Cartesian Genetic Programming

Jiří Kubalík

Czech Institute of Informatics, Robotics and Cybernetics
CTU Prague

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

Contents

Cartesian Genetic Programming

- Representation
- Genotype-phenotype mapping
- Examples: Design of boolean Circuits

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.

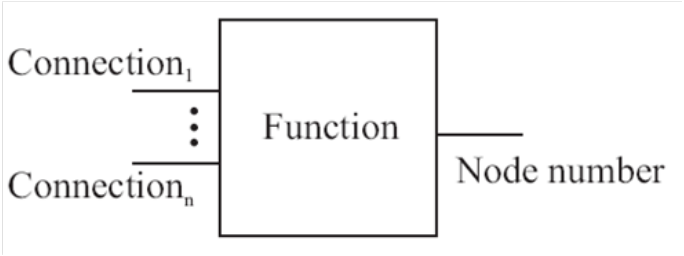
CGP Node

CGP program is a set of interconnected nodes.

A CGP node contains

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

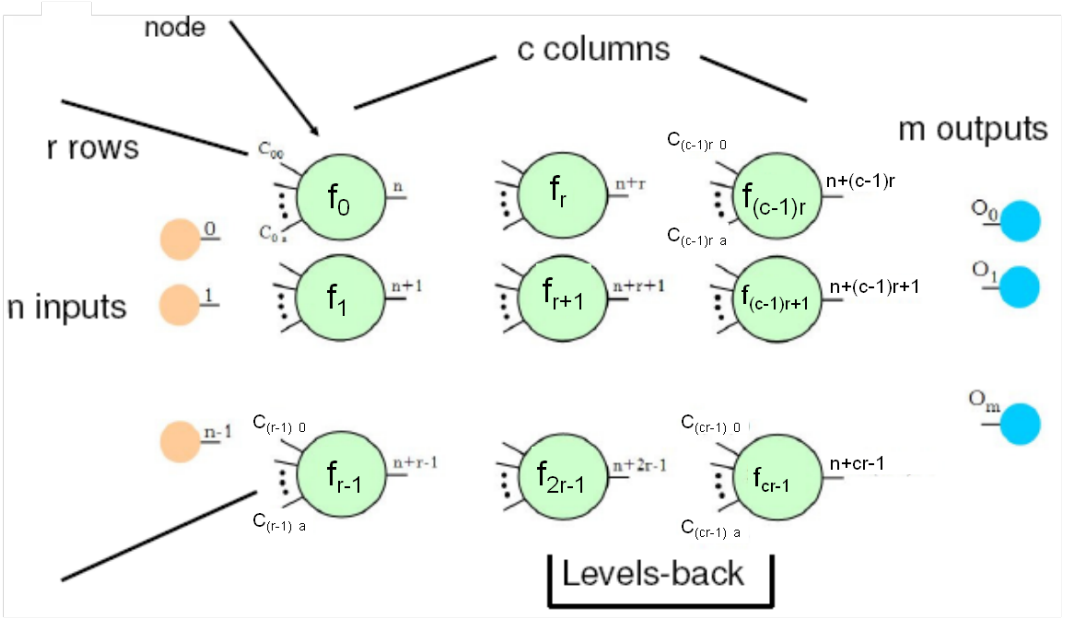
Each CGP node has an output with its unique number assigned that may be used as an input for another node.



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

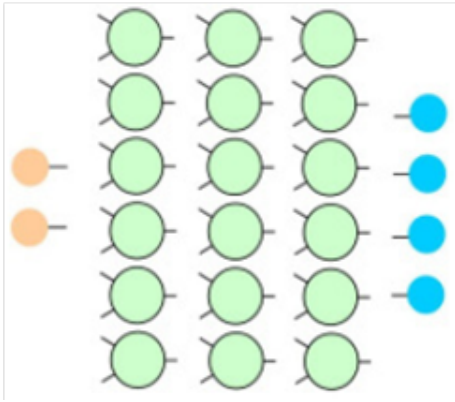
Nodes in the same column are not allowed to be connected to each other.

