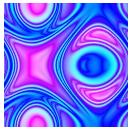
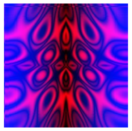


GECCO 2012 Tutorial:
Cartesian Genetic Programming



Julian F. Miller
Dept of Electronics
University of York, UK
julian.miller@york.ac.uk

Simon L. Harding
IDSIA, Galleria 2
6928 Manno-Lugano, Switzerland

Machine Intelligence Ltd, UK
slh@evolutioninmaterio.com

GECCO'12 Companion, July 7–11, 2012, Philadelphia, PA, USA.
ACM 978-1-4503-1178-6/12/07

Abstract

Cartesian Genetic Programming (CGP) is an increasingly popular and efficient form of Genetic Programming that was developed by Julian Miller in 1999 and 2000.

In its classic form, it uses a very simple integer based genetic representation of a program in the form of a directed graph. Graphs are very useful program representations and can be applied to many domains (e.g. electronic circuits, neural networks). In a number of studies, CGP has been shown to be comparatively efficient to other GP techniques. It is also very simple to program.

Since then, the classical form of CGP has been developed made more efficient in various ways. Notably, by including automatically defined functions (modular CGP) and self-modification operators (self-modifying CGP). SMCGP was developed by Julian Miller, Simon Harding and Wolfgang Banzhaf. It uses functions that cause the evolved programs to change themselves as a function of time. Using this technique it is possible to find general solutions to classes of problems and mathematical algorithms (e.g. arbitrary parity, n-bit binary addition, sequences that provably compute pi and e to arbitrary precision, and so on).

The tutorial will cover the basic technique, advanced developments and applications to a variety of problem domains.

Contents

Genetic Programming

- The automatic evolution of computer programs
  - Tree-based, Koza 1992
  - Stack-based, Perkis 1994, Spector 1996 onwards (push-pop GP)
  - Linear GP, Nordin and Banzhaf 1996
  - **Cartesian GP,** Miller 1997
  - Parallel Distributed GP, Poli 1996
  - Grammatical Evolution, Ryan 1998
  - Lots of others…

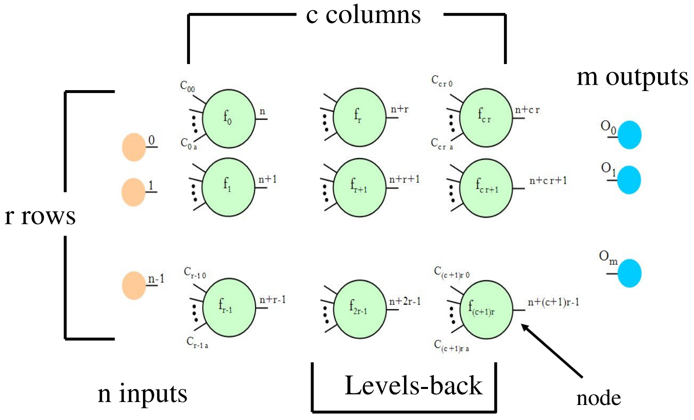
Origins of Cartesian Genetic Programming (CGP)

- Grew out of work in the evolution of digital circuits, Miller and Thomson 1997. First actual mention of the term *Cartesian Genetic Programming* appeared at GECCO in 1999.
- Originally, represents programs or circuits as a two dimensional grid of program primitives.
- This is loosely inspired by the architecture of digital circuits called FPGAs (field programmable gate arrays)

What defines CGP?

- The genotype is a list of integers (and possibly parameters) that represent the program primitives and how they are connected together
  - CGP represents programs as *graphs* in which there are *non-coding genes*
- The genes are
  - Addresses in data (connection genes)
  - Addresses in a look up table of functions
  - Additional parameters
- This representation is very simple, flexible and convenient for many problems

CGP General form



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

Allelic constraints for directed acyclic graphs

All function genes f_i must takes allowed function alleles: $0 \leq f_i \leq n_f$

Nodes connections C_{ij} of a node in column j, and levels-back l, must obey (to retain directed acyclicity)

$j \geq l \qquad n + (j-l)r \leq C_{ij} \leq n + jr$

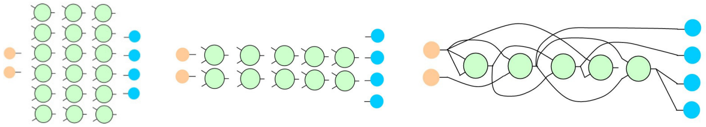
$j < l \qquad 0 \leq C_{ij} \leq n + jr$

Output genes (can connect to any previous node or input)

$$0 \leq O_i \leq n + cr - 1$$

Types of graphs easily controlled

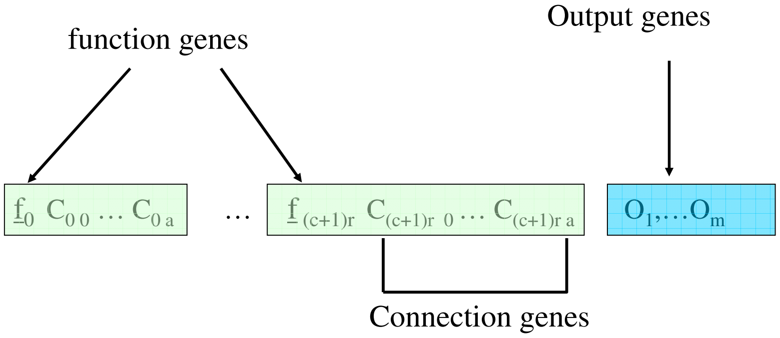
- Depending on *rows*, *columns* and *levels-back* a wide range of graphs can be generated



- When *rows* =1 and *levels-back* = *columns* *arbitrary* directed graphs can be created with a maximum depth
  - In general choosing these parameters imposes the least constraints. So without specialist knowledge this is the best and most general choice

---

CGP genotype



function genes

Output genes

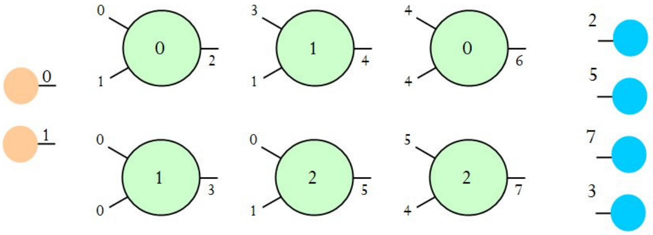
$f_0\ C_{0\,0} \ldots C_{0\,a}$ … $f_{(c+1)r}\ C_{(c+1)r\ 0} \ldots C_{(c+1)r\ a}$ $O_1,\ldots O_m$

Connection genes

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

Unused connections are ignored

Example



Encoding of graph as a list of integers (i.e. the genotype)

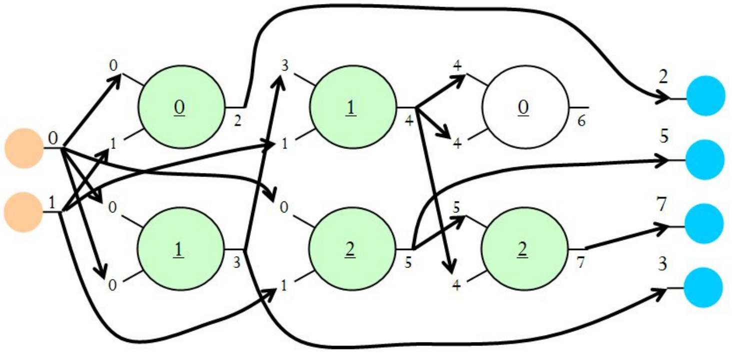
0 0 1 1 0 0 1 3 1 2 0 1 0 4 4 2 5 4 2 5 7 3

Example: Function look up table

The function genes are the *addresses* in a user-defined lookup table of functions

0	+	Add the data presented to inputs
1	-	Subtract the data presented to inputs
2	*	Multiply data presented to inputs
3	/	Divide data presented to inputs (protected)

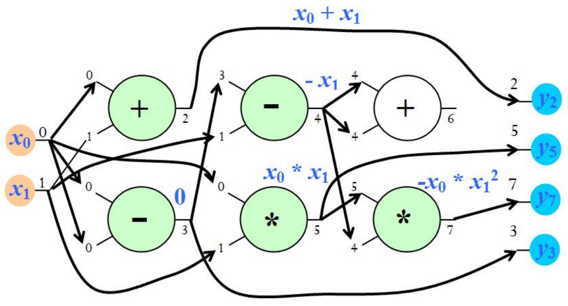
Obtaining the graph



Encoding of graph as a list of integers (i.e. the genotype)

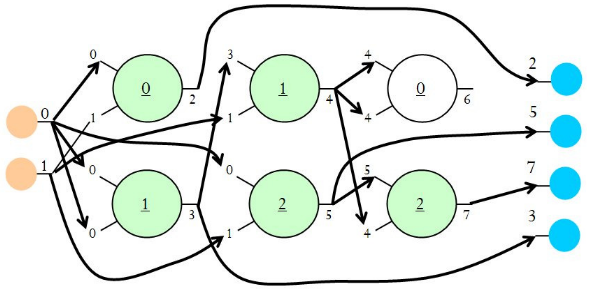
0 0 1 1 0 0 1 3 1 2 0 1 0 4 4 2 5 4   2 5 7 3

So what does the graph represent?



$$y_2 = x_0+x_1$$
$$y_5 = x_0*x_1$$
$$y_7 = -x_0*x_1^2$$
$$y_3 = 0$$

What happened to the node whose output label is 6?



The node was not used so the genes are *silent* or *non-coding*

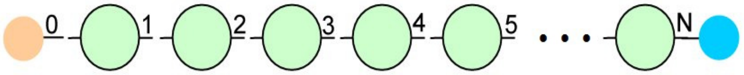
0 0 1 1 0 0 1 3 1 2 0 1 0 4 4 2 5 4   2 5 7 3

The CGP genotype-phenotype map

- When you decode a CGP genotype many nodes and their genes can be ignored because they are not referenced in the path from inputs to outputs
- These genes can be altered and make no difference to the *phenotype*, they are non-coding
- Clearly there is a many-to-one genotype to phenotype map
- How redundant is the mapping?

