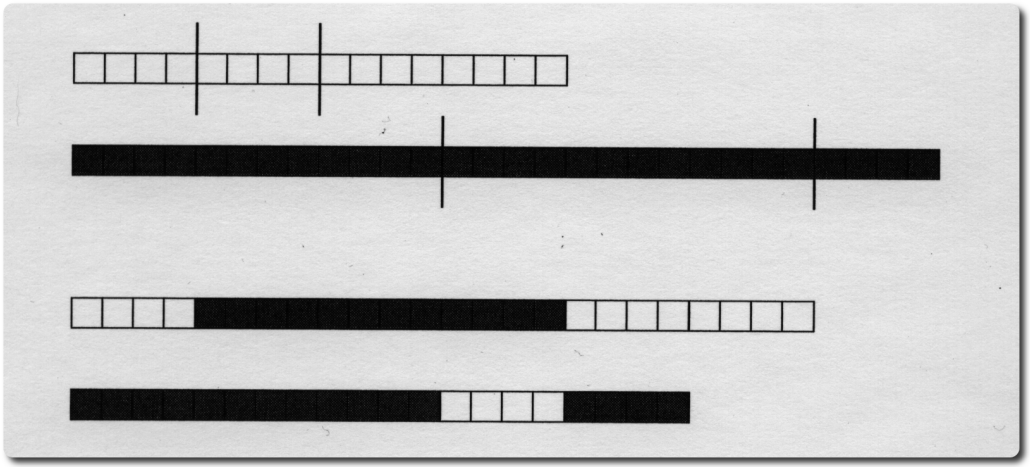
- Before a chromosome can be evaluated, the registers must be **initialized**. Suppose that:
  - Constant registers: $c_1 = 1, c_2 = 3, c_3 = 10$
  - Variable registers: $r_1 = 1, r_2 = r_3 = 0$

Genes	**Instruction**	**Result** (r_1, r_2, r_3)
1 2 1 4	$r_2 := r_1 + c_1$	1, 2, 0
1 3 2 2	$r_3 := r_2 + r_2$	1, 2, 4
3 1 2 3	$r_1 := r_2 \times r_3$	8, 2, 4
5 1 1 5	$\texttt{if}(r_1 > c_2)$	8, 2, 4
1 1 1 4	$r_1 := r_1 + c_1$	9, 2, 4

- Furtermore, suppose that the **input** was provided through register r_1 and that the **output** was taken from r_1 as well: $Output = r_1 = 9$
- However, note that *any* of the registers $r_1 - r_3$ could be utilized as input or output register(s)!

Evolutionary operators in LGP

- It is common to use two-point, length-varying (non-homologous), **crossover** in LGP:



- Apart from being **non-homologous**, crossover in LGP works pretty much in the same way as for the bit strings commonly used in GAs.
- **Mutation** in LGP proceeds as follows: Randomly select an **instruction** for mutation, then with some probability:
- change (i) the **destination** register, (ii) the **operator**, and (iii) the **operands**. That is, within the allowed ranges of course!

Table of Contents

Example 3.11: Function regression

Objective

An "unknown" function $f(x)$ has been measured at $L = 101$ equidistant points in the range $[-2, 2]$. Use LGP to find a function $\hat{f}(x)$ that fits the measured data.

- **Solution**: An LGP system was implemented with the following setup:
- The set of **operators** where taken as $\mathcal{O} = \{o_1, o_2, o_3, o_4\} = \{+, -, \times, /\}$, encoded as 1, 2, 3 and 4.
- The set of **constant registers**, $\mathcal{C} = \{c_1, c_2, c_3\}$, where initialized to $c_1 = 1, c_2 = 3, c_3 = -1$.
- The set of **variable registers**, $\mathcal{R} = \{r_1, r_2, r_3\}$, where initialized to $r_1 = x_i, r_2 = r_3 = 0$. Thus, $\mathcal{R} \cup \mathcal{C} = \mathcal{A} = \{r_1, r_2, r_3, c_1, c_2, c_3\}$.
- The $L = 101$ data points $f(x_i), i = 1, ..., L$ where considered one at a time taking, the **output** $\hat{f}(x_i)$ as the content of r_1 at the end of the evaluation.

Example 3.11: Function regression

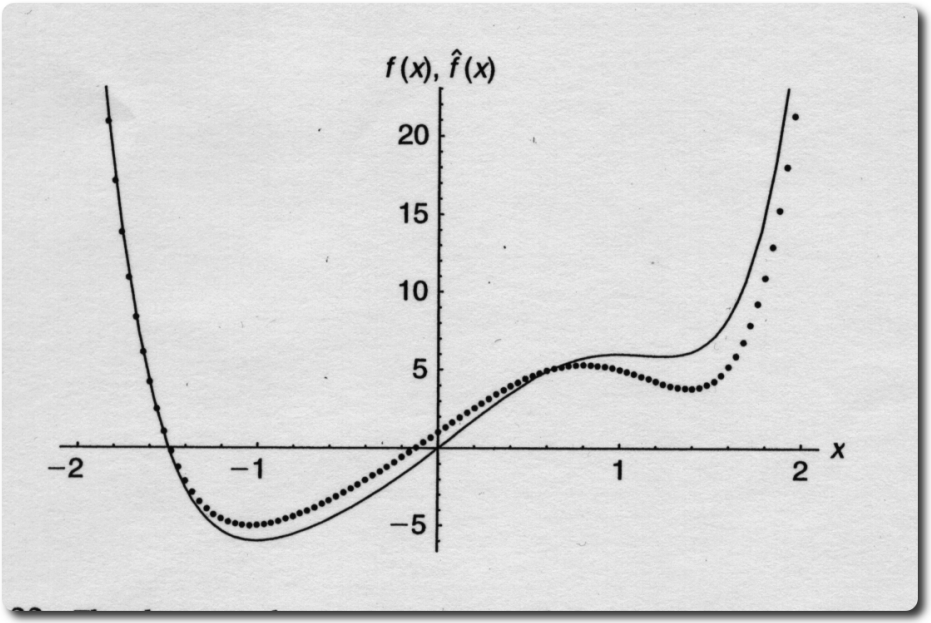
- The **fitness function** was taken as $F = 1/E_{\text{RMS}}$, where E_{RMS} is the **root-mean-square** (RMS) error:

$$E_{\text{RMS}} = \sqrt{\frac{1}{L}\sum_{i=1}^{L}(f(x_i) - \hat{f}(x_i))^2}$$

- The evolutionary parameters:
  - Tournament selection, size 5, $p_{\text{tour}} = 0.75$
  - Two-point non-homologous crossover, $p_{\text{c}} = 0.20$
  - Mutation rate $p_{\text{mut}} = 0.04$.
  - Chromosome lenght initially: $[5, 25]$.
  - Population size: 100 individuals.

Example 3.11: Function regression

- Result of the LGP run, after about 50,000 generations:



- The "unknown" function: $f(x) = 1 + 8x - 3x^3 - 2x^4 + x^6$.
- A typical "best" function found by the LGP system: $\hat{f}(x) = 9x + x^2 - 3x^3 + 2x^4 + x^6$, $E_{\text{RMS}} = 1.723$.
- The chromosome consisted of 13 instructions.

Table of Contents

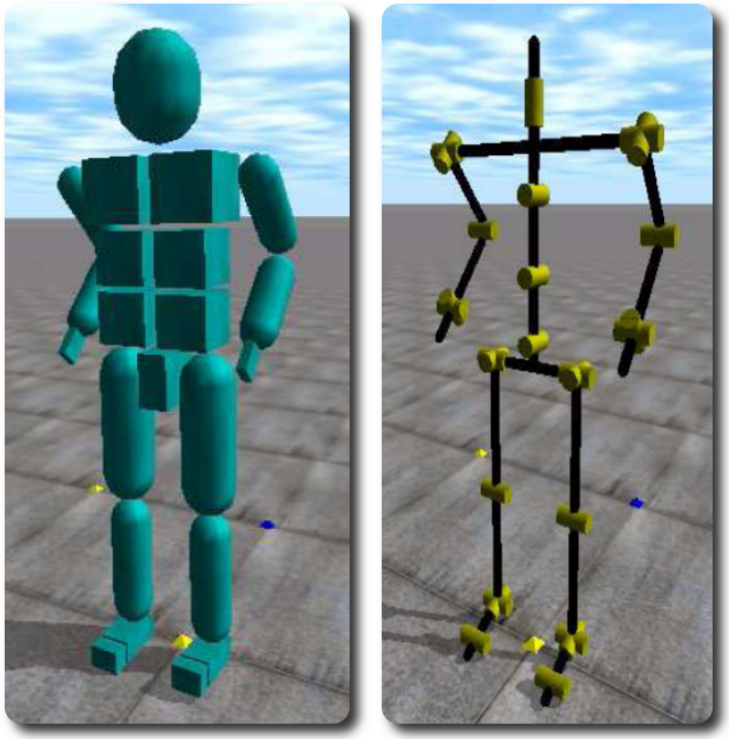
Example: Generation of bipedal robot gaits using LGP

Objective

Evolve bipedal walking for a simulated, high DOF humanoid robot, by means of LGP,

- from first principles,
- with no domain knowledge on walking,
- no information of morphology availible to the LGP system.

The simulated high DOF humanoid robot

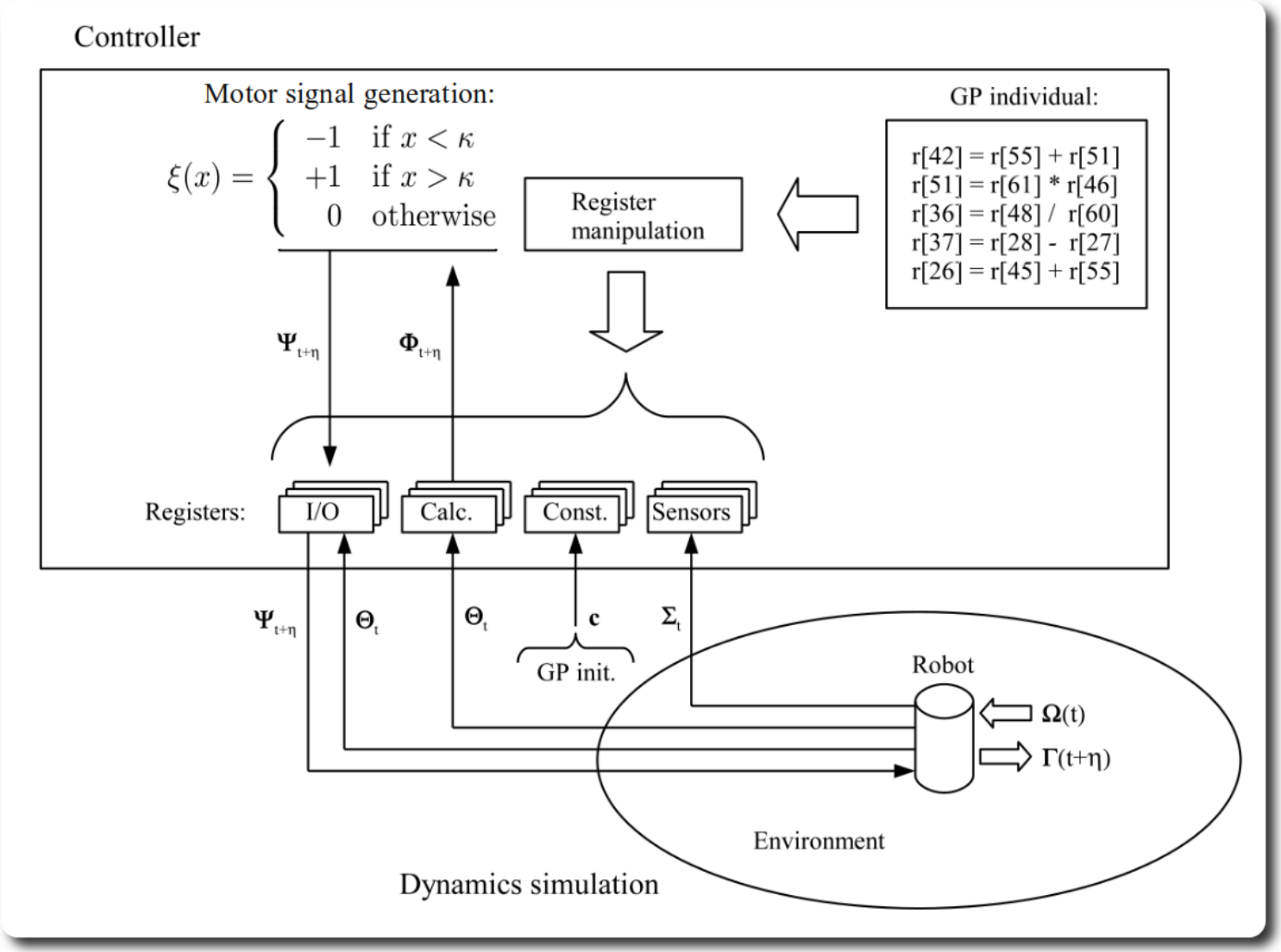


- Generic humanoid robot with 26 DOFs (*cf.* Honda ASIMO).
- Sensor feedback: (i) 1 Inertial measurement unit (IMU), (ii) 2 accelerometers and (iii) 26 joint angle sensors.
- Simulation environment:
  Open dynamics engine (ODE), see `http://www.ode.org/`

LGP setup

- The **operator** set:
  $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7\} = \{+, -, \times, /, \sin, >, \leq\}$, which were encoded as 1, 2, 3, 4, 5, 6 and 7.
- The set of **registers** (variable *and/or* constant):
- $\mathcal{I} = \{r_0, ..., r_{25}\}$ are used for **input**. Initialized with **joint angles**. Constant during execution of en individual.
- $\mathcal{R} = \{r_{26}, ..., r_{51}\}$ are used as **output** registers (variable during execution). Initialized with **joint angles**.
- $\mathcal{C} = \{r_{52}, ..., r_{54}\}$ are used for the **ephemeral constants** (see below).
- $\mathcal{S} = \{r_{55}, ..., r_{66}\}$ are used for the **sensors** (IMU and accelerometers: variable).
- Thus, $\mathcal{A} = \mathcal{I} \cup \mathcal{R} \cup \mathcal{C} \cup \mathcal{S} = \{r_0, ..., r_{66}\}$.
- Note: An **ephemeral constant** is initialized with a *random value* in the beginning of an LGP run.

Controller model



LGP instruction set

- The LGP for bipedal walking used the following **instruction set**:

Instruction type	*General notation*	*Input range*
Arithmetic operations	$r_i := r_j + r_k$ $r_i := r_j - r_k$ $r_i := r_j \times r_k$ $r_i := r_j / r_k$	$r_i, r_j, r_k \in \mathbf{R}$
Trigonometric functions	$r_i := \sin r_j$	$r_i, r_j \in \mathbf{R}$
Conditional branches	$if\,(r_j > r_k)$ $if\,(r_j \leq r_k)$	$r_j, r_k \in \mathbf{R}$

Encoding scheme for walking

- The **operators** where encoded as: $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5, o_6, o_7\} = \{+, -, \times, /, \text{sine}, >, \leq\} = \{1, 2, 3, 4, 5, 6, 7\}$.
- Consider the following LGP **gene**: `55 51 3 42`.
- The encoding follows this scheme:
  - 55: Operand 1 $\in \mathcal{A}$ (any register)
  - 51: Operand 2 $\in \mathcal{A}$ (any register)
  - 3: Operator $\in \mathcal{O}$ (multiplication)
  - 42: Destination $\in \mathcal{R}$ (variable register)
- Thus, the corresponding **instruction** is: $r_{42} := r_{55} \times r_{51}$.

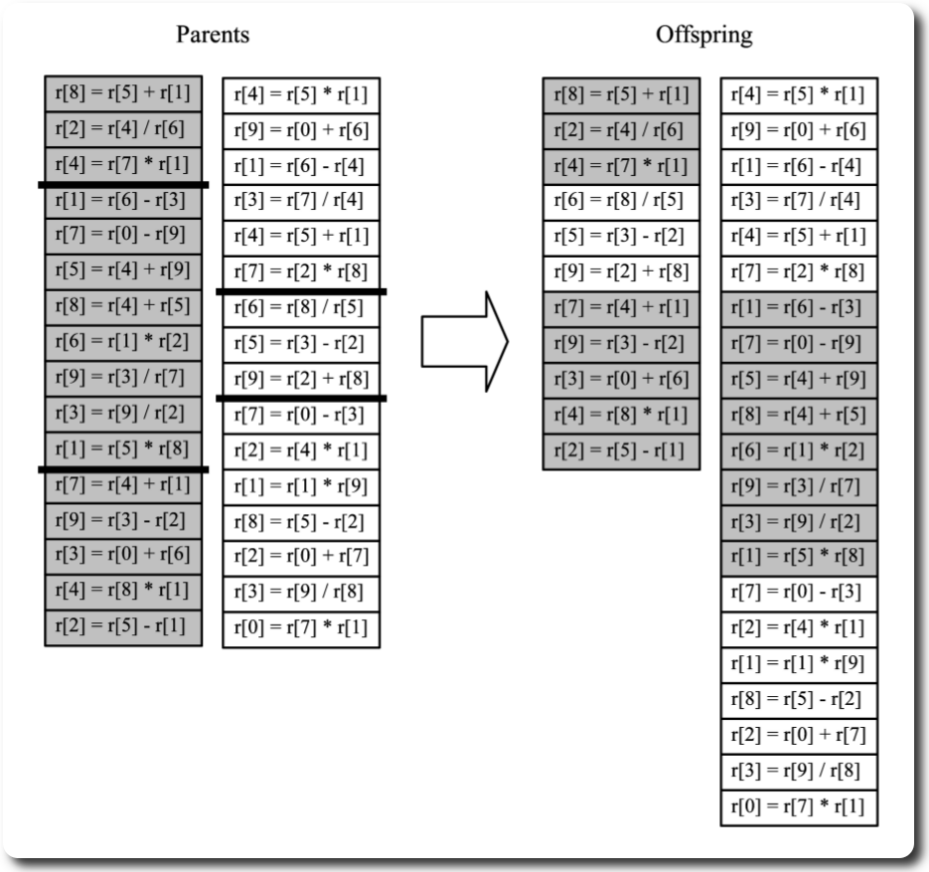
An LGP program: (*individual* and *chromosome*)

```
1:  void GP(double* r){
2:     r[42] = r[55] * r[51];
3:         if(r[18] > r[32]);
4:     r[51] = r[61] * r[46];
5:     r[36] = r[48] - r[60];
6:     r[37] = r[28] * r[27];
7          if(r[15] > r[27]);
8:     r[42] = r[18] + r[45];
9:     r[37] = r[53] / r[36];
10:    r[26] = r[45] + r[55];
11:     return;
12: }
13: int main(void){
14:     double reg[80];
15:     ...
16:     GP(reg);
17:     ...
18:     return 0;
19: }
```

```
55, 51, 3, 42,
18, 32, 6, 41,
61, 46, 3, 51,
48, 60, 2, 36,
28, 27, 3, 37,
15, 27, 6, 33,
18, 45, 1, 42,
53, 36, 4, 37,
45, 55, 1, 26,
```

Genetic operators

- Two-point, non-homologous, **crossover** was used:



- As for the **mutation** operator, an LGP standard procedure was used (described on a previous slide).

LGP parameters

- The parameters of the LGP system are listed in a **Koza tableau**:

Parameter	Value
Objective	Approximate a function that produce robust biped gait
Terminal Set	66 integer registers
Function Set	`add, sub, mul, div, sine, branching`
Raw Fitness	According to Eq. 1, scalar value
Standardized Fitness	Same as Raw Fitness
Population Size	between 8 and 512 individuals
Initialization Method	Random Gaussian distribution
Simulation Time	corresponding to 36 s. of real time
Crossover Probability	Between 0.0 and 1.0%
Mutation Probability	1.0
Initial Program Length	Gaussian distribution, between $\langle 32 \rangle$ and $\langle 512 \rangle$ instructions.
Maximum Program Length	1024 instructions
Maximum tournament number	25000
Selection Scheme	Tournament, size 4
Termination Criteria	Max tournament number

Evolutionary algorithm

- The main steps in the EA can be summarized as follows:
- (1) Randomly **initialize** a population.
- (2) **Evaluate** all individuals according to the fitness measure:
  - (2.1) Execute the individual for N simulation timesteps.
  - (2.2) Compute the fitness value according to the fitness function.
- (3) Perform **tournament selection**.
- (4) Apply genetic operators **crossover** and **mutation** in order to produce two new individuals.
- (5) **Replace** the two losers individuals in the population with the new offspring.
- (6) If **termination criterion** is not met, go to step (2).
- (7) **Stop**. The best individual represents the best solution found.

Objective function

- A simple fitness function turned out to work best:

$$F = \sum_{t=0}^{N} (\mathbf{r}_R + \mathbf{r}_L)\hat{\mathbf{y}}$$

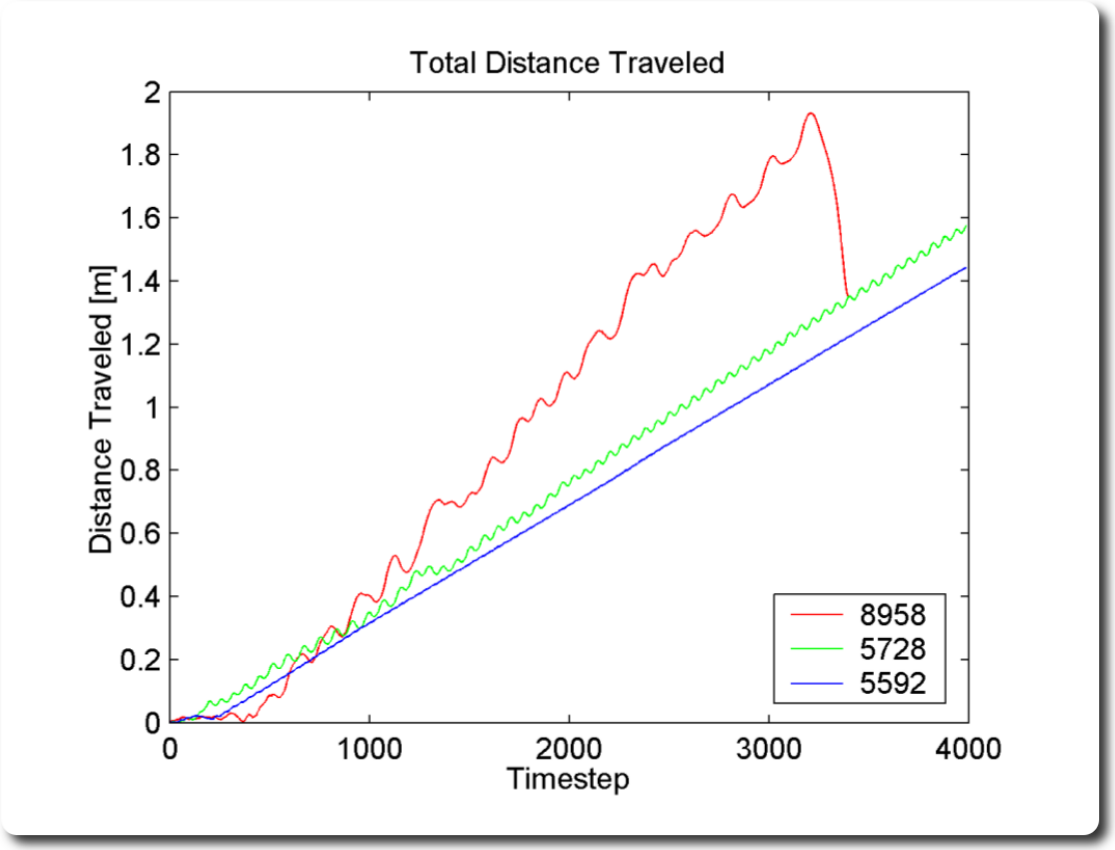
- where r_L and r_R are the position vectors of the robot's feet and $\hat{y}$ is the unit vector in forwards direction.
- Thus, the fitness was (rougly) proportionate to the distance walked by the robot, along a straight line.

Additions

- Energy discharging: Human bipedal walking is very *energy efficient*!
- Termination of slow individuals. If too *slow progress*, terminate!
- Different body proportions. An adult and a child have different *body proportions*.
- Dual controllers (brains). The robot was starting from standstill, thus a *transition phase* from standing to walking was involved!
- Reducing the number of foot steps. *Large* steps were prefered!
- Climbing over obstacles. Force the individuals to take *larger* and *higher* steps!
- Lifting force. Learning to walk without any *supervision* might be too hard!

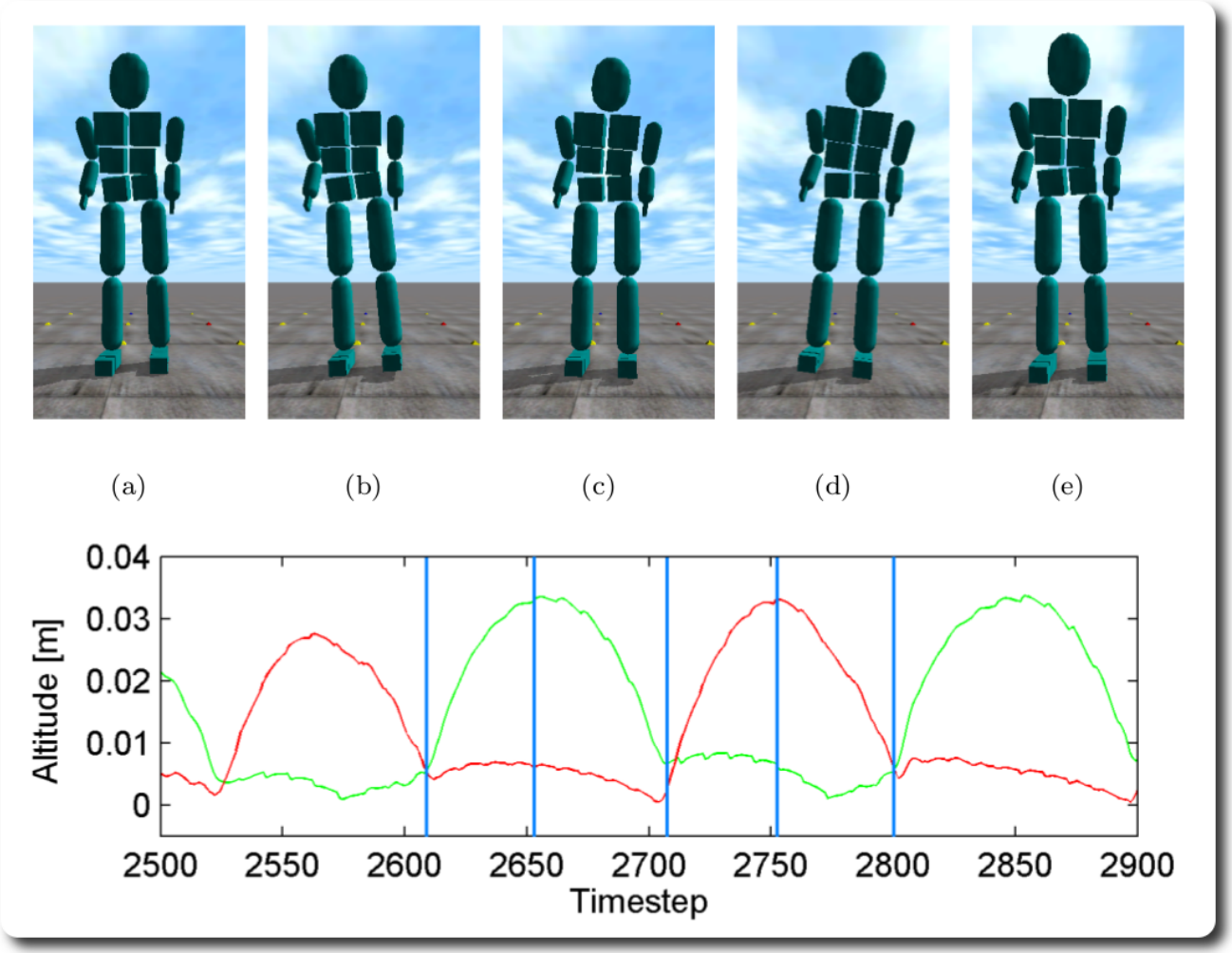
Results

- Individuals evolved that could walk essentially **indefinitely**.



Robot walking

- A gait cycle:



Conlusions from the LGP study

- Unseeded evolution of bipedal gaits was achieved in physical robot simulations (runs lasting about 12 hours of real time).
- Individuals evolved that could walk essentially indefinitely.
- Energy discharging was the only addition that made a significant improvement.
- The dual controller approach made the resulting gaits look more realistic.
- However, the (best) evolved gaits were still not very humanlike, since the incentive (for the EA) to generate such gaits was insufficient.

Table of Contents

Sources

- This lecture was based on the following material:
  (1) Course book by **Wahde, M.**, pp. 72-78.
  (2) Paper on LGP, by **Wolff, K.**, *Evolution of biped locomotion using linear genetic programming*. Will be availible on the course web page shortly.