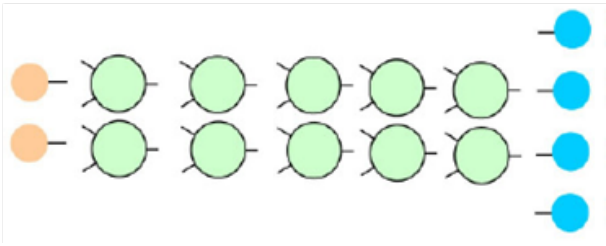
The **nodes interconnectivity** defines the maximum distance (in terms of the number of columns) between two connected nodes.

- 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

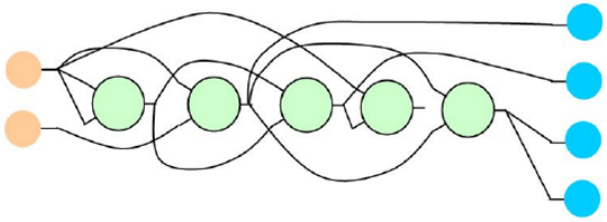




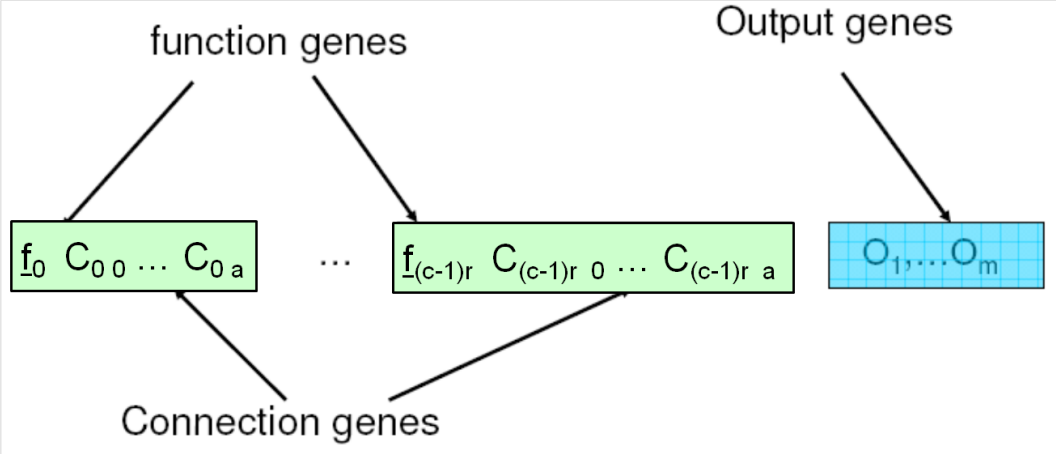
The length of the genotype (i.e. the maximum size of the CGP program) is fixed, however the **actual size and structure of the program can vary**.

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

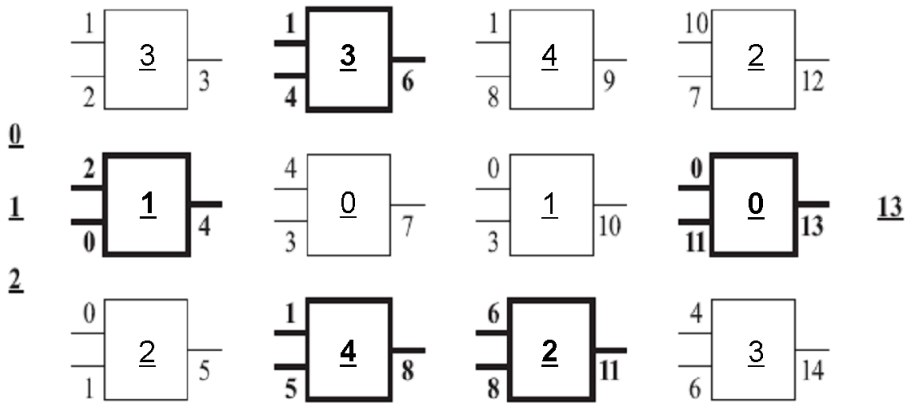


Usually, all functions have as many inputs as the maximum function arity.