A mathematical aside: CGP and Stirling numbers

- Assume that a CGP graph has the following parameters
- Number of rows_= 1
- Levels-back = num_cols = n
- Arity of functions = 1
- There is one input
- Assume that the output is taken from the last node



The number of genotypes, G, that have a phenotype of size k(nodes) can be shown to obey a recurrence relation obeyed by *unsigned Stirling numbers of the first kind.*

$$G(n+1, k) = nG(n,k) + G(n, k-1)$$

How many genotypes of length *n* map to a phenotypes of length *k*?

n	*k*								
	1	2	3	4	5	6	7	8	9
1	1								
2	1	1							
3	2	3	1						
4	6	11	6	1					
5	24	50	35	10	1				
6	120	274	225	85	15	1			
7	720	1764	1624	735	175	21	1		
8	5040	13068	13132	6759	1960	322	28	1	
9	40320	109584	118124	67284	22449	4536	546	36	1

Average number of active nodes in a genotype of length 9 is 2.83

Clearly, with say a genotype of 100 nodes, the number of genotypes that map to a phenotype with say about 10 nodes is an astronomical number

Decoding CGP chromosomes is *easy*

①
```
// L  = MaxGraph.Length
// I  = Number of program inputs
// N = Number of program outputs
bool        ToEvaluate[L]
double      NodeOutput[L+I]
int         NodesUsed[M]
```

②
```
// identify initial nodes that need to be evaluated
p = 0
do
  ToEvaluate[OutputGene[p]] = true
  p = p + 1
while (p < N)
// determine nodes used
p = L-1
q=0
do
  if (ToEvaluate[p])
    x = Node[p].Connection1
    y = Node[p].Connection2
    ToEvaluate[x] = true
    ToEvaluate[y] = true
    q=q+1
   NodesUsed[q]=p;
  endif
  p = p - 1
while ( p >= 0)
```

③
```
// load input data values
p = 0
do
  NodeOutput[p] = InputData[p]
  p = p + 1
while (p < I)
```

④
```
//Execute graph
for  p = I to p < q+I
    x = Node[NodesUsed[p]].Connection1
    y = Node[NodesUsed[p]].Connection2
    z = Node[NodesUsed[p]].Function
    NodeOutput[p] = ComputeNode(NodeOutput[x], NodeOutput[y],z)
endfor
```

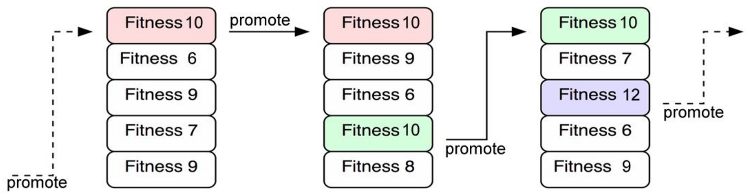
Point mutation

- Most CGP implementations only use mutation.
- Carrying out mutation is very simple. It consists of the following steps. The genes must be chosen to be valid alleles

```
//Decide how many genes to change:num_mutations
while (mutation_counter < num_mutations)
{
        get gene to change
        if (gene is a function gene)
          change gene to randomly chosen new valid function
        else if (gene is a connection gene)
          change gene to a randomly chosen new valid connection
        else
          change gene to a new valid output connection
}
```

Evolutionary Strategy

- CGP often uses a variant of a simple algorithm called (1 + 4) Evolutionary Strategy
  - However, an offspring is always chosen if it *is equally as fit* or has better fitness than the parent



promote

Fitness 10
Fitness 6
Fitness 9
Fitness 7
Fitness 9

promote

Fitness 10
Fitness 9
Fitness 6
Fitness 10
Fitness 8

promote

Fitness 10
Fitness 7
Fitness 12
Fitness 6
Fitness 9

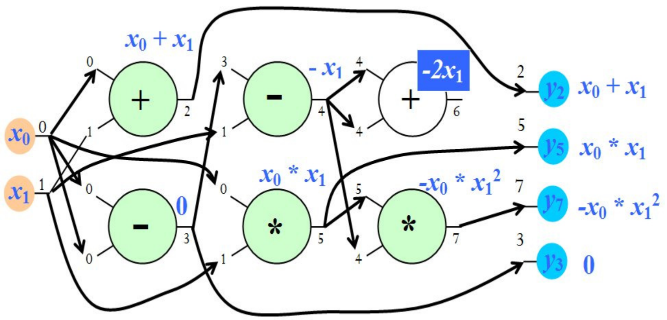
promote

Crossover or not?

- Recombination doesn't seem to add anything *(Miller 1999, "An empirical study…")*
- However if there are multiple chromosomes with independent fitness assessment then it helps a LOT (*Walker, Miller, Cavill 2006, Walker, Völk, Smith, Miller, 2009)*
- Some work using a floating point representation of CGP has suggested that crossover might be useful (*Clegg, Walker, Miller 2007)*

Silent mutations and their effects

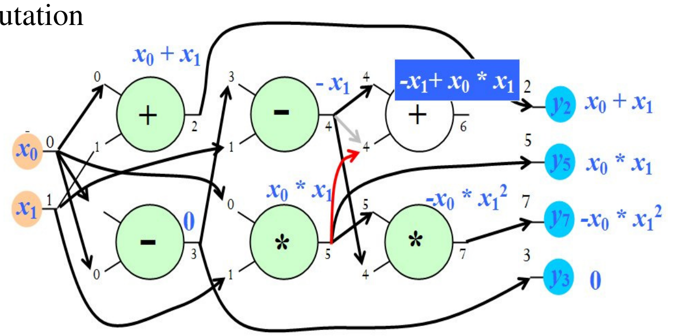
Original



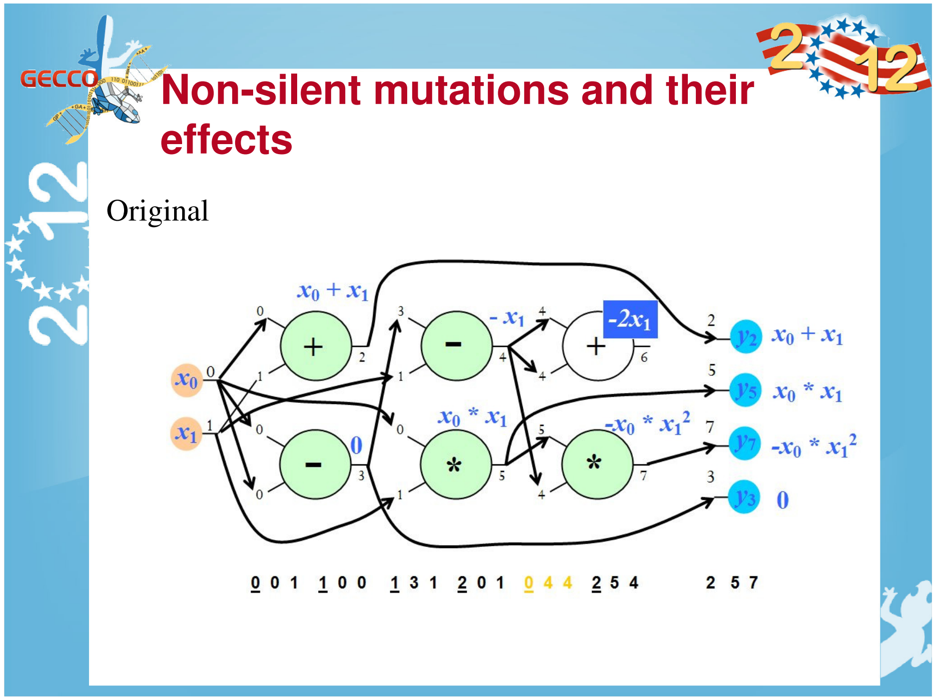
0 0 1   1 0 0   1 3 1   2 0 1   0 4 4   2 5 4   2 5 7

Silent mutations and their effects

After silent mutation

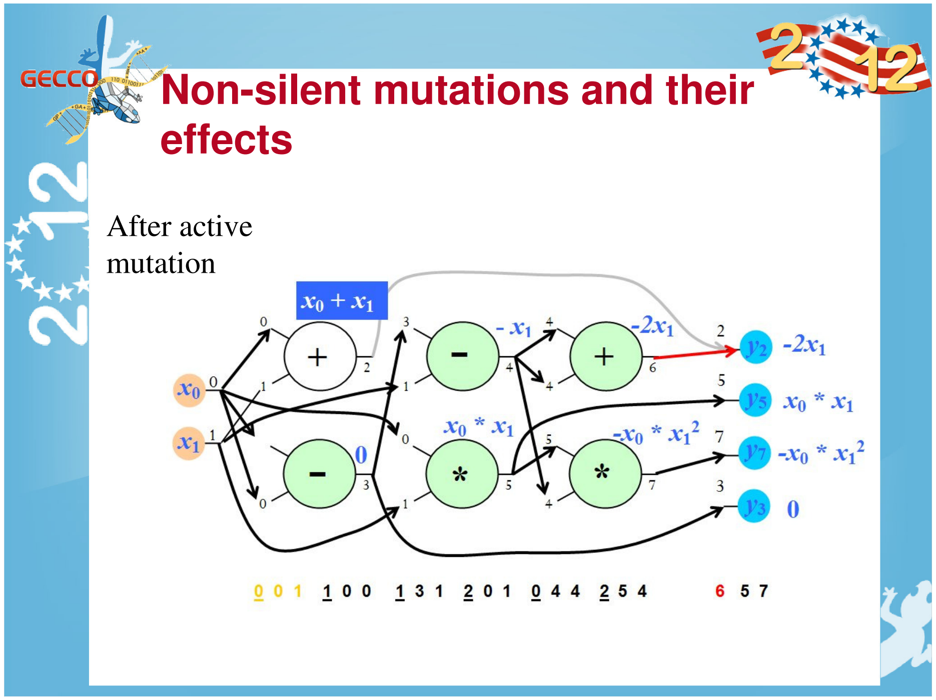


0 0 1   1 0 0   1 3 1   2 0 1   0 4 5   2 5 4   2 5 7



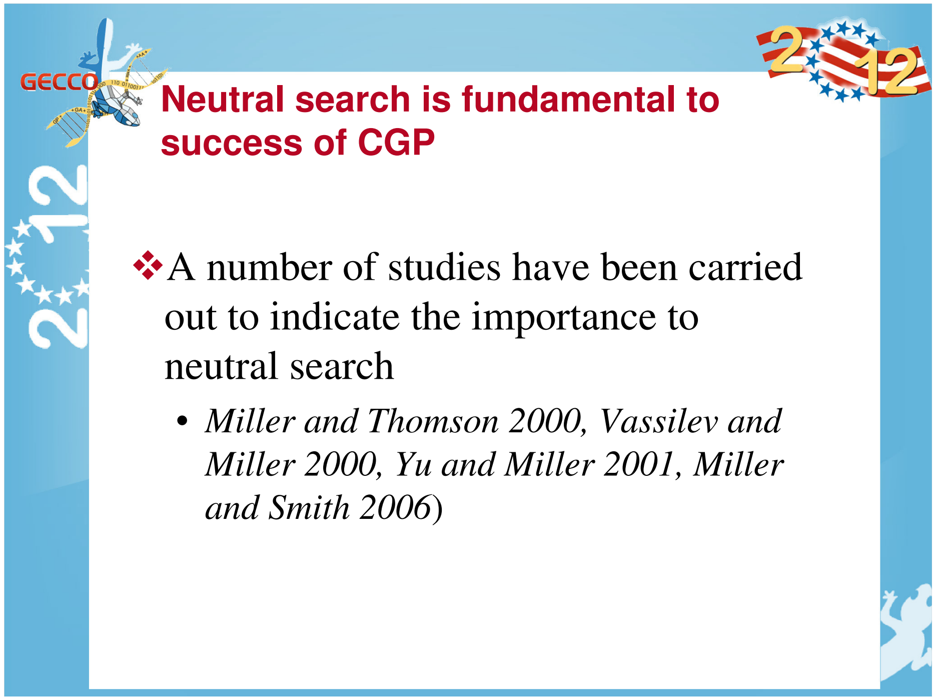
Non-silent mutations and their effects

Original



Non-silent mutations and their effects

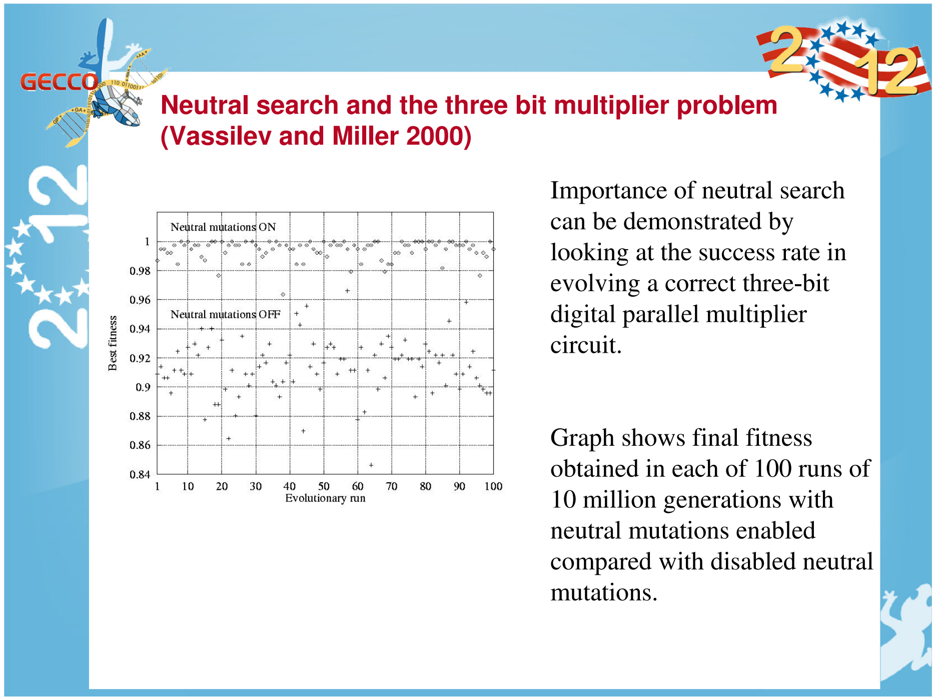
After active mutation



Neutral search is fundamental to success of CGP

❖ A number of studies have been carried out to indicate the importance to neutral search

- *Miller and Thomson 2000, Vassilev and Miller 2000, Yu and Miller 2001, Miller and Smith 2006*)



Neutral search and the three bit multiplier problem (Vassilev and Miller 2000)

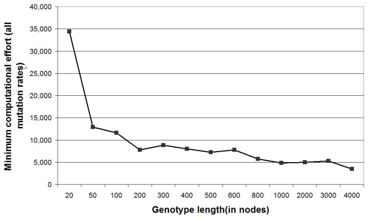
Importance of neutral search can be demonstrated by looking at the success rate in evolving a correct three-bit digital parallel multiplier circuit.

Graph shows final fitness obtained in each of 100 runs of 10 million generations with neutral mutations enabled compared with disabled neutral mutations.

In CGP, *large* genotypes and *small* mutation evolve solutions to problems more quickly [Miller and Smith 2006]

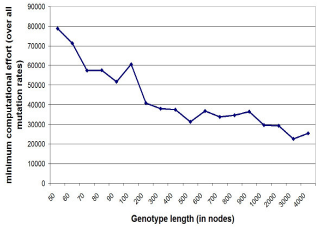
Even 3 parity with gate set

{AND, OR, NAND, NOR}.



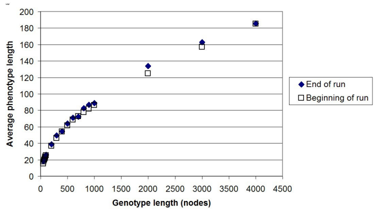
Two-bit multiplier with gate set

{AND, OR, NAND, NOR}.

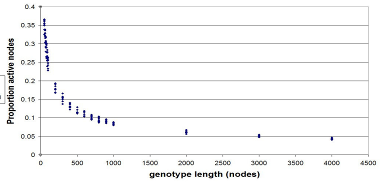


•However big genotypes does NOT mean big phenotypes (programs)….

Phenotype length versus genotype length (two-bit multiplier)



Average phenotype length for the initial population contrasted with the average phenotype length at conclusion of evolutionary run versus genotype length with 1% mutation



Average proportion of active nodes in genotype at the conclusion of evolutionary run for all mutation rates versus genotype length

SEARCH MOST EFFECTIVE WHEN 95% OF ALL GENES ARE INACTIVE!!

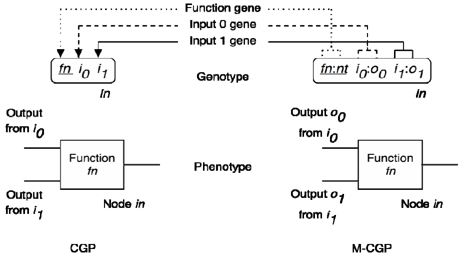
Modular/Embedded CGP (Walker, Miller 2004)

❖ So far have described a form of CGP (classic) that does not have an equivalent of Automatically Defined Functions (ADFs)

❖ Modular CGP allows the use of modules (ADFs)

- Modules are dynamically created and destroyed
- Modules can be evolved
- Modules can be re-used

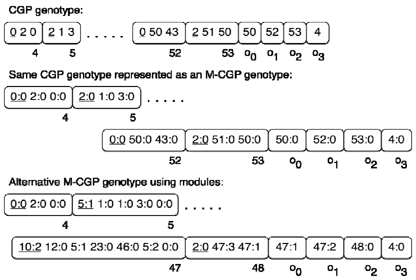
Representation Modification 1



❖ Each gene encoded by *two* integers in M-CGP

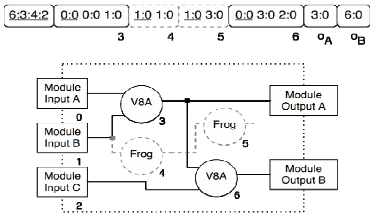
- Function/module number and node type
- Node index and node output
  - – nodes can have multiple outputs

Representation Modification 2



- ❖ M-CGP has a bounded variable length genotype
  - Compression and expansion of modules
    - Increases/decreases the number of nodes
  - Varying number of module inputs
    - Increases/decreases the number of genes in a node

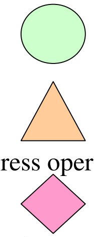
Modules



- ❖ Same characteristics as M-CGP
  - Bounded variable length genotype
  - Bounded variable length phenotype
- ❖ Modules also contain inactive genes as in CGP
- ❖ Modules can *not* contain other modules!

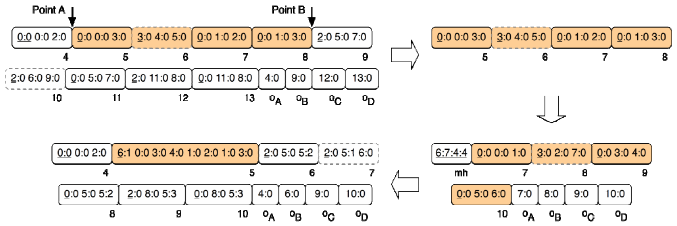
Node Types

- ❖ Three node types:
  - Type 0
    - Primitive function
  - Type I
    - Module created by compress operator
  - Type II
    - Module replicated by genotype point-mutation



- ❖ Control excessive code growth
  - Genotype can return to original length at any time

Creating and Destroying a Module



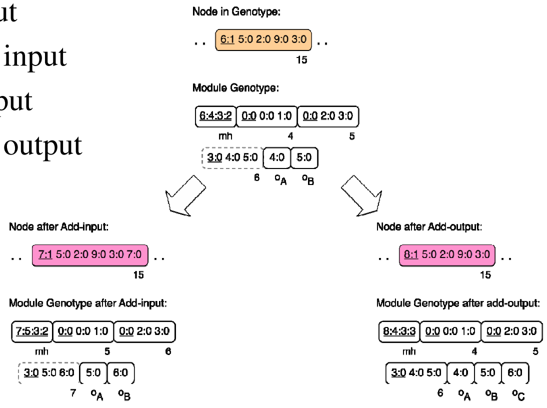
- ❖ Created by the compress operator
  - Randomly acquires sections of the genotype into a module
    - Sections must ONLY contain type 0 nodes
- ❖ Destroyed by the expand operator
  - Converts a random type I module back into a section of the genotype

Module Survival

- ❖ Twice the probability of a module being destroyed than created
- ❖ Modules have to replicate to improve their chance of survival
  - Lower probability of being removed
- ❖ Modules must also be associated with a high fitness genotype in order to survive
  - Offspring inherit the modules of the fittest parent

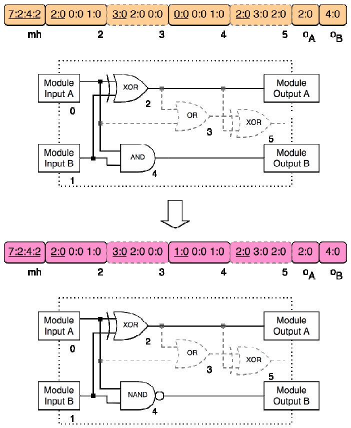
Evolving a Module I

- ❖ Structural mutation
  - Add input
  - Remove input
  - Add output
  - Remove output



Evolving a Module II

- ❖ Module point-mutation operator
  - Restricted version of genotype point-mutation operator
    - Uses only primitive functions



Re-using a Module

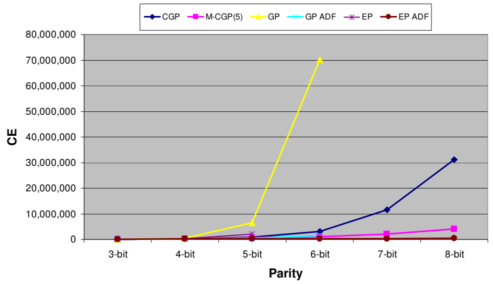
- ❖ Genotype point-mutation operator
  - Modified CGP point-mutation operator
- ❖ Allows modules to replicate in the genotype
  - Primitive (type 0) → module (type II)
  - Module (type II) → module (type II)
  - Module (type II) → primitive (type 0)
- ❖ Does *NOT* allow type I modules to be mutated into primitives (type 0) or other modules (type II)
  - Type I modules can only be destroyed by *Expand* (and are only created by *Compress*)

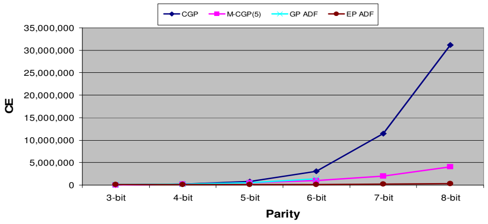
Experimental parameters

Parameter	Value
Population size	5
Initial genotype size	100 nodes (300 genes)
Genotype point mutation rate	3% (9 genes)
Genotype point mutation probability	1
Compress/Expand probability ◇	0.1/0.2
Module point mutation probability ◇	0.04
Add/Remove input probability ◇	0.01/0.02
Add/Remove output probability ◇	0.01/0.02
Module list initial contents ◇	Empty
Number of independent runs	50

- NOTES: ◊ these parameters only apply to Modular (Embedded) CGP
- Results heavily dependent on the maximum number of nodes allowed. Much better results are obtained when larger genotype lengths are used.

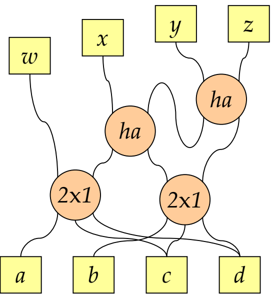
Even Parity Results



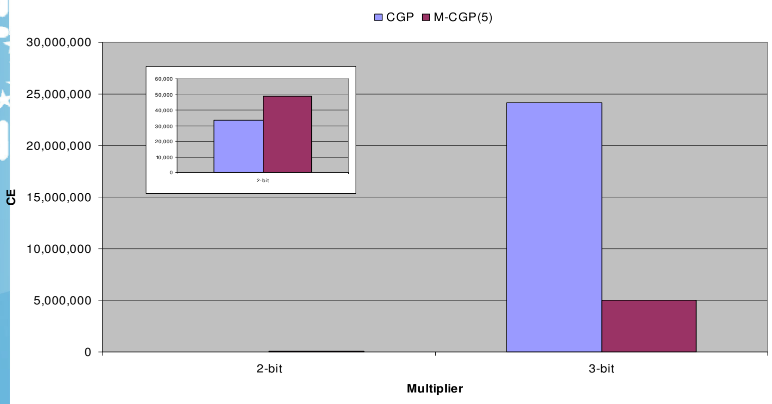


Digital Multiplier

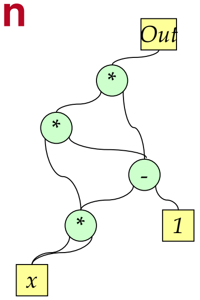
- Two digital multiplier problems:
  - 2-bit and 3-bit
- Function set:
  - AND, AND (one input inverted), XOR, OR
- Fitness Function:
  - Number of phenotype output bits that differ from the perfect *n*-bit digital multiplier solution
  - Perfect solution has a fitness of zero
- Results are averaged over fifty independent runs



Multiplier Results

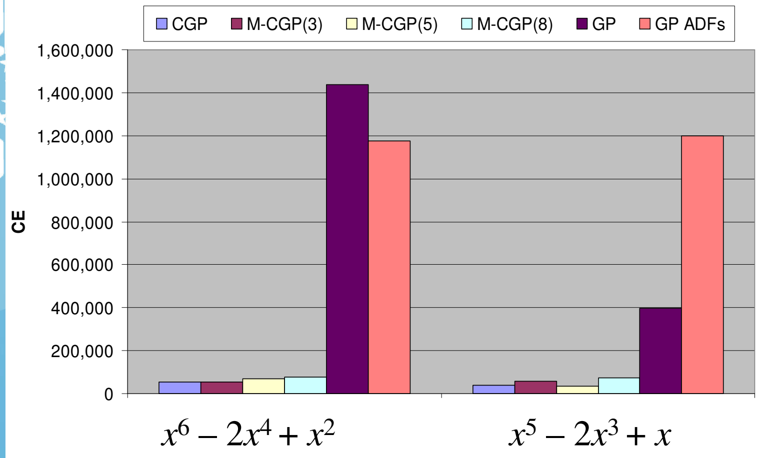


Symbolic Regression



- Two problems:
    - $x^6 - 2x^4 + x^2$
    - $x^5 - 2x^3 + x$
- Function set:
    - +, -, *, / (protected)
- Fitness Function:
    - Absolute error over all fifty points in the input set
    - Solution found when absolute error is within 0.01 of each point
- Results averaged over fifty independent runs

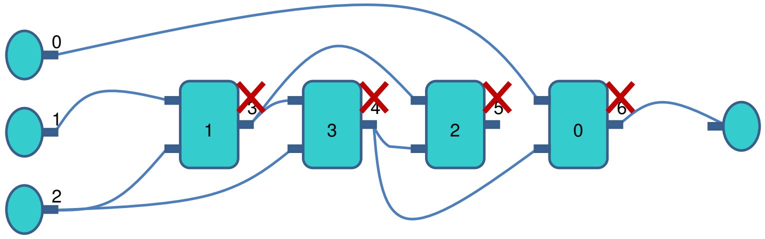
Symbolic Regression Results



Self-modifying Cartesian Genetic programming

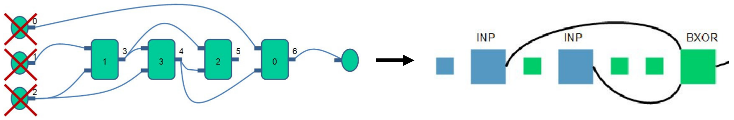
- A developmental form of Cartesian Genetic Programming (CGP)
    - Includes self modification functions.
    - ‘General purpose’ GP system
    - Phenotype can vary over time (with iteration)
    - Can switch off its own self-modification

Changes to CGP: relative addressing



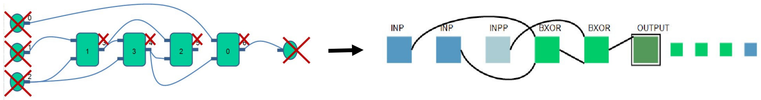
- Replaced direct node addressing with relative addressing
    - Always use 1 row (not rectangular)
    - Connection genes say how many nodes back

Changes to CGP: Inputs



- ❖Replace input calls with a function.
  - We call these functions INP, INPP, SKIPINP
- ❖Pointer keeps track of ‘current input’.
  - Call to INP returns the current input, and moves the pointer to the next input.
- ❖Connections beyond graph are assigned value 0.

Changes to CGP: Outputs



- ❖Removed output nodes.
- ❖Genotype specifies which nodes are outputs.
- ❖If no OUTPUT function then last active node is used
  - Other defaults are used in situations where the number of outputs does not match the number required

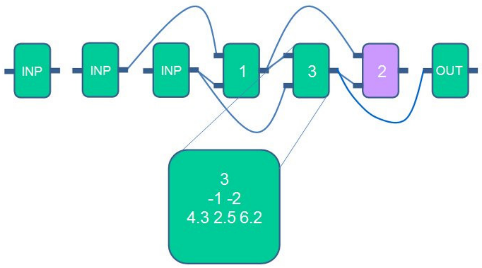
---

Changes to CGP: Arguments

- ❖Nodes also contain a number of ‘arguments’.
  - 3 floating point numbers
  - Used in various self-modification instructions
  - Cast to integers when required

SMCGP Nodes: summary

- ❖Each node contains:
  - Function type
  - Connections as relative addresses
  - 3 floating point numbers



SMCGP: Functions

- Two types of functions:
  - Computational
    - Usual GP computational functions
  - Self-modifying
    - Passive computational role (see later)

Some Self-Modification Functions

Operator	Parameters: use node address and the three node arguments	Function
MOVE	Start, End, Insert	Moves each of the nodes between Start and End into the position specified by Insert
DUP	Start, End, Insert	Inserts copies of the nodes between Start and End into the position specified by Insert
DELETE	Start, End	Deletes the nodes between Start and End indexes
CHF	Node, New Function	Changes the function of a specified node to the specified function
CHC	Node, Connection1, Connection2	Changes the connections in the specified node

SMCGP Execution

- Important first step:
  - Genotype is duplicated to phenotype.
  - Phenotypes are executed:
    - Self modifications are only made to the phenotype.

Self Modification Process: The To Do list

- Programs are iterated.
- If triggered, self modification instruction is added to a To Do list.
- At the end of each iteration, the instructions on this list are processed.
- The maximum size of the To Do list can be predetermined

Computation of a SM node

- Functions are appended to the To Do list if:
  - The first input > the second input.
- And:
  - The To Do list isn’t too big.

Publications using SMCGP

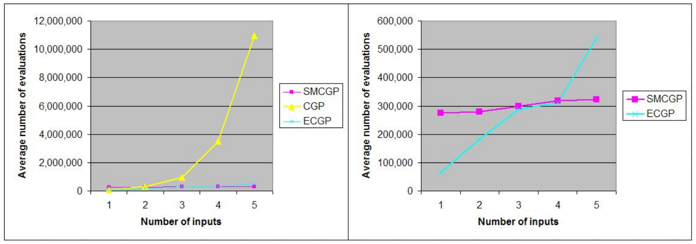
- General Parity Problem (CEC 2009)
- Mathematical Problems (EuroGP 2009, GECCO 2007)
- Learning to Learn (GECCO 2009)
- Generating Arbitrary Sequences (GECCO 2007)
- Computing the mathematical constants pi and e (GECCO 2010 in GDS track)
- General adder and many other problems (GPEM Tenth Anniversary Special Issue, 2010)

Authors: Harding, Miller, Banzhaf

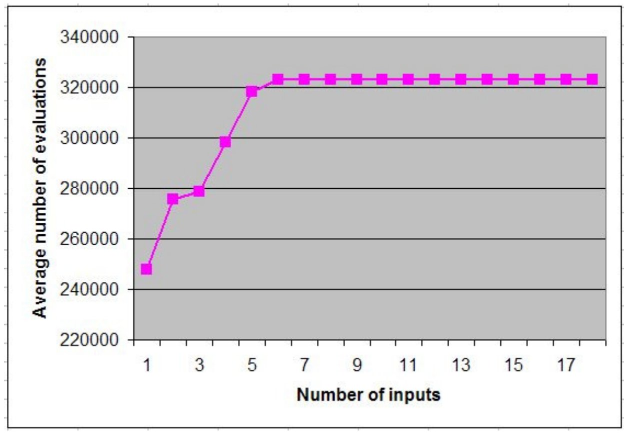
Evolving Parity

- Each iteration of program should produce the next parity circuit.
  - On the first iteration the program has to solve 2 bit parity. On the next iteration, 3 bit ... up to 22 parity
  - Fitness is the cumulative sum of incorrect bits
- Aim to find *general* solution
  - Solutions can be proved to general
    - See GPEM 2010 paper
- CGP or GP cannot solve this problem as they have a finite set of inputs (terminals)

Parity results: SMCGP versus CGP and ECGP



Scaling behaviour of SMCGP



Evolving pi

- Iterate a maximum of 10 times
- If program output does not get closer to pi at the next iteration, the program is stopped and large fitness penalty applied
- Fitness at iteration, *i*, is absolute difference of output at iteration *i* and pi
- One input: the numeric constant 1.

Evolving pi: an evolved solution

- An evolved solution

$$f(i) = \begin{cases} cos(sin(cos(sin(0)))) & i = 0 \\ f(i-1) + sin(f(i-1)) & i > 0 \end{cases}$$

- f(10) is correct to the first 2048 digits of pi
- It can be proved that *f*(*i*) rapidly converges to pi in the limit as *i* tends to infinity

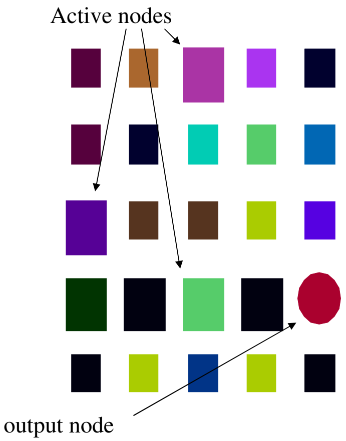
---

Further results

- Other mathematically provable results found so far:
  - Evolved a program that can carry out the bitwise addition of an arbitrary number of inputs
  - Evolved a sequence that converges to e
- Other results
  - Evolved a sequence function that generates the first 10 Fibonacci numbers
  - Evolved a power function x^n
  - Bioinformatics classification problem (finite inputs)
    - SMCGP performed no worse than CGP

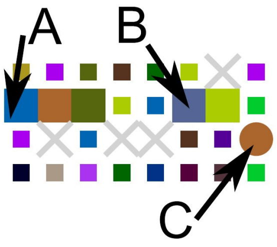
Two dimensional SMCGP (SMCGP2)

- SMCGP2: genes
  - Function
  - Connections
  - Numeric Constant
- Arguments are now 2 D vectors
  - SM size (SMS)
  - SM location (SML)



SMCGP2: Vector relative addressing and Empty nodes

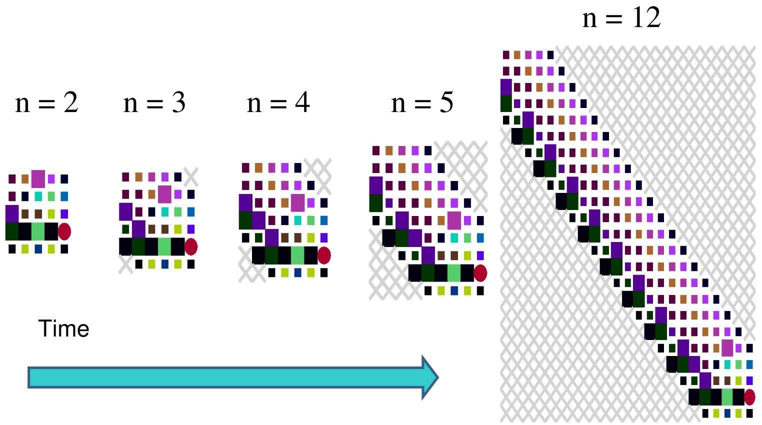
- There are **empty nodes** are represented by X
- The relative address from C to B is (2, 1)
  - meaning 2 nodes to the left, and one node up.
- The relative address of C to A is (4,1).
- Note how the empty nodes are not counted when computing how many nodes back to connect.



SMCGP2: Self Modifying Functions

- Simplified SM function set
  - Duplicate section, insert elsewhere.
  - Duplicate section, overwrite elsewhere.
  - Crop to a section.
  - Delete a section.
  - Add a row or column.
  - Delete a row or column.
  - NULL

SMCGP2: Solving even-n parity



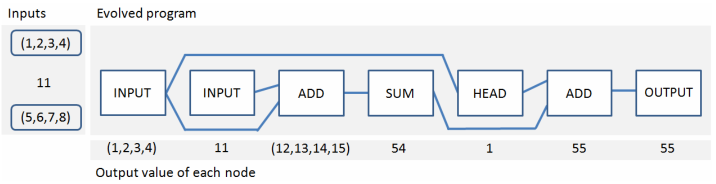
SMCGP:Some observations

- In SMCGP there are implicit
  - Loops
  - Recursion
  - Modules/functions
  - Halting (telomeres)
- Also have “partial” loops/recursion

Multi-type CGP (MT-CGP)

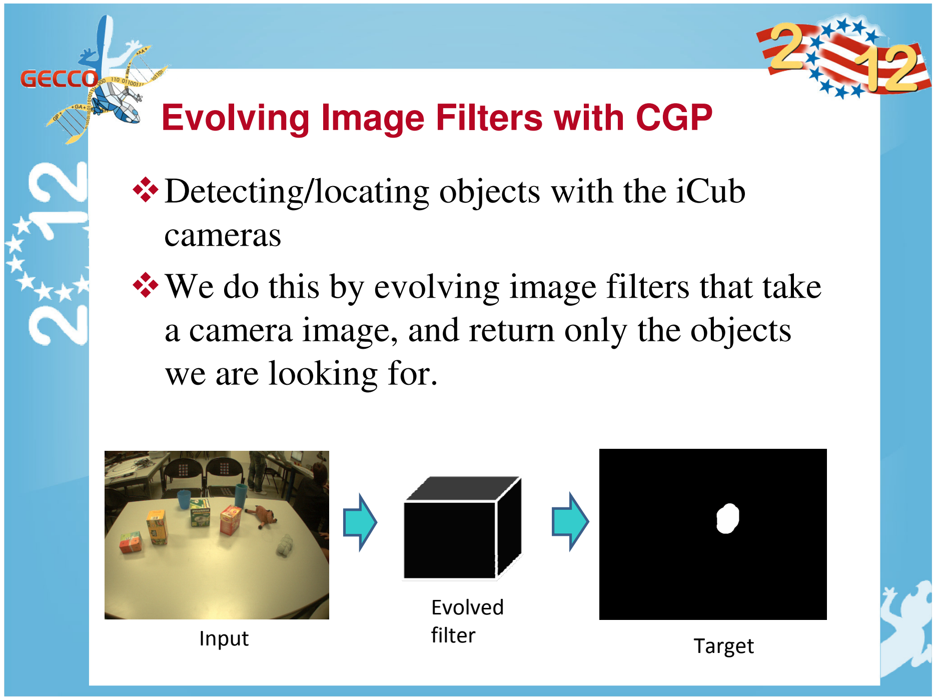
- Genotype pretty much classic CGP
  - Genotype is a (partly connected, feed-forward) graph
  - Graph is a list of nodes
    - Each node contains:
      - Function (from a function set)
      - Two connections (to other nodes)
      - real number (to use for parameters)
- Handles multiple data types
  - So far: *reals* and *vectors*
- Adds lots of functionality
  - Domain knowledge
- See GECCO2012 paper for more details

MT-CGP: Example



MT-CGP

- Has *a big* function set
- Trying to incorporate *domain knowledge*
  - Easy to add new functions to help with a particular problem
- Functions deal with multiple data types
  - Functions are *overloaded*
  - Attempts are made at *human readable consistency*



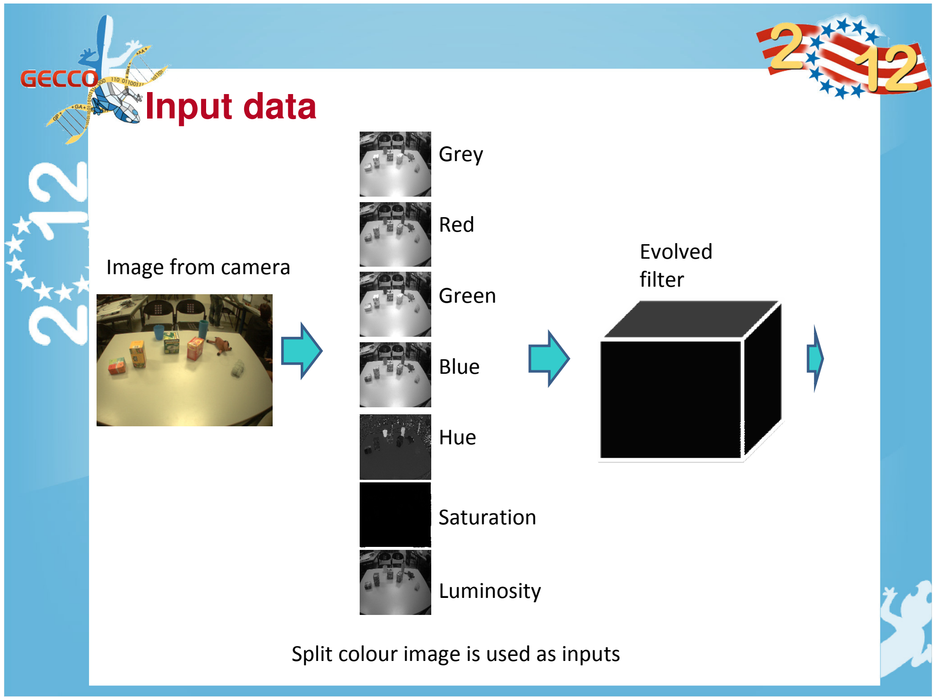
Evolving Image Filters with CGP

- ❖ Detecting/locating objects with the iCub cameras
- ❖ We do this by evolving image filters that take a camera image, and return only the objects we are looking for.

Input

Evolved filter

Target



Input data

Image from camera

Grey

Red

Green

Blue

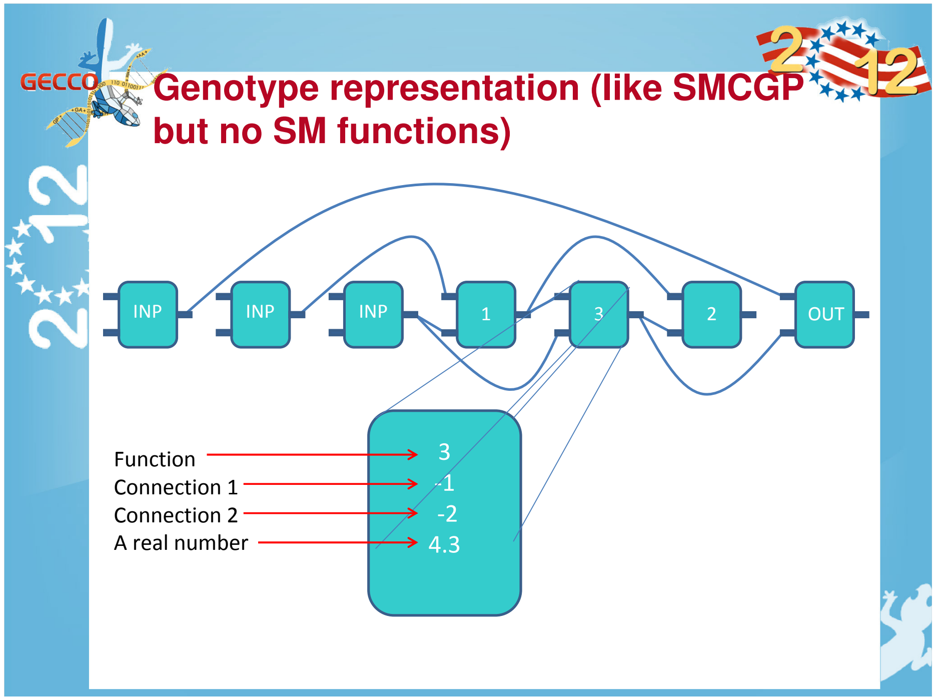
Hue

Saturation

Luminosity

Evolved filter

Split colour image is used as inputs



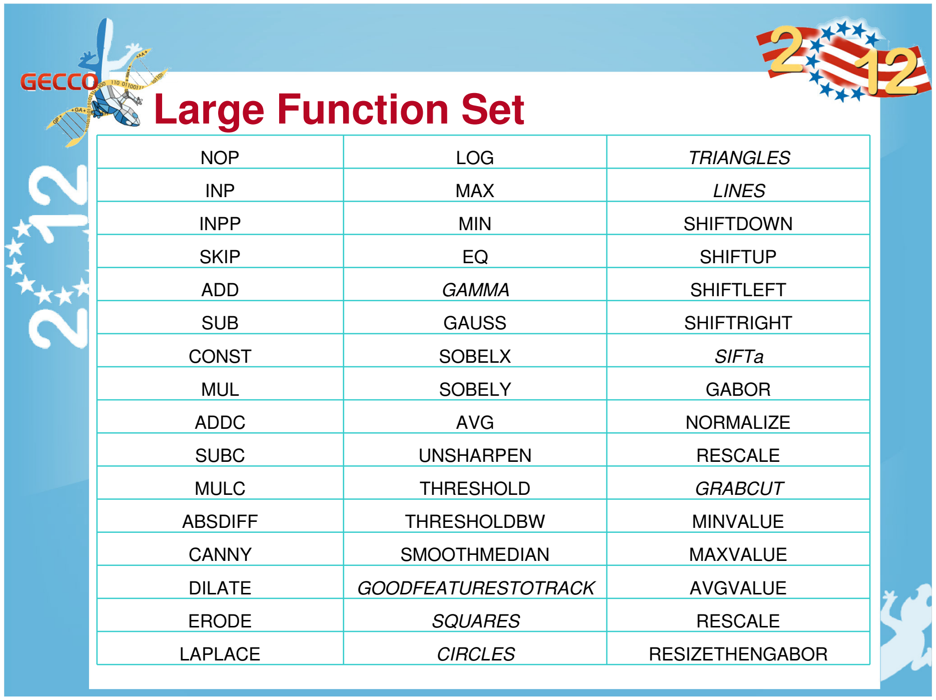
Genotype representation (like SMCGP but no SM functions)

Function → 3

Connection 1 → -1

Connection 2 → -2

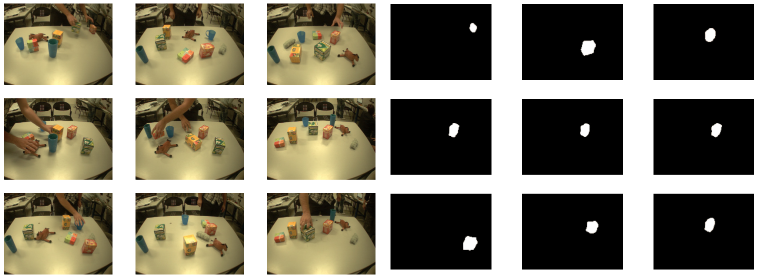
A real number → 4.3



Large Function Set

NOP	LOG	*TRIANGLES*
INP	MAX	*LINES*
INPP	MIN	SHIFTDOWN
SKIP	EQ	SHIFTUP
ADD	*GAMMA*	SHIFTLEFT
SUB	GAUSS	SHIFTRIGHT
CONST	SOBELX	*SIFTa*
MUL	SOBELY	GABOR
ADDC	AVG	NORMALIZE
SUBC	UNSHARPEN	RESCALE
MULC	THRESHOLD	*GRABCUT*
ABSDIFF	THRESHOLDBW	MINVALUE
CANNY	SMOOTHMEDIAN	MAXVALUE
DILATE	*GOODFEATURESTOTRACK*	AVGVALUE
ERODE	*SQUARES*	RESCALE
LAPLACE	*CIRCLES*	RESIZETHENGABOR

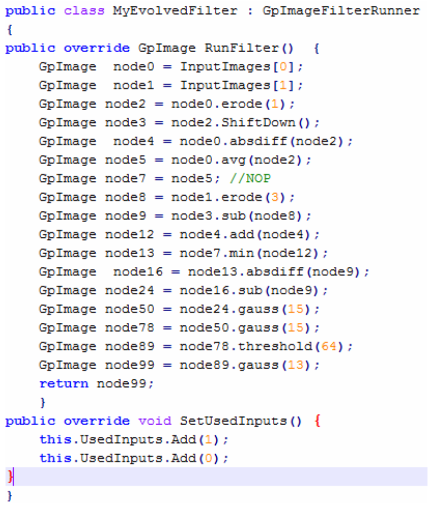
Fitness



•Fitness = sum of mean square error of pixel values between each input/target

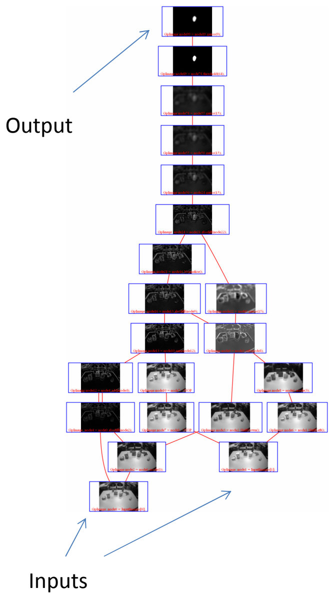
---

Evolved Filter code



```
public class MyEvolvedFilter : GpImageFilterRunner
{
public override GpImage RunFilter()  {
    GpImage  node0 = InputImages[0];
    GpImage  node1 = InputImages[1];
    GpImage node2 = node0.erode(1);
    GpImage node3 = node2.ShiftDown();
    GpImage  node4 = node0.absdiff(node2);
    GpImage node5 = node0.avg(node2);
    GpImage node7 = node5; //NOP
    GpImage node8 = node1.erode(3);
    GpImage node9 = node3.sub(node8);
    GpImage node12 = node4.add(node4);
    GpImage node13 = node7.min(node12);
    GpImage  node16 = node13.absdiff(node9);
    GpImage node24 = node16.sub(node9);
    GpImage node50 = node24.gauss(15);
    GpImage node78 = node50.gauss(15);
    GpImage node89 = node78.threshold(64);
    GpImage node99 = node89.gauss(13);
    return node99;
    }
public override void SetUsedInputs() {
    this.UsedInputs.Add(1);
    this.UsedInputs.Add(0);
}
}
```

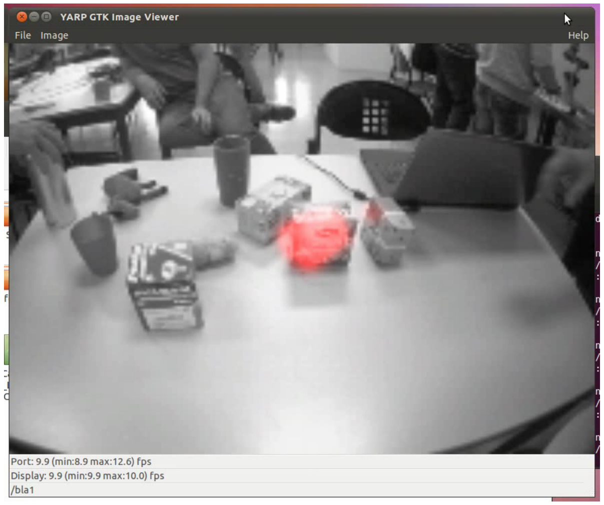
Evolved Filter Dataflow



Things we can do already:

- ❖Generate different filters for other objects.
- ❖Find fast running filters.
- ❖Find them quickly.
- ❖Show that filters are robust.
- ❖Transfer code from offline learning to yarp module.
  - Software emits C# and C++ code
  - Running on Windows/Linux/Mac.

Tea-box filter: demonstration



Port: 9.9 (min:8.9 max:12.6) fps
Display: 9.9 (min:9.9 max:10.0) fps
/bla1

CGP encoded Artificial Neural Networks (CGPANN)

- ❖ CGP has been used to encode both feed-forward ANNs and recursive ANNs. The nodes genes consist of:
  - Connection genes (as usual)
  - Function genes (two)
    - Sigmoid, hyperbolic tangent
  - Weights
    - Each connection gene carries a real-numbered weight
  - Switch genes
    - Binary genes that switch off or on the connection
- ❖ Applied to Markovian and non-Markovian single and double pole-balancing problems
  - Shown to outperform all previously published topology and weights altering evolutionary ANNS (TWEANNs) (*Khan, Khan and Miller 2010*)
- ❖ Breast cancer detection (see GECCO 2012 proceedings)

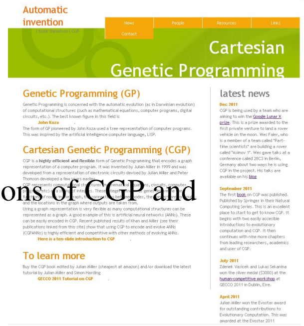
Cyclic CGP

- ❖ When outputs are allowed to connect to inputs through a clocked delay (flip-flop) it is possible to allow CGP to include feedback.
- ❖ By feeding back outputs generated by CGP to an input, it is possible to get CGP to generate sequences
  - In this way iteration is possible
- ❖ There are a couple of recent publications using recursion or iteration in CGP (*Khan, Khan and Miller 2010, Walker, Liu, Tempesti,Tyrrell 2010*)

Applications of CGP

- ❖ Digital Circuit Design
  - ALU, parallel multipliers, digital filters, analogue circuits
- ❖ Mathematical functions
  - Prime generating polynomials
- ❖ Control systems
  - Maintaining control with faulty sensors, helicopter control, general control, simulated robot controller
- ❖ Image processing
  - Image filters
  - Mammary Tumour classification
- ❖ Bio-informatics
  - Molecular Post-docking filters
- ❖ Artificial Neural Networks
- ❖ Developmental Neural Architectures
  - Wumpus world, checkers, maze solving
- ❖ Evolutionary Art
- ❖ Artificial Life
  - Regenerating 'organisms'
- ❖ Optimization problems
  - Applying CGP to solve GA problems

CGP Resources


Automatic invention

I think therefore I CGP

Cartesian Genetic Programming

Genetic Programming (GP)

Genetic Programming is concerned with the automatic evolution (as in Darwinian evolution) of computational structures (such as mathematical equations, computer programs, digital circuits, etc.). The best known figure in this field is **John Koza**.

The form of GP pioneered by John Koza used a tree representation of computer programs. This was inspired by the artificial intelligence computer language, LISP.

Cartesian Genetic Programming (CGP)

CGP is a **highly efficient and flexible** form of Genetic Programming that encodes a graph representation of a computer program. It was invented by Julian Miller in 1999 and was developed from a representation of electronic circuits devised by Julian Miller and Peter Thomson developed a few years earlier.

CGP represents computational structures (mathematical equations, circuits, computer programs etc) as a string of integers. These integers, known as genes determine the functions of nodes in the graph, the connections between nodes, the connections to outputs and the locations in the graph where outputs are taken from.

Using a graph representation is very flexible as many computational structures can be represented as a graph. A good example of this is artificial neural networks (ANNs). These can be easily encoded in CGP. Recent published results of Khan and Miller (see their publications linked from this site) show that using CGP to encode and evolve ANNs (CGPANNs) is highly efficient and competitive with other methods of evolving ANNs.

Here is a ten-slide introduction to CGP.

To learn more

Buy the CGP book edited by Julian Miller (cheapest at amazon) and/or download the latest tutorial by Julian Miller and Simon Harding

GECCO 2011 Tutorial on CGP.

latest news

Dec 2011

CGP is being used by a team who are aiming to win the Google Lunar X prize. This is a prize awarded to the first private venture to land a rover vehicle on the moon. Wes Faler, who is a member of a team called "Part-time scientists" are building a rover called "Asimov 1". Wes gave talks at a conference called 28C3 in Berlin, Germany about two ways he is using CGP in the project. His talks are available on his blog

September 2011

The first book on CGP was published. Published by Springer in their Natural Computing Series. This is an excellent place to start to get to know CGP. It begins with two easily accesible introductions to evolutionary computation and CGP. It then continues with nine more chapters from leading researchers, academics and user of CGP.

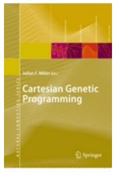
July 2011

Zdenek Vasicek and Lukas Sekanina won the silver medal ($3000) at the human-competitive workshop at GECCO 2011 in Dublin, Eire.

April 2011

Julian Miller won the Evostar award for outstanding contributions to Evolutionary Computation. This was awarded at the Evostar 2011

- ❖ Home site:
  http://www.cartesiangp.co.uk
- ❖ Julian Miller: C implementations of CGP and SMCGP available at
  http://www.cartesiangp.co.uk
- ❖ Simon Harding
  http://www.evolutioninmaterio.com
- ❖ David Oranchak has implemented CGP in Java. Documentation is available at
  http://oranchak.com/cgp/doc/
- ❖ Cartesian Genetic Programming book
  - Published in 2011 by Springer



Conclusions

- ❖ Cartesian Genetic Programming is a graph based GP method capable of representing many computational structures
  - programs, circuits, neural networks, systems of equations…
- ❖ Genetic encoding is compact, simple and easy to implement and can handle multiple outputs easily.
- ❖ The unique form of genetic redundancy in CGP makes mutational search highly effective
- ❖ The effectiveness of CGP has been compared with many other GP methods and it is very competitive

---

CGP Bibliography

Ashmore L. An investigation into cartesian genetic programming within the field of evolutionary art. http://www.emoware.org/evolutionary_art.asp, Department of Computer Science, University of Birmingham (2000)

Clegg J., Walker J. A., Miller J. F. A New Crossover Technique for Cartesian Genetic Programming. Proceedings of Genetic and Evolutionary Computation Conference, ACM Press (2007) 1580-1587.

DiPaola S., Gabora L. Incorporating characteristics of human creativity into an evolutionary art algorithm, Genetic Programming and Evolvable Machines (2009) Vol. 10. For further info see: http://dipaola.org/evolve/

DiPaolo S. Evolving Creative Portrait Painter Programs using Darwinian Techniques with an Automatic Fitness Function. Electronic Visualizationa and the Arts Conference (2005)

Gajda, Z., Sekanina, L.. Gate-Level Optimization of Polymorphic Circuits Using Cartesian Genetic Programming, Proceedings of Congress on Evolutionary Computation. IEEE Press (2009)

Gajda Z., Sekanina, L.. Reducing the Number of Transistors in Digital Circuits Using Gate-Level Evolutionary Design, Proceedings of Genetic and Evolutionary Computation Conference. ACM, (2007) 245-252.

Garmendia-Doval B., Miller J.F., Morley S.D. Post Docking Filtering using Cartesian Genetic Programming. *Genetic Programming Theory and Practice II.* O'Reilly U-M., Yu T., Riolo R., Worzel B. (Eds.). University of Michigan Illinois USA. Springer (2004).

Glette K., Torresen J., Paul Kaufmann P., Platzner., M. A Comparison of Evolvable Hardware Architectures for Classification Tasks. In Proceedings of the 8th International Conference on Evolvable Systems: From Biology to Hardware, Springer LNCS 5216 (2008) 22-33.

Harding S. L., Leitner, J., Schmidhuber, J.. Cartesian Genetic Programming for Image Processing, Genetic Programming Theory and Practice, University of Michigan Illinois USA. Springer. 2012

Harding S. L., Miller J. F. Banzhaf W. Developments in Cartesian Genetic Programming: Self-modifying CGP. Genetic Programming and Evolvable Machines, Vol. 11 (2010)

Harding S. L., Miller J. F. Banzhaf W. Self Modifying Cartesian Genetic Programming: Finding algorithms that calculate pi and e to arbitrary precision, Proceedings of the Genetic and Evolutionary Computation Conference, 2010.

Harding S. L., Miller J. F., Banzhaf W. A Survey of Self-Modifying CGP. Genetic Programming Theory and Practice, Riolo R., (Eds.). University of Michigan Illinois USA. Springer. 2010

Harding S. L., Miller J. F. Banzhaf W. Self Modifying Cartesian Genetic Programming: Parity. Proceedings of Congress on Evolutionary Computation, IEEE Press (2009) 285-292

Harding S. L., Miller J. F. Banzhaf W. Self Modifying Cartesian Genetic Programming: Fibonacci, Squares, Regression and Summing, Proceedings of the 10th European Conference on Genetic Programming, Springer LNCS (2009) 133-144

Harding S. L., Miller J. F., Banzhaf W. Self-Modifying Cartesian Genetic Programming, Proceedings of Genetic and Evolutionary Computation Conference, ACM Press, (2007) 1021-1028.

Harding S., Banzhaf W. Fast Genetic Programming on GPUs. Proceedings of 10th European Conference on Genetic Programming, Springer LNCS 4445 (2007) 90-101

Harding S. L., Miller J. F. Evolution of Robot Controller Using Cartesian Proceedings of the 6th European Conference on Genetic Programming, Springer LNCS 3447 (2005) 62-72.

Hirayama Y., Clarke T, Miller J. F. Fault Tolerant Control Using Cartesian Genetic Programming, Proceedings of Genetic and Evolutionary Computation Conference, ACM Press, (2008) 1523-1530 .

Kalganova T., Miller J. F., Evolving More Efficient Digital Circuits by Allowing Circuit Layout Evolution and Multi-Objective Fitness. Proceedings of the First NASA/DOD Workshop on Evolvable Hardware, IEEE Computer Society (1999) 54-63.

Kalganova T., Miller J. F., Fogarty T. C. Some Aspects of an Evolvable Hardware Approach for Multiple-Valued Combinational Circuit Design Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware. Springer LNCS 1478 (1998) 78-89.

Kaufmann P., Platzner M. Advanced Techniques for the Creation and Propagation of Modules in Cartesian Genetic Programming. Proceedings of the Genetic and Evolutionary Computation Conference, ACM Press, (2008) 1219-1226.

Kaufmann P., Platzner M. MOVES: A Modular Framework for Hardware Evolution. In Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems, IEEE Computer Society Press (2007) 447-454

Kaufmann P., Platzner M. Toward Self-adaptive Embedded Systems: Multiobjective Hardware Evolution. In Proceedings of the 20th International Conference on Architecture of Computing Systems, Springer, LNCS 4415 (2007) 119-208.

M. M. Khan, G. M. Khan, and J. F. Miller, “Efficient representation of recurrent neural networks for markovian/non-markovian non-linear control problems,” in Proceedings of the 10th International Conference on Intelligent Systems Design and Applications (ISDA2010) (2010) 615–620

Khan, G. M., Miller J. F., Khan, M. M. Evolution of Optimal ANNs for Non-Linear Control Problems Using Cartesian Genetic Programming. Proceedings of International Conference on Artificial Intelligence (ICAI 2010)

Khan, G. M., Halliday, D. M., Miller, J. F.,Intelligent agents capable of developing memory of their environment, Angelo Loula A., Queiroz, J. (Eds.) Advances in Modelling Adaptive and Cognitive Systems, Editora UEFS (2010)

Khan G. M., Halliday D. M., Miller J. F. In Search of Intelligent Genes: The Cartesian Genetic Programming Neuron. Proceedings of Congress on Evolutionary Computation, IEEE Press (2009)

Khan G. M., Halliday D. M., Miller J. F. Breaking the synaptic dogma: evolving a neuro-inspired developmental network. Proceedings of 7th International Conference on Simulated Evolution and Learning, LNCS, 5361 (2008) 11-20

Khan G. M., Halliday D. M., Miller J. F. Coevolution of neuro-developmental programs that play checkers. Evolvable Systems: From Biology to Hardware. Springer LNCS 5216 (2008) 352 - 361.

Khan G. M., Halliday D. M., Miller J. F. Coevolution of Intelligent Agents using Cartesian Genetic Programming. Proceedings of Genetic and Evolutionary Computation Conference, ACM Press, (2007) 269-276.

Kuyucu T., Trefzer M. A., Miller J. F., Tyrrell. A. M. On the Properties of Artificial Development and Its Use in Evolvable Hardware. Proceedings of Symposium on Artificial Life , Part of IEEE Symposium on Computational Intelligence, IEEE Press (2009).

Liu H., Miller J. F., Tyrrell A. M. , Intrinsic evolvable hardware implementation of a robust biological development model for digital systems, Proceedings of the NASA/DOD Evolvable Hardware Conference, IEEE Computer Society (2005) 87-92.

Liu H., Miller J. F., Tyrrell A. M. A Biological Development Model for the Design of Robust Multiplier. Applications of Evolutionary Computing: EvoHot 2005, Springer LNCS 3449 (2005) 195-204

Liu H., Miller J. F., Tyrrell A. M. An Intrinsic Robust Transient Fault-Tolerant Developmental Model for Digital Systems. Workshop on Regeneration and Learning in Developmental Systems, Genetic and Evolutionary Computation Conference (2004).

Sekanina, L. Evolvable Components - From Theory to Hardware Implementations, Springer (2003)

Sekanina, L. Image Filter Design with Evolvable Hardware, Proceedings of Evolutionary Image Analysis and Signal Processing, Springer LNCS 2279 (2002) 255-266.

Sekanina, L, Vašíček Z. On the Practical Limits of the Evolutionary Digital Filter Design at the Gate Level, Proceedings of EvoHOT, Springer, LNCS 3907 (2006) 344-355.

Miller J. F. Cartesian Genetic Programming, Springer 2011.

Miller J.F., Smith S.L. Redundancy and Computational Efficiency in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 10 (2006) 167-174.

Miller J. F. Evolving a self-repairing, self-regulating, French flag organism. Proceedings of Genetic and Evolutionary Computation Conference, Springer LNCS 3102 (2004) 129-139.

Miller J. F., Thomson P. Beyond the Complexity Ceiling: Evolution, Emergence and Regeneration. Workshop on Regeneration and Learning in Developmental Systems, Genetic and Evolutionary Computation Conference (2004).

Miller J.F., Banzhaf W., Evolving the Program for a Cell From French Flags to Boolean Circuits. Kumar S., Bentley P. *On Growth, Form and Computers*. Elsevier Academic Press (2003).

Miller J. F., Thomson P. A Developmental Method for Growing Graphs and Circuits. Proceedings of the 5th International Conference on Evolvable Systems: From Biology to Hardware, Springer LNCS 2606 (2003) 93-104.

Miller J. F. Evolving developmental programs for adaptation, morphogenesis, and self-repair. Proceedings of the 7th European Conference on Artificial Life, Springer LNAI 2801 (2003) 256-265.

Miller J. F. What bloat? Cartesian Genetic Programming on Boolean problems. Genetic and Evolutionary Computation Conference, Late breaking paper (2001) 295 - 302.

Miller J. F., Hartmann M. Evolving *messy* gates for fault tolerance: some preliminary findings. Proceedings of the 3rd NASA/DOD Workshop on Evolvable Hardware. IEEE Computer Society (2001) 116-123.

Miller J. F., Hartmann M. Untidy evolution: Evolving *messy* gates for fault tolerance. Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware. Springer LNCS 2210 (2001) 14-25.

Miller J.F., Kalganova T., Lipnitskaya N., Job D. The Genetic Algorithm as a Discovery Engine: Strange Circuits and New Principles. *Creative Evolutionary Systems*. Morgan Kaufmann (2001).

Miller J.F., Job D., Vassilev V.K. Principles in the Evolutionary Design of Digital Circuits - Part I. *Journal of Genetic Programming and Evolvable Machines*, 1 (2000) 8-35.

Miller J.F., Job D., Vassilev V.K. Principles in the Evolutionary Design of Digital Circuits - Part II. *Journal of Genetic Programming and Evolvable Machines*, 3 (2000) 259-288.

Miller J. F., Thomson P. Cartesian Genetic Programming. Proceedings of the 3rd European Conference on Genetic Programming. Springer LNCS 1802 (2000) 121-132.

Miller J. F. On the filtering properties of evolved gate arrays. Proceedings of the First NASA/DOD Workshop on Evolvable Hardware. IEEE Computer Society (1999) 2-11.

Miller J. F. Digital Filter Design at Gate-level using Evolutionary Algorithms. Proceedings of the 1st Genetic and Evolutionary Computation Conference. Morgan Kaufmann (1999) 1127-1134.

Miller J. F. An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming Approach. Proceedings of the 1st Genetic and Evolutionary Computation Conference. Morgan Kaufmann (1999) 1135-1142.

Miller J. F. Evolution of Digital Filters using a Gate Array Model. Proceedings of the First Workshop on Image Analysis and Signal Processing. Springer LNCS 1596 (1999) 17-30.

Miller J. F., Kalganova T., Lipnitskaya N., Job D. The Genetic Algorithm as a Discovery Engine: Strange Circuits and New Principles. Proceedings of the workshop on the AISB Symposium on Creative Evolutionary Systems. AISB (1999) 65-74.

Miller J. F., Thomson P. Aspects of Digital Evolution: Evolvability and Architecture. Proceedings of The Fifth International Conference on Parallel Problem Solving from Nature. Springer LNCS 1498 (1998) 927-936.

Miller J. F., Thomson P. Aspects of Digital Evolution: Geometry and Learning. Proceedings of the 2nd International Conference on Evolvable Systems: From Biology to Hardware. Springer LNCS 1478 (1998) 25-25.

Miller J. F., Thomson P. Evolving Digital Electronic Circuits for Real-Valued Function Generation using a Genetic Algorithm . Proceedings of the 3rd Conference on Genetic Programming. Morgan Kaufmann (1998) 863-868.

Miller J.F., Thomson P., Fogarty T.C. Designing Electronic Circuits Using Evolutionary Algorithms: Arithmetic Circuits: A Case Study. *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications*. Quagliarella, D., Periaux J., Poloni C., Winter G. (Eds.). Wiley (1997)

Payne, A. J., Stepney, S.. Representation and Structural biases in CGP, Proceedings of Congress on Evolutionary Computation, IEEE Press (2009)

Rothermich J., Wang F., Miller J. F. Adaptivity in Cell Based Optimization for Information Ecosystems. Proceedings of the Congress on Evolutionary Computation. IEEE Press (2003) 490-497.

Rothermich J., Miller J. F. Studying the Emergence of Multicellularity with Cartesian Genetic Programming in Artificial Life. Proceedings of the 2002 U.K. Workshop on Computational Intelligence (2002).

Seaton, T., Brown G., Miller J. F.., Analytic Solutions to Differential Equations under Graph-based Genetic Programming. Proceedings of the 13th European Conference on Genetic Programming. Springer LNCS 6021 (2010) 232-243

Vašíček Z, Sekanina L. Hardware Accelerators for Cartesian Genetic Programming, In: Eleventh European Conference on Genetic Programming, Springer (2008) 230-241

Vašíček, Z. Sekanina, L.. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. Genetic Programming and Evolvable Machines, 12(3) (2011) 305-327, 2011.

Vassilev V. K., Miller J. F. Scalability Problems of Digital Circuit Evolution. Proceedings of the 2nd NASA/DOD Workshop on Evolvable Hardware. IