Gene Expression Programming & Cartesian Genetic Programming

Jiří Kubalík
Department of Cybernetics, CTU Prague

Substantial part of this material is based on the article
Candida Ferreira: Gene Expression Programming: A New Adaptive Algorithm for Solving,
see http://arxiv.org/ftp/cs/papers/0102/0102027.pdf
and slides for tutorial 'Cartesian Genetic Programming'
presented at GECCO 2008 by J.F. Miller and S.L. Harding,
see http://portal.acm.org/citation.cfm?id=1389075

http://cw.felk.cvut.cz/doku.php/courses/a0m33eoa/start

Contents

Gene Expression Programming

Gene Expression Programming (GEP) - genotype/phenotype genetic algorithm for creation of computer programs

- GEP uses **fixed length linear chromosomes** of specific structural organization of genes.
- The chromosomes are subjected to variation operators.
- The **linear chromosomes are expressed as nonlinear expression trees** (ETs) of different sizes and shapes that are evaluated an upon which the selection acts.

  Any modification made in the genome always results in syntactically correct ETs (given that the closure property holds).
- GEP provides means for automatic defining and reusing functions.

GEP recalls to its analogy to the natural **gene expression**:

Gene expression is the process by which information from a gene is used in the synthesis of a functional gene product. These products are often proteins, but in non-protein coding genes such as rRNA genes or tRNA genes, the product is a functional RNA.

Wikipedia

GEP: Representation

A chromosome consists of a **linear string of fixed length** composed of one or more genes.

Within each **gene**, a coding sequences of symbols (**open reading frame - ORF**) can be followed by noncoding region.

ORFs are K-expressions that are translated into ETs using a breadth-first parsing:

input: K - expression (from Karva language)
output: ET – expression tree
o – current open node, s – currently processed symbol of K

1. Read the first symbol s of K
2. Make an empty root node of the ET
3. $o \leftarrow root$
4. Attach the corresponding function/terminal to the node o
5. and create bellow the node as many empty children nodes as there are arguments to that function.
6. If the ET is incomplete then end.
8. Else
9. Read next symbol s from K
10. $o \leftarrow left\_upper\_open\_node(ET)$
11. Goto line 4.

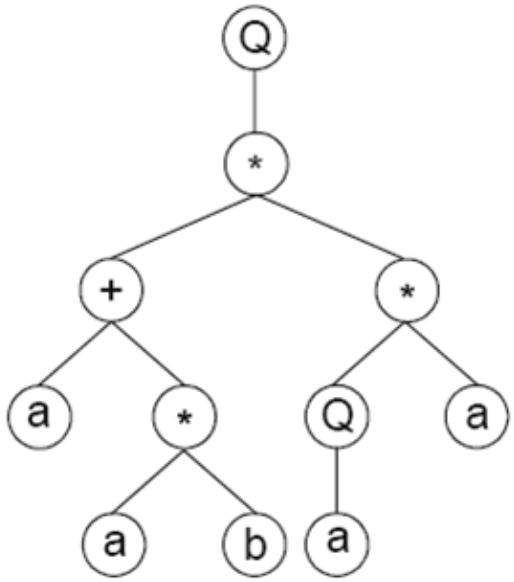
Translation: Example

Assume K-expression

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
Q * + * a * Q a a b a b b a a b
```

where `Q` denotes the square root function, and `a` and `b` are variables.

Then the following ET is



GEP: Structural Organization of Genes

Organization of genes

- **Head** – contains both the function and terminal symbols.

  The head length, h, is chosen by the user for each problem.

- **Tail** – contains only terminal symbols.

  The length of the tail t is a function of h and the number of arguments of the function with the biggest arity.

  The tail must be long enough to ensure that there are sufficient number of terminals for all arguments induced by functions present in the head part.

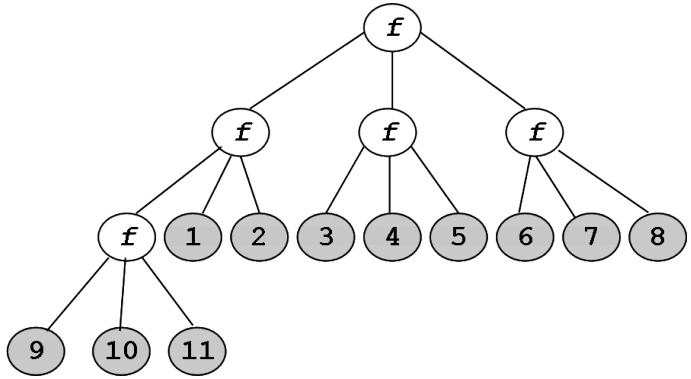
GEP: Size of the Tail Region

Consider

- Head length $h = 5$,
- The maximum arity of functions $n = 3$.

and assume that all of the symbols in the head part represent functions with arity $n = 3$.

What is the number of open nodes that must be filled in by terminal symbols?



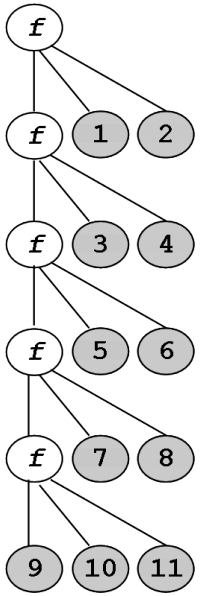
GEP: Size of the Tail Region

Consider

- Head length $h = 5$,
- The maximum arity of functions $n = 3$.

and assume that all of the symbols in the head part represent functions with arity $n = 3$.

What is the number of open nodes that must be filled in by terminal symbols?



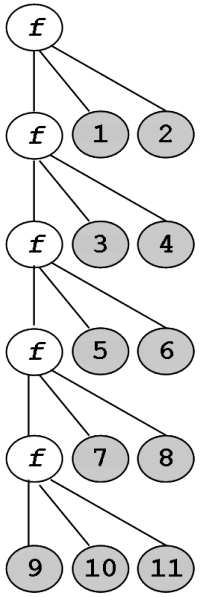
GEP: Size of the Tail Region

Consider

- Head length $h = 5$,
- The maximum arity of functions $n = 3$.

and assume that all of the symbols in the head part represent functions with arity $n = 3$.

What is the number of open nodes that must be filled in by terminal symbols?



The size of the tail t must be

$$t = h(n - 1) + 1$$

in order to ensure a sufficient number of terminal symbols even for the worst case scenario with only function symbols in the head part.

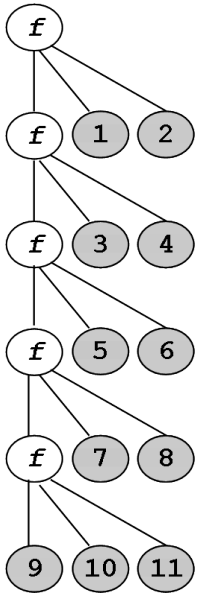
GEP: Size of the Tail Region

Consider

- Head length $h = 5$,
- The maximum arity of functions $n = 3$.

and assume that all of the symbols in the head part represent functions with arity $n = 3$.

What is the number of open nodes that must be filled in by terminal symbols?



The size of the tail t must be

$$t = h(n - 1) + 1$$

in order to ensure a sufficient number of terminal symbols even for the worst case scenario with only function symbols in the head part.

Note, the **length of the ORFs varies**, not the length of genes.

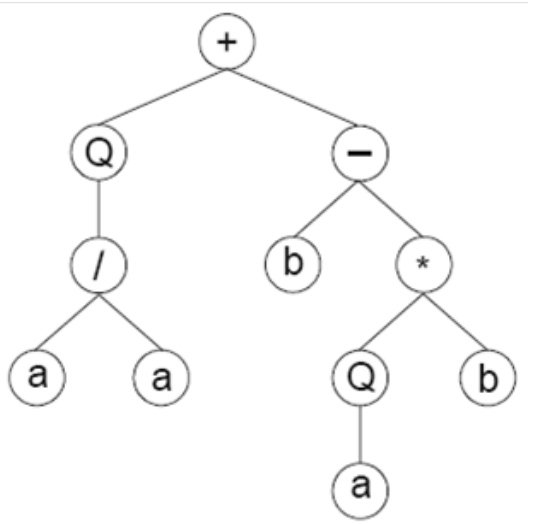
- the noncoding regions in genes allow modifications of the genome always producing syntactically correct programs.

GEP: Effect of Variable ORF Size

Consider $T = \{\mathtt{a}, \mathtt{b}\}$, $F = \{\mathtt{Q}, +, -, *, /\}$ and a gene (the tail is shown in red):