Unused connections are ignored.

CGP Program Example

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



Look up table of 5 functions

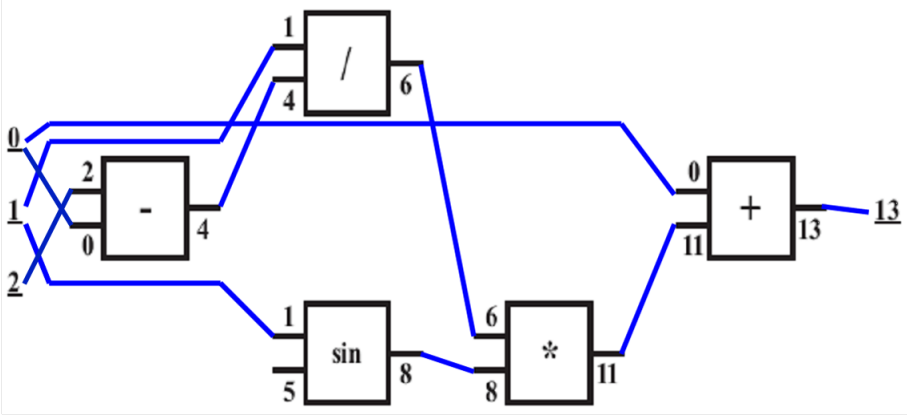
0	+	Add the arg1 to arg2
1	-	Subtract arg2 from arg1
2	*	Multiply arg1 to arg2
3	/	Divide arg1 by arg2
4	`sin`	Calculate sin of arg1

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)

The chromosome represents the following function: $y = x_0 + (x_1/(x_2 - x_0)) * \sin x_1$

CGP: Algorithm

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

$(1 + \lambda)$-ES:

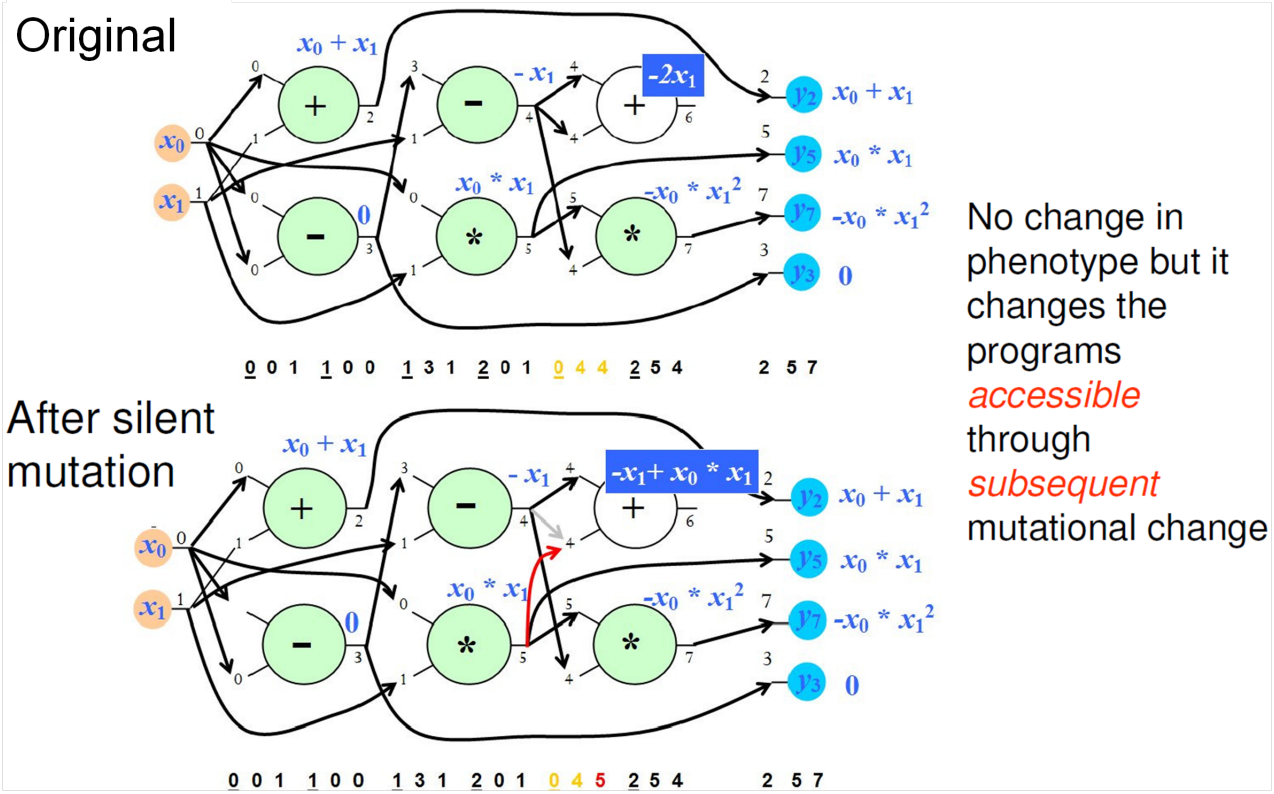
1. Generate a random solution S
2. while not stopping criterion do
3. Generate λ mutated versions of S
4. Replace S by the best individual individual out of the λ new solutions iff it is **not worse** than S.
5. Return S as the best solution found

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. This allows an introduction of new pieces of genetic code that can be plugged into the functional code later on.

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

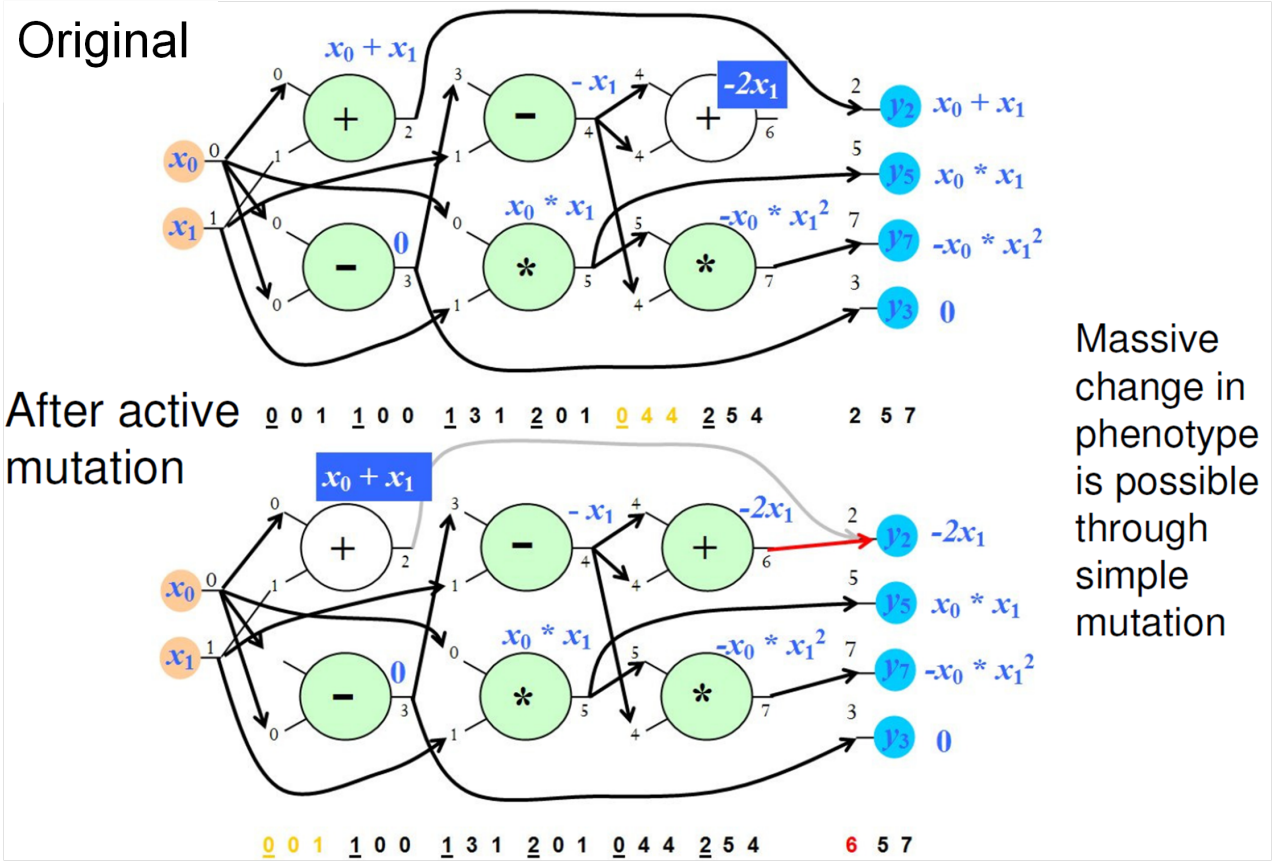
CGP: Point Mutation

Silent mutations and their effects



CGP: Point Mutation

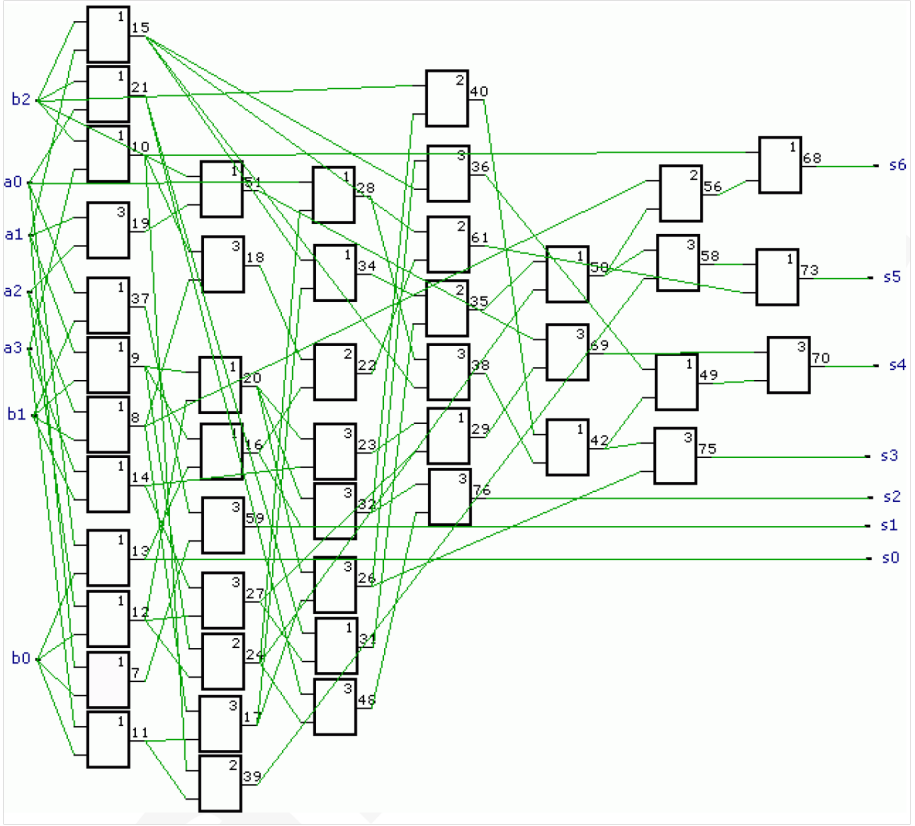
Non-silent mutations and their effects



CGP: Evolutionary Design of Boolean Circuits

CGP for evolution of **3x4-bit multiplier**

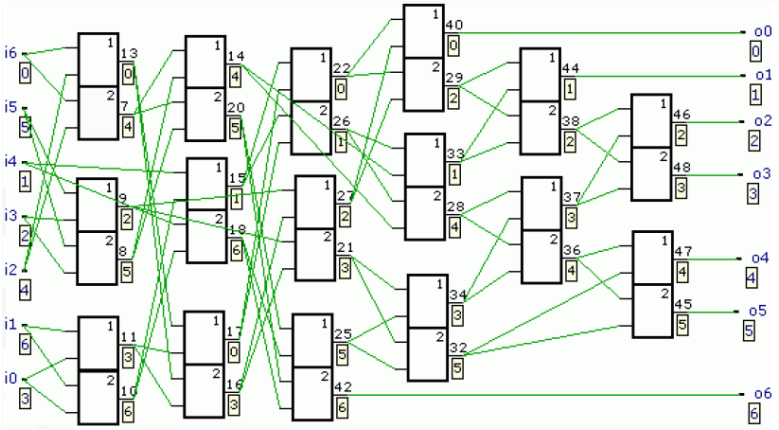
- $F = \{\texttt{AND}, \texttt{OR}, \texttt{XOR}, \texttt{Wire-Jumper}\}$
- $T = \{a_0, \ldots, a_3, b_0, \ldots, b_2\}$
- (1+4)-ES
- $r = 10$, $c = 7$, $l = 7$



CGP: Evolutionary Design of Boolean Circuits

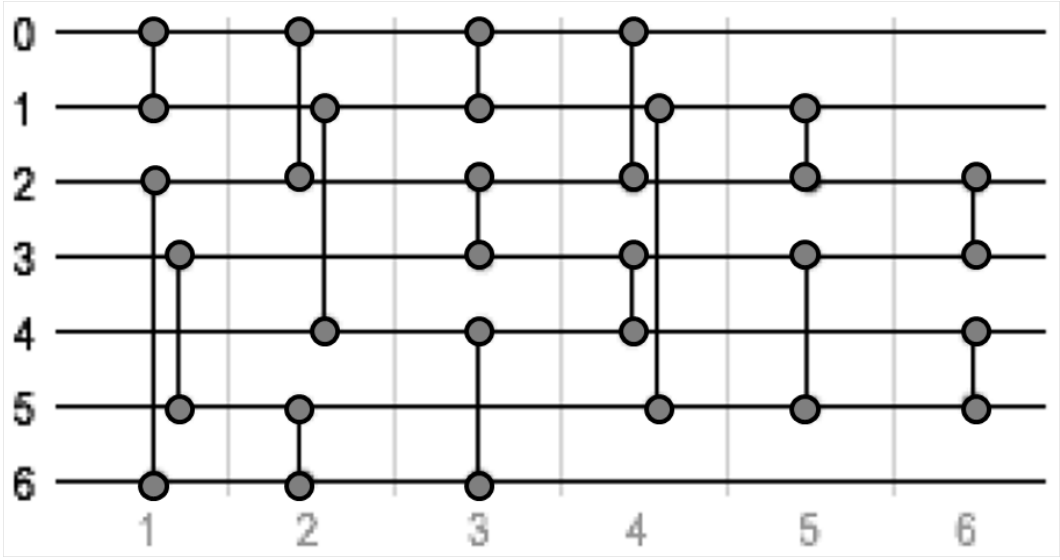
CGP for evolution of **7-bit sorting network**

- $F = \{\texttt{Compare\&Swap}, \texttt{Wire-Jumper}\}$ 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
- (+) Allows for an evolution of modules
- (–) Does not allow for multi level hierarchy in the ADFs

CGP: Sources

- Miller, J.F.: GECCO 2013 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/~simonh/`
- Lukas Sekanina: `http://www.fit.vutbr.cz/~sekanina/`

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