```
0 1 2 3 4 5 6 7 8 9 0 | 1 2 3 4 5 6 7 8 9 0
+ Q - / b * a a Q b a | a b a a b b a a a b
```

that codes for the following ET:



In this case the ORF ends at position 10.

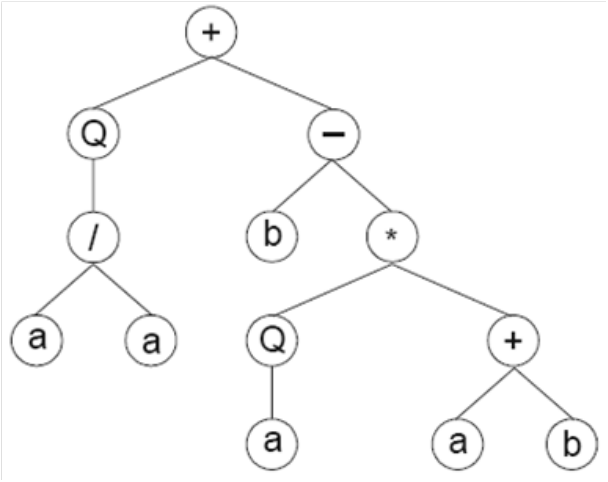
GEP: Effect of Variable ORF Size

Suppose now the symbol at position 9 changed from 'b' into '+'.

The following gene will be:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 | 3 4 5 6 7 8 9 0
+ Q - / b * a a Q + a a b | a a b b a a a b
```

that codes for the following ET:



In this case the ORF ends at position 12.

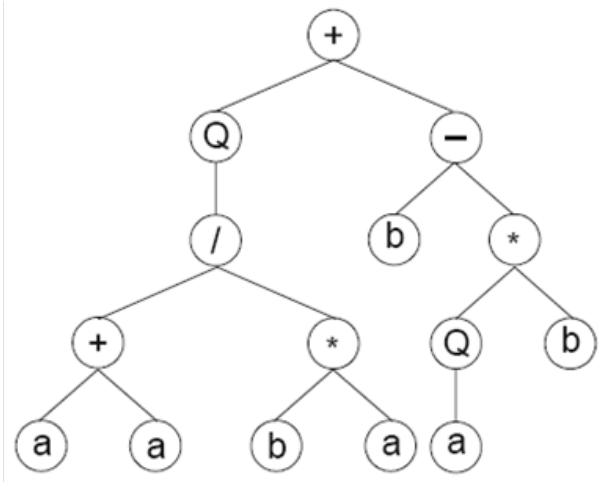
GEP: Effect of Variable ORF Size

Suppose now the symbols at positions 6 and 7 (in the original gene) changed into '+' and '*'.

The following gene will be:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 | 5 6 7 8 9 0
+ Q - / b * + * Q b a a b a a | b b a a a b
```

that codes for the following ET:



In this case the ORF ends at position 14.

Despite its fixed length, each gene can code for ETs of different sizes and shapes!

GEP: Multigenic Chromosomes

Chromosomes are usually composed of more than one gene of equal length – the number of genes is the control parameter defined by a user.

Each gene codes for a sub-ET

1. The sub-ETs can interact with one another and form a more complex ET.
   (a) The sub-ETs are linked together by a **linking function**.
   The linking function can be either defined for a problem at hand or can be evolved along with the genes.
   (b) The sub-ETs are linked together by means of so-called **homeotic genes** – genes representing the "main program".
   The homeotic genes are evolved to determine which sub-ETs are called upon and how the sub-ETs interact with one another (the concept similar to ADFs in GP).
2. Each sub-ET can define one individual output in multi-output problems (for example, each sub-ET can be responsible for an identification to a particular class in classification problems).

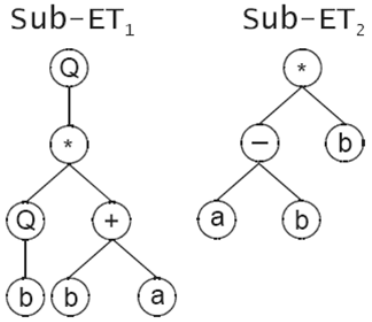
Multigenic Chromosomes: Linking Functions

Using an addition linking function '+' for algebraic sub-ETs.

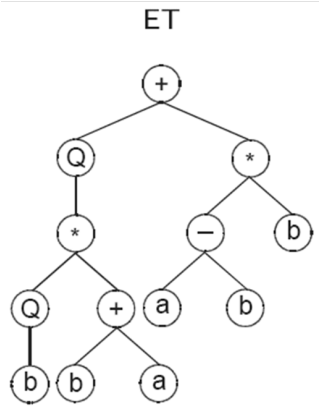
A two-genic chromosome:

```
0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8
Q * Q + b b a a a * - b a b a a b b
```

expresses as two sub-ETs



that after linking by '+' function result in the final ET



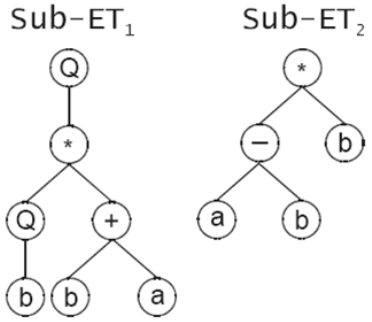
Multigenic Chromosomes: Linking Functions

Using an addition linking function '+' for algebraic sub-ETs.

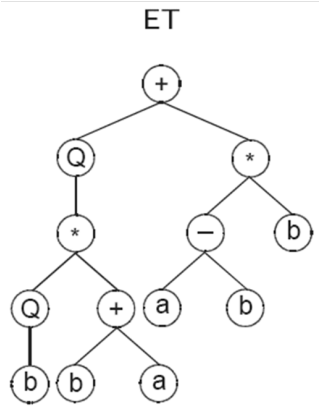
A two-genic chromosome:

```
0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8
Q * Q + b b a a a * - b a b a a b b
```

expresses as two sub-ETs



that after linking by '+' function result in the final ET



Note that the final ET could be encoded by a single-genic chromosome

```
0 1 2 3 4 5 6 7 8 9 0 1 2
+ Q * * - b Q + a b b b a
```

however, multigenic chromosomes, allowing for modular construction of complex, hierarchical structures are more efficient.

Multigenic Chromosomes: Homeotic Genes

A chromosome is composed of

- one or more conventional genes – that act as ADFs,
- plus so called **homeotic gene** – the expression of such genes results in the main program that determines which genes are expressed and how the corresponding sub-ETs (ADFs) interact with each another.

Homeotic genes have exactly the same structural organization as conventional genes – they have specific length and specific sets of functions and terminals.

- **Head** – contains **linking functions** and so-called **genic terminals** representing conventional genes.
- **Tail** – contains only **genic terminals**.

The concept using homeotic genes allow for encoding the ADFs that can be further called many times from many different places in the "main program".

Contrary to GP, the ADFs in GEP do not allow formal parameters to be defined.

Multigenic Chromosomes: Homeotic Genes

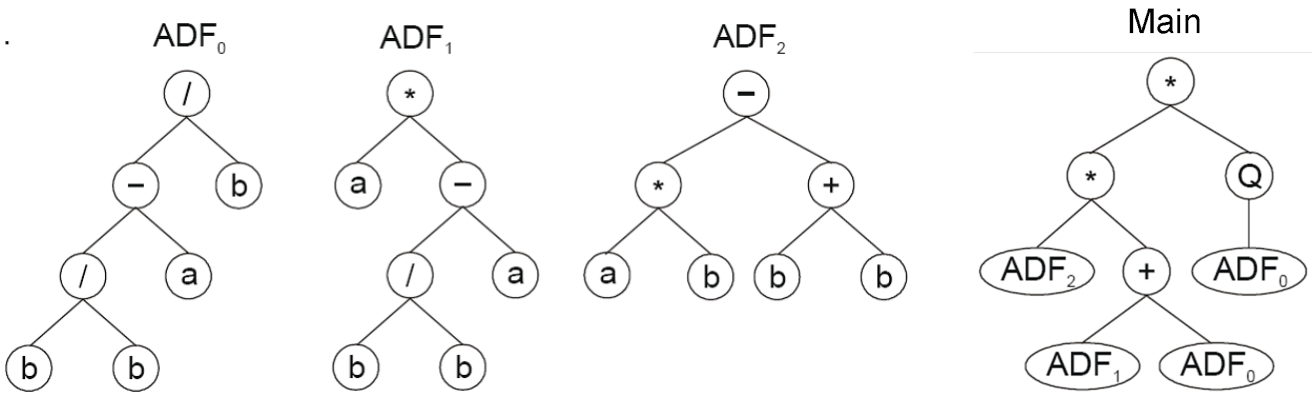
Example:

- the head length of the homeotic gene $h_H = 5$
- the head length of the conventional genes $h = 4$
- $F_H = \{+, -.*, /, Q\}$, $T_H = \{1, 2, 3\}$ denoting ADF_1, ADF_2 and ADF_3
- $F = \{+, -, *, /\}$ and $T = \{a, b\}$

The following chromosome codes for three conventional genes and one homeotic gene (red)

```
012345678012345678012345678012345678901234567890
/-b/abbaa*a-/abbab-*+abbbaa**Q2+010102
```

that express as



GEP: Random Numerical Constants

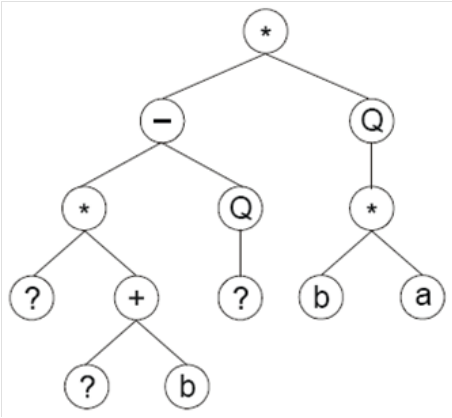
Handling **random numerical constants** in GEP:

- **Extra terminal '?'** – represents the random constant (similar to the *ephemeral random constant* in GP).
- **Extra domain Dc** – comes after the tail, has a length equal to t and is composed of symbols chosen to represent the random constants.
- For each gene the initial random constants are generated during the creation of the initial population and kept in an array. Specific operators to introduce genetic variation in the pool of random constants and for effective circulation of the random constants in the population.

Example: The following single-gene chromosome with $h = 8$ (the Dc is shown in red)

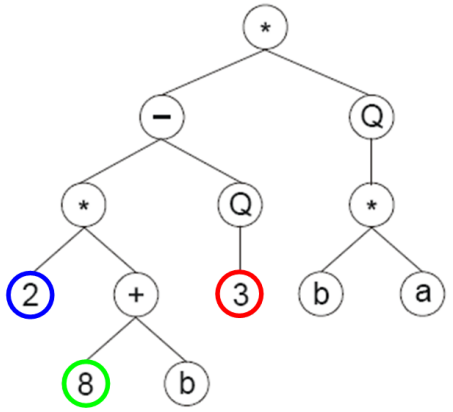
```
012345678901234567890123 45
*-Q*Q*?+?ba?babba238024198
```

expresses as



GEP: Random Numerical Constants

Then the ?'s in the ET are replaced from left to right and from top to bottom by the symbols in Dc

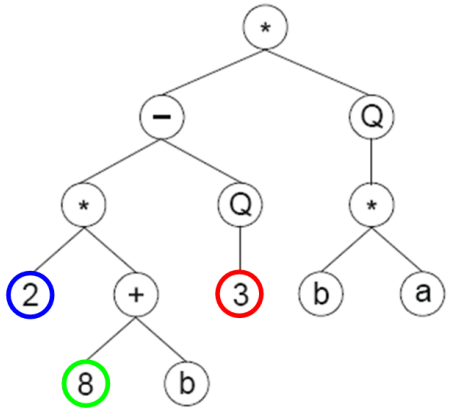


The random constants corresponding to these symbols are kept in an array; the number represented by the numeral indicates the order in the array

$$A = \{1.095, 1.816, 2.399, 1.618, 0.725, 1.997, 0.094, 2.998, 2.826, 2.057\}.$$

GEP: Random Numerical Constants

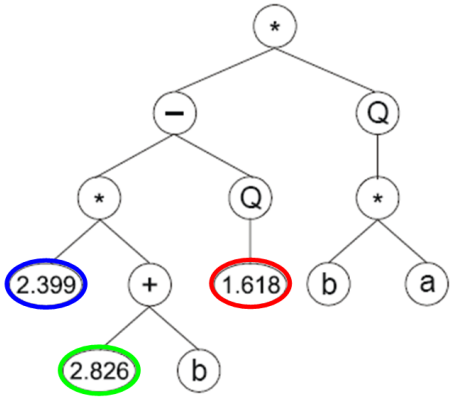
Then the ?'s in the ET are replaced from left to right and from top to bottom by the symbols in Dc



The random constants corresponding to these symbols are kept in an array; the number represented by the numeral indicates the order in the array

$$A = \{1.095, 1.816, 2.399, 1.618, 0.725, 1.997, 0.094, 2.998, 2.826, 2.057\}.$$

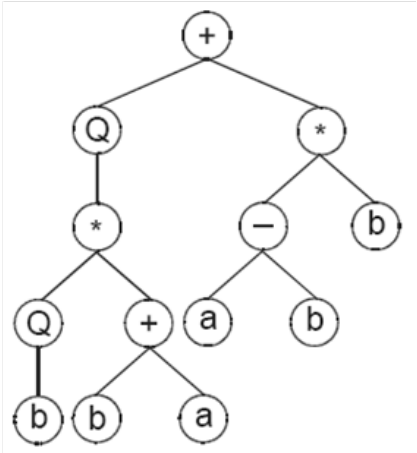
Finally, after substituting numerals for the actual values we get



GEP Genetic Operators: Mutation

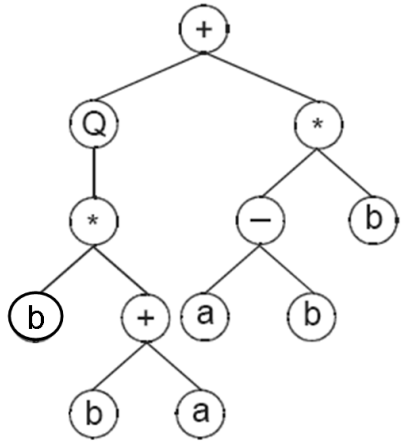
Mutation – changes a symbol anywhere in the chromosome.

- The desired structural organization of chromosomes must be ensured.
  In the heads, any symbol can change to another (function or terminal).
  In the tails, only terminals can be introduced.
- Neutral mutations are possible – when it occurs in the noncoding region.



012345678012345678
Q*Q+bbaaa*-ababaabb

A mutation occurred at position 2 in gene 1 and at position 6 in gene 2.



012345678012345678
Q*b+bbaaa*-ababbbb

GEP Genetic Operators: Transposition

Transposition – certain fragments of the genome can be copied/moved to another place in the chromosome.

Three types of transposable elements

- **Insertion sequence elements** (IS) – short fragments with a function or terminal in the first position that transpose to the head of genes (except to the root).

  This transposition can drastically reshape the ET (the closer to the gene beginning the insertion site is the more profound the change).

- **Root IS elements** – short fragments with a function in the first position that transpose to the root of genes.

  Modifications introduced by this transposition are extremely radical – useful for creating genetic variation.

- **Entire genes** – an entire gene transposes itself to the beginning of the chromosome and is deleted in the place of origin.

  Useful only in situations where the linking function is not commutative.

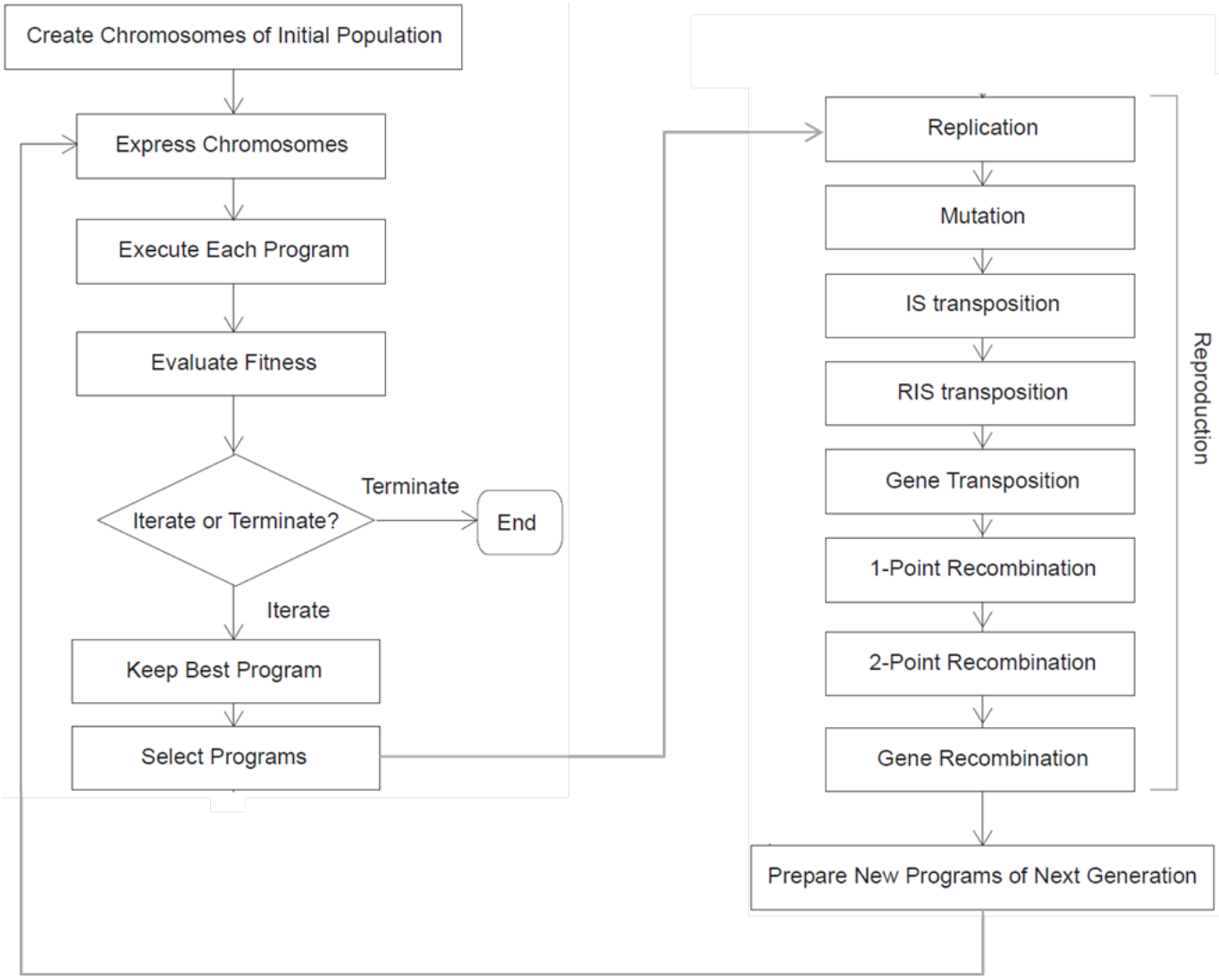
In all types of transposition the structural organization of chromosomes is maintained.

GEP Genetic Operators: Recombination

Three type of recombination:

- 1-point recombination, 2-point recombination
- Gene recombination

GEP: Flowchart



GEP on Symbolic Regression

Target function: $y = a^4 + a^3 + a^2 + a$

GEP

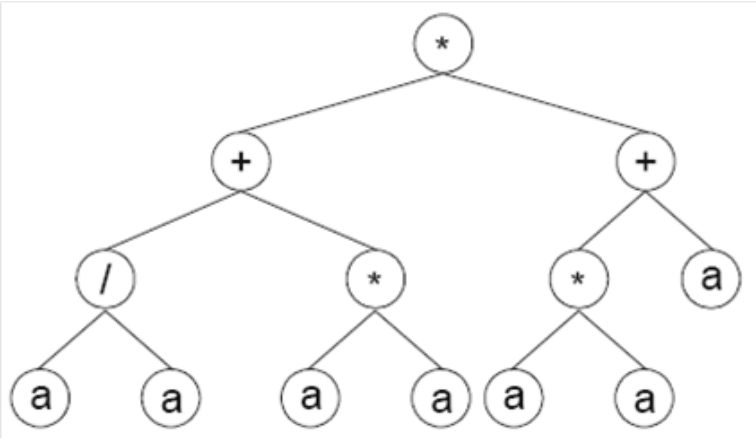
- $F = \{+, -, *, /\}$
- $T = \{a\}$

Example solution found with $h = 6$

A single-genic chromosome

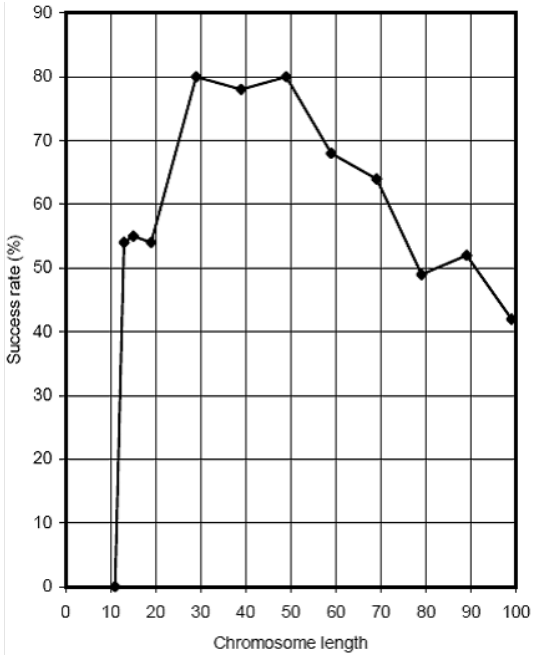
```
0123456789012
***+/**aaaaaaa
```

expresses as



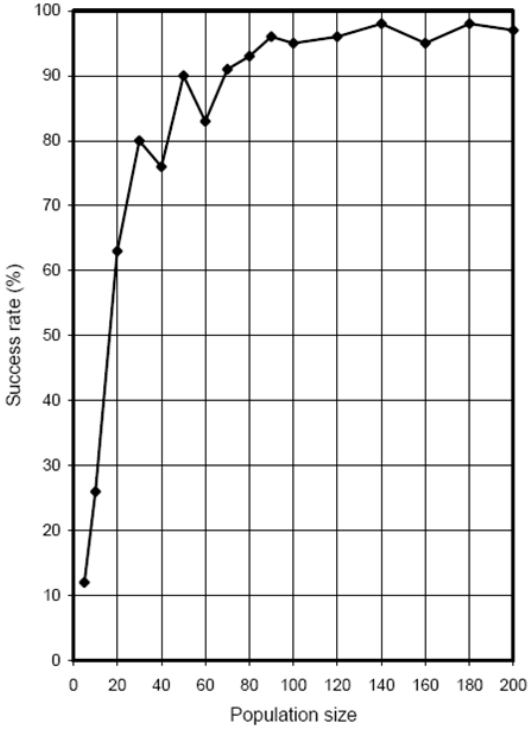
GEP on Symbolic Regression

Chromosome length analysis



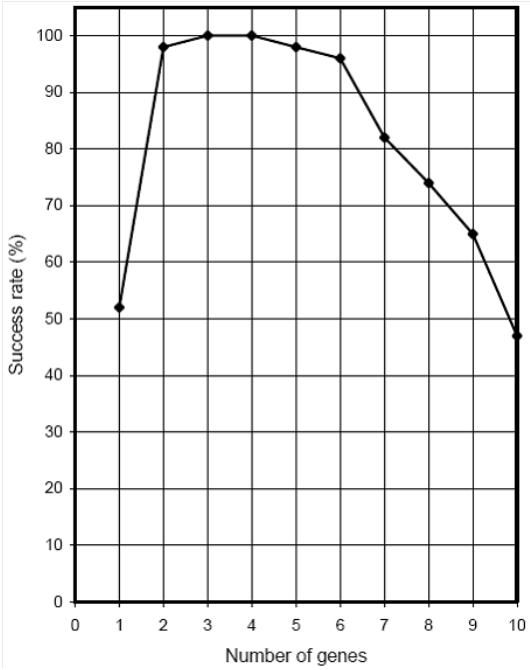
$G = 50$, $P = 30$

Population size analysis



$G = 50$, a medium value of $h = 24$

Number of genes analysis



$h = 6$

GEP: Summary

Application areas

- Symbolic regression
- Classification problems
- Logic synthesis
- Design of neural networks

Pros/cons:

- (+) Structural organization of genotypes provide provides an efficient way of encoding syntactically correct programs.
- (+) Efficient means for preventing bloat.
- (-) ADFs can only be reused with the same inputs.
- (-) It is not possible to evolve programs with function nodes of different output type.
- (-) Requires proper setting of control parameters – h and the number of genes.

GEP: Sources

- Gene Expression Programming website (Candida Ferreira, Gepsoft Limited):

  `http://www.gepsoft.com/`

- Candida Ferreira: Gene Expression Programming: A New Adaptive Algorithm for Solving, Complex Systems, Vol. 13, issue 2: 87-129, 2001

  `http://arxiv.org/ftp/cs/papers/0102/0102027.pdf`

Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) is a GP technique that, in its classic form, uses a very simple integer based genetic representation of a program in the form of a directed graph.

- The genotype is a list of integers that represent the program primitives and how they are connected together.

  The genotype usually contains many non-coding genes.

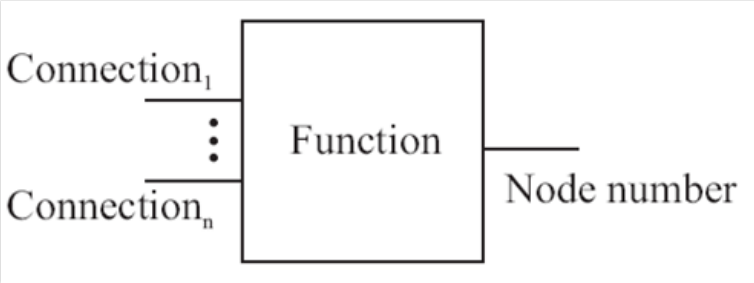
- The genes are
  - Addresses in data (connection genes)
  - Addresses in a look up table of functions
- The representation is very simple, flexible and convenient for many problems.

CGP Node

CGP program is a set of interconnected nodes. A CGP node contains

- a function symbol – specifies the operation performed by the node,
- connections – pointers toward nodes providing input for the function of the node.

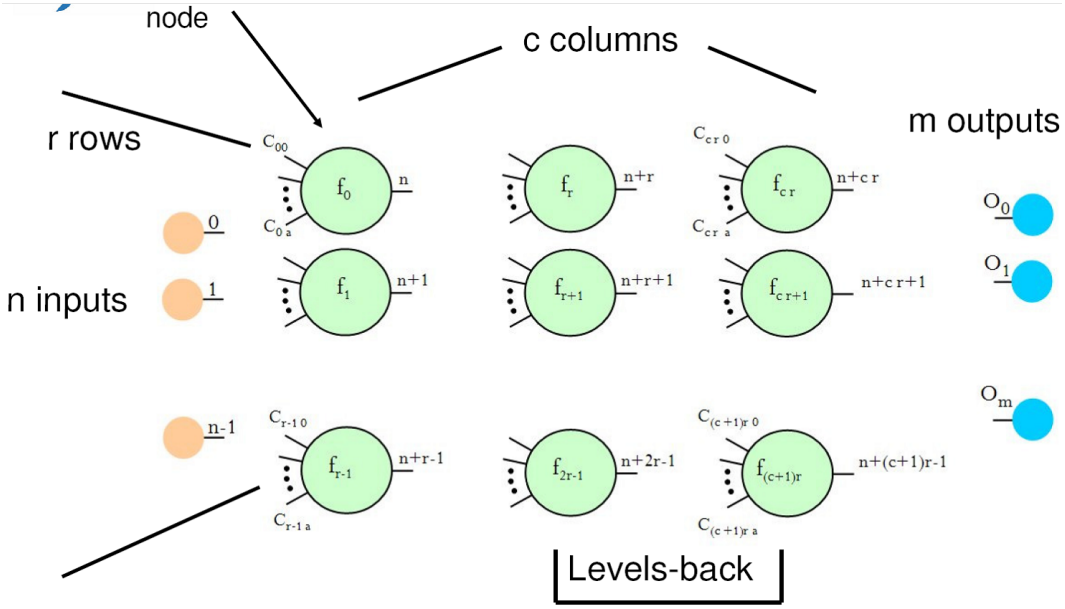
Each CGP node has an output that may be used as an input for another node.



CGP node

CGP General Form

CGP is Cartesian in the sense that the graph nodes are represented in Cartesian coord. system



Each CGP program is defined by

- number of rows r,
- number of columns c,
- number of inputs n,
- number of outputs m,
- number of functions f,
- nodes interconnectivity l

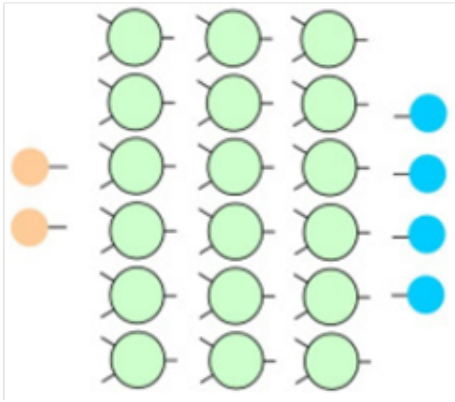
Nodes in the same column are not allowed to be connected to each other.

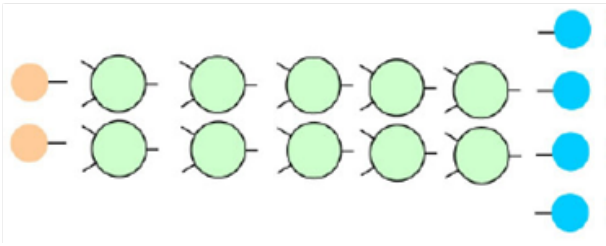
The nodes interconnectivity defines the maximum distance (in terms of the number of columns) between two connected nodes.

- If equal to 1, each node can be connected only with nodes in the previous column.
- If equal to c, each node can be connected to any other node in the previous columns.

CGP: Variety of Graphs

Depending on r, c and l a wide range of graphs can be generated

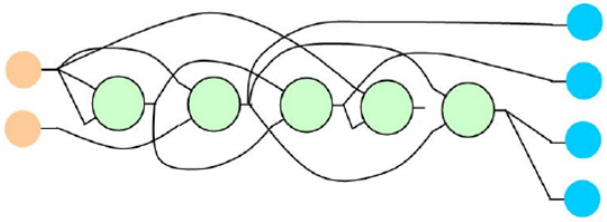




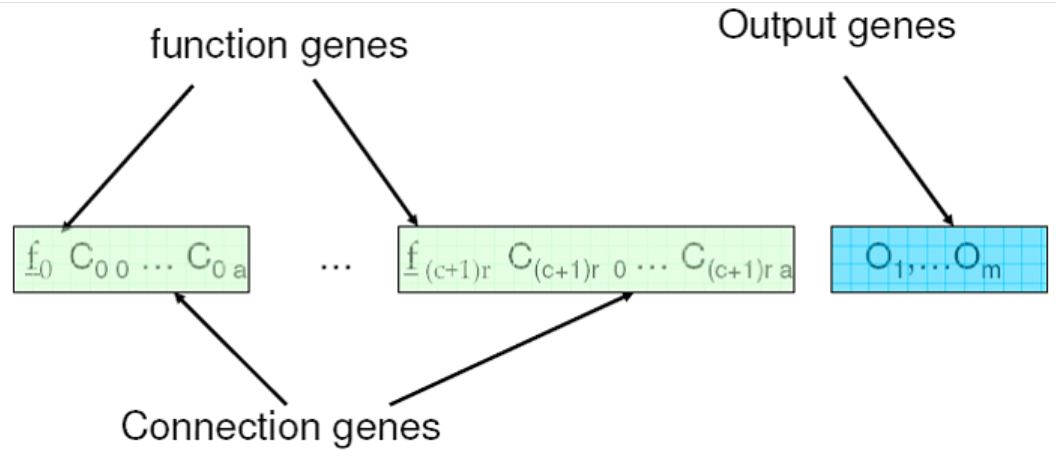
The length of the genotype (i.e. the maximum size of the CGP program) is fixed, however the **actual size and structure of the program can vary**.

The most general choice is $r = 1$ and $l = c$

- Arbitrary directed graphs can be created with a maximum depth.
- Suitable when no prior knowledge about the solution is available.



CGP Genotype

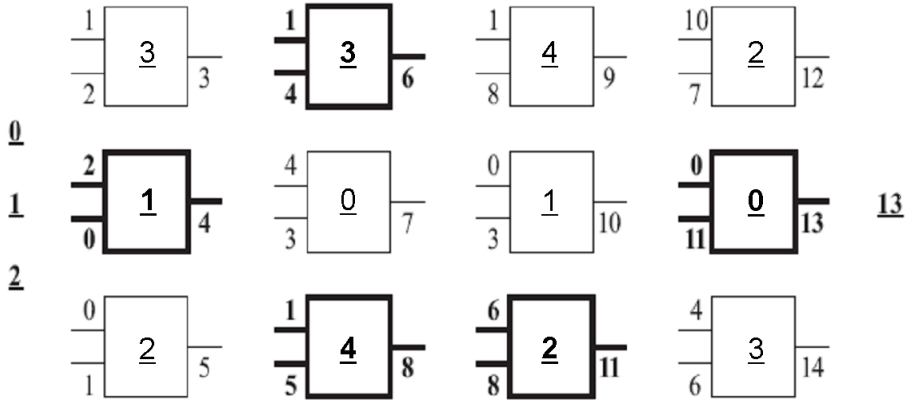


Usually, all functions have as many inputs as the maximum function arity.

Unused connections are ignored.

CGP Program Example

CGP program with 3×4 architecture, 3 inputs and 1 output.



Look up table of 5 functions

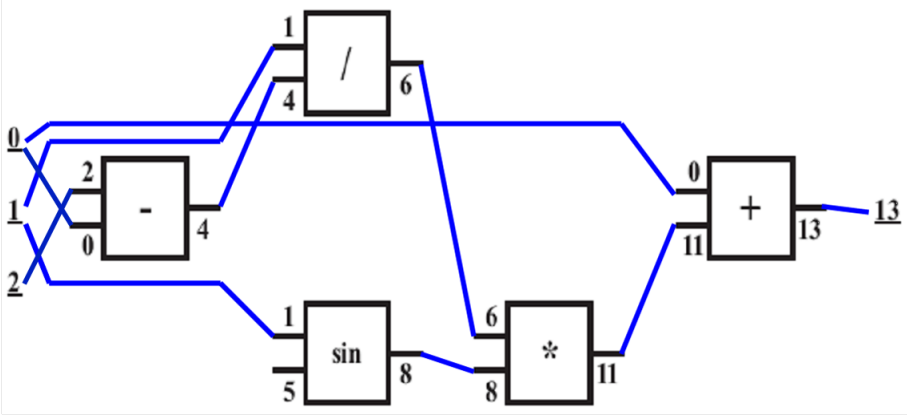
0	+	Add the arg1 to arg2
1	-	Subtract arg2 from arg1
2	*	Multiply arg1 to arg2
3	/	Divide arg1 by arg2
4	`sin`	Calculate sin of arg1

CGP chromosome

C=(3,1,2, **1,2,0**, 2,0,1, **3,1,4**, 0,4,3, **4,1,5**, 4,1,8, 1,0,3, **2,6,8**, 2,10,7, **0,0,11**, 3,4,6, 13)

CGP Program Example

CGP program with 3×4 architecture, 3 inputs and 1 output.



Look up table of 5 functions

0	+	Add the arg1 to arg2
1	-	Subtract arg2 from arg1
2	*	Multiply arg1 to arg2
3	/	Divide arg1 by arg2
4	`sin`	Calculate sin of arg1

CGP chromosome

C=(3,1,2, **1,2,0**, 2,0,1, **3,1,4**, 0,4,3, **4,1,5**, 4,1,8, 1,0,3, **2,6,8**, 2,10,7, **0,0,11**, 3,4,6, 13)

Graph function: $y = x_0 + (x_1/(x_2 - x_0)) * \sin x_1$

CGP: Algoritmus

In its classic form, CGP uses a variant of a simple algorithm called $(1 + \lambda)$-Evolutionary Strategy with a point mutation variation operator, where λ is usually 4.

$(1 + \lambda)$-ES:

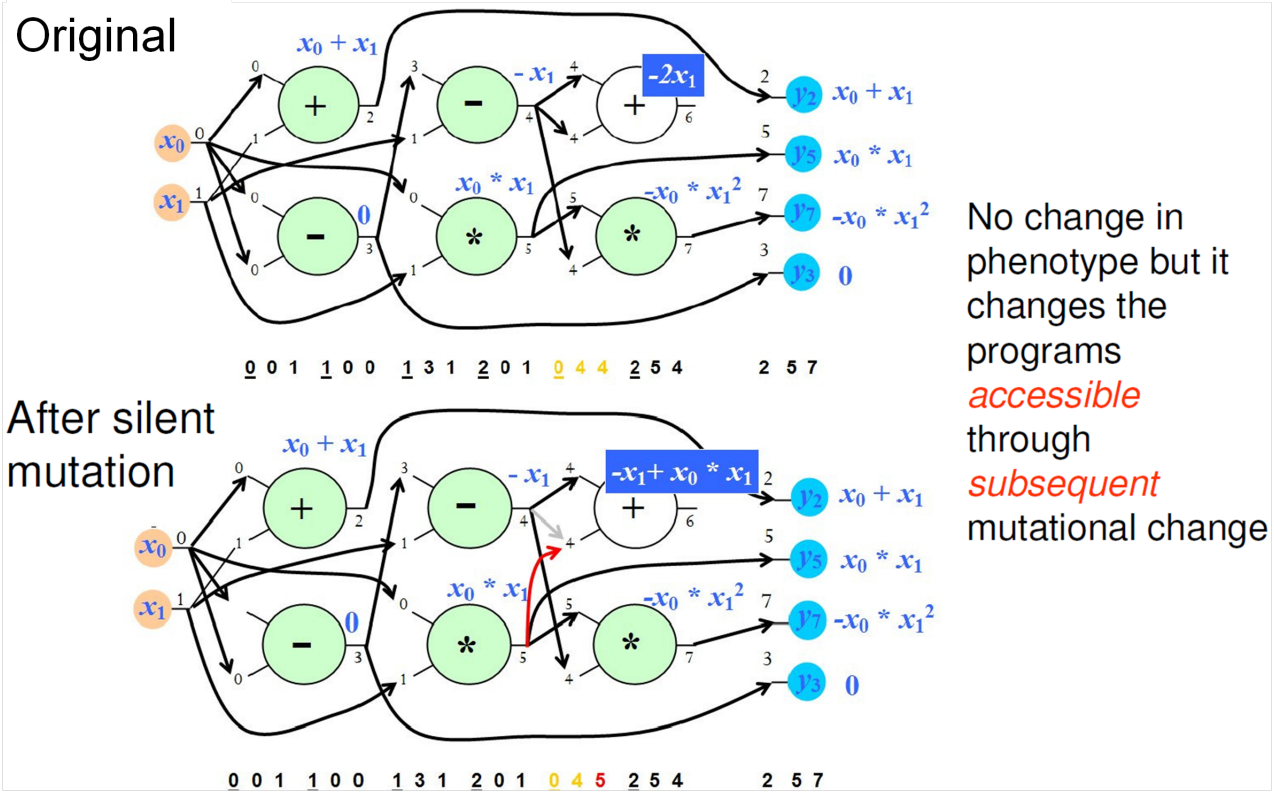
1. Generate a random solution S
2. while not stopping criterion do
3. Generate λ mutated versions of S
4. Replace S by the best individual individual out of the λ new solutions iff it is not worse than
5. Return S as the best solution found

Neutral search – in step 4 we accept move to new states of the solution space that do not necessarily improve the quality of the current solution.

If only improving steps are allowed then the search would be non-neutral.

CGP: Point Mutation

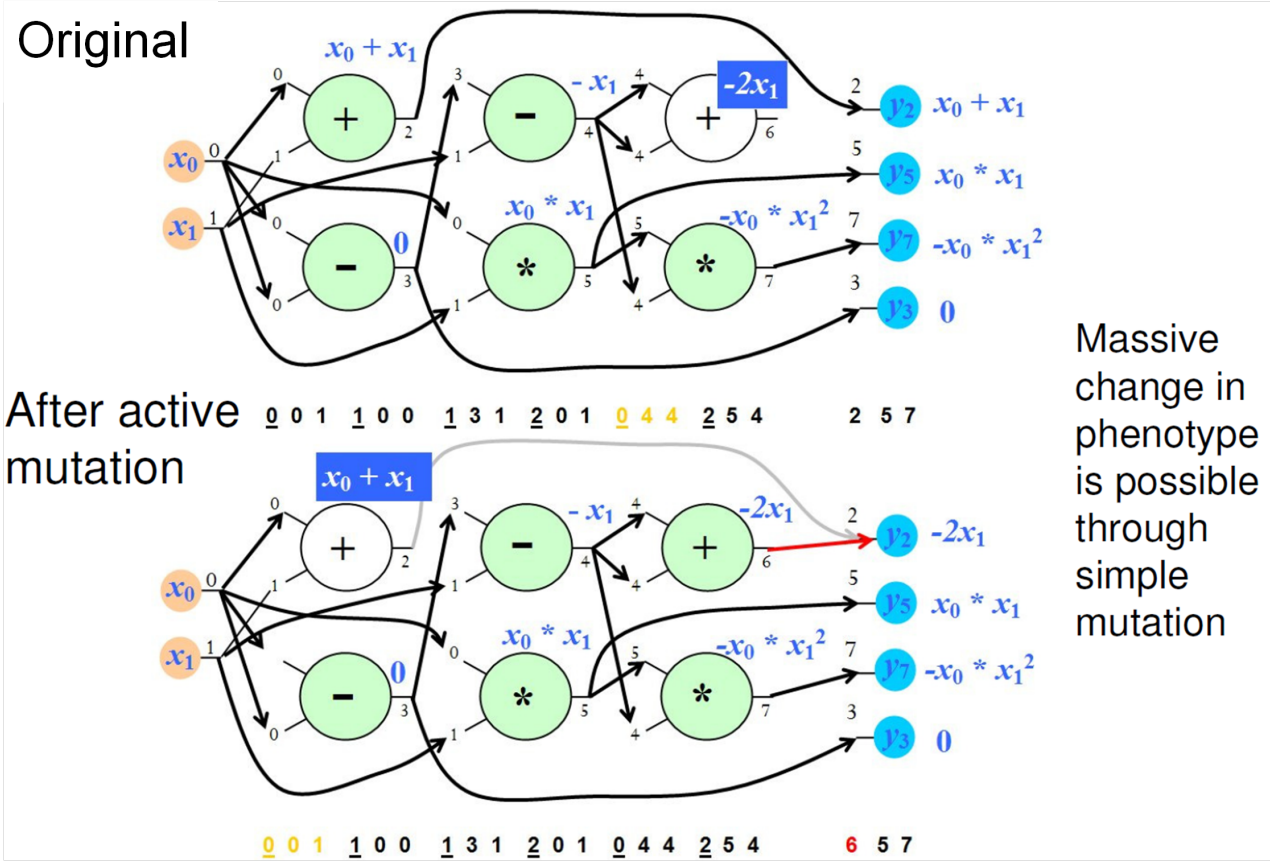
Silent mutations and their effects



No change in phenotype but it changes the programs *accessible* through *subsequent* mutational change

CGP: Point Mutation

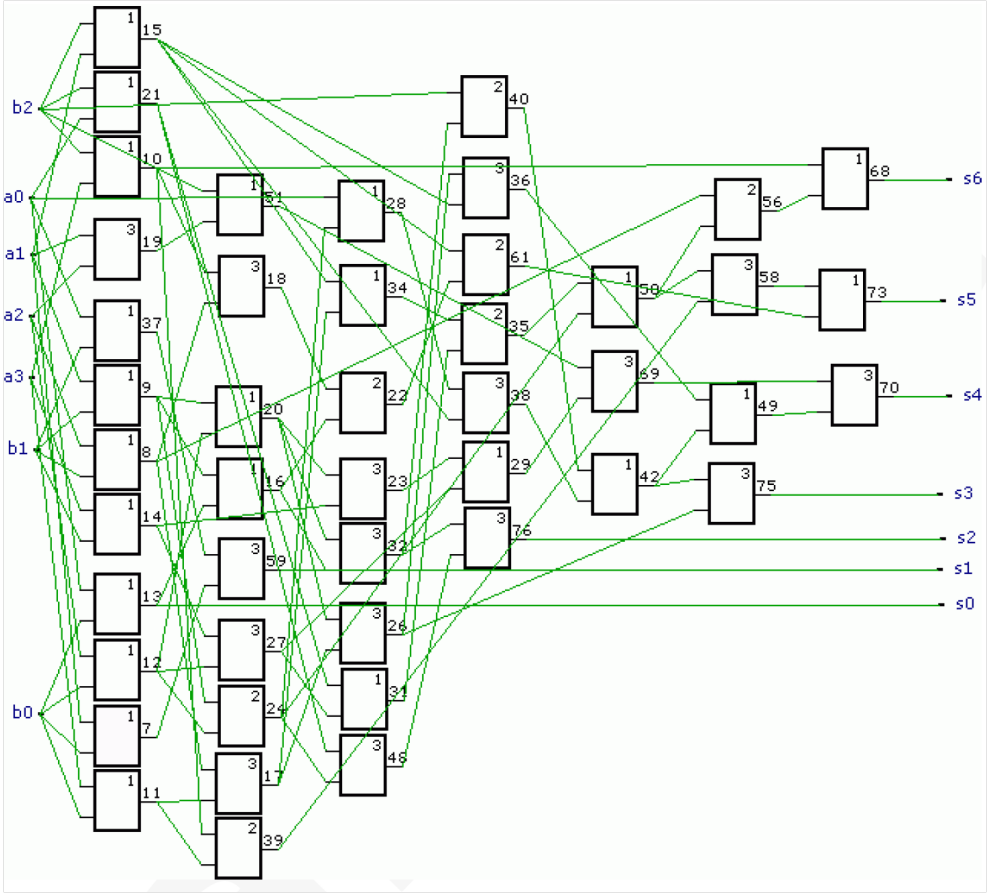
Non-silent mutations and their effects



CGP: Evolutionary Design of Boolean Circuits

CGP for evolution of 3×4-bit multiplier

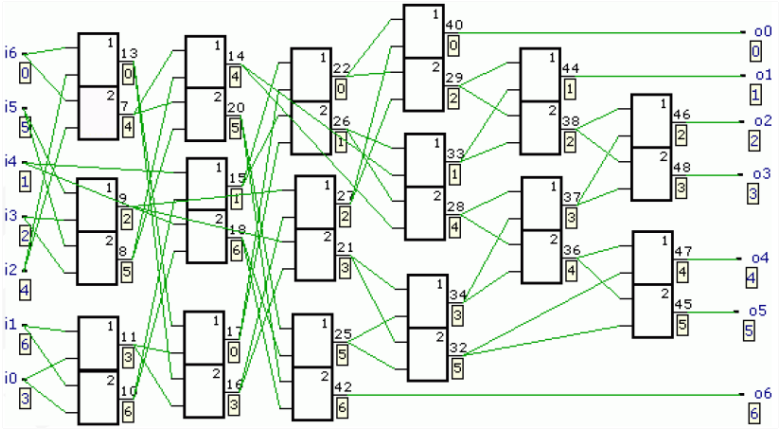
- $F = \{\texttt{AND}, \texttt{OR}, \texttt{XOR}, \texttt{Wire-Jumper}\}$
- $T = \{a_0, \ldots, a_3, b_0, \ldots, b_2\}$
- (1+4)-ES
- $r = 10$, $c = 7$, $l = 7$



CGP: Evolutionary Design of Boolean Circuits

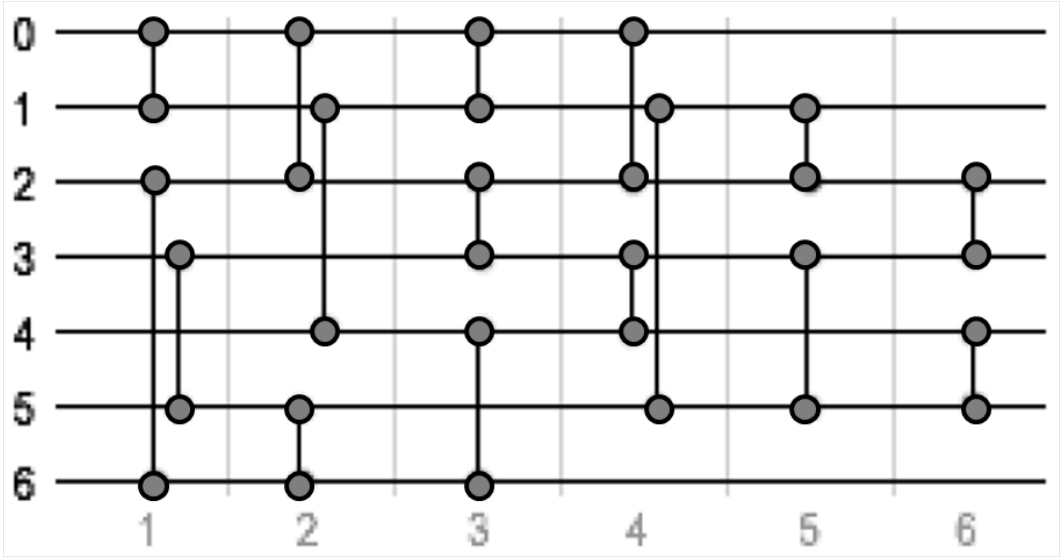
CGP for evolution of 7-bit sorting network

- $F = \{\texttt{Compare\&Swap}, \texttt{Wire-Jumper}\}$ realized by `AND-OR` units
- $T = \{a_0, \ldots, a_6\}$
- (1+4)-ES
- $r = 7$, $c = 8$, $l = 8$



CGP: Evolutionary Design of Boolean Circuits

7-bit sorting network represented by the CGP from previous slide realized by 16 C&S operations



CGP: Summary

Application areas

- Digital Circuit Design – parallel multipliers, digital filters, analogue circuits
- Mathematical functions –
- Control systems – Maintaining control with faulty sensors, helicopter control, simulated robot controller
- Artificial Neural Networks – Developmental Neural Architectures
- Image processing – Image filters

Pros/cons:

- (+) Flexible program representation – genotype-phenotype mapping allows for a neutral evolution.
- (+) Fixed genotype size but variable size and structure of the programs.
- (+) Explicit automatic code reuse.
- (-) Requires proper setting of the number of columns.

CGP: Sources

- Gene Expression Programming website (Candida Ferreira, Gepsoft Limited):
  `http://www.gepsoft.com/`
- Candida Ferreira: Gene Expression Programming: A New Adaptive Algorithm for Solving, Complex Systems, Vol. 13, issue 2: 87-129, 2001
  `http://arxiv.org/ftp/cs/papers/0102/0102027.pdf`
- Miller, J.F., Harding, S.L.: GECCO 2008 Tutorial: Cartesian Genetic Programming
  `http://portal.acm.org/citation.cfm?id=1389075`
- Home site: `http://www.cartesiangp.co.uk`
- Julian Miller: `http://www.elec.york.ac.uk/staff/jfm7.html`
- Simon Harding: `http://www.cs.mun.ca/šimonh/`
- Lukas Sekanina: `http://www.fit.vutbr.cz/šekanina/`

Sekanina L., Vašíček Z., Růžička R., Bidlo M., Jaroš J., Švenda P.: Evoluční hardware: Od automatického generování patentovatelných invencí k sebemodifikujícím se strojům. Academia Praha 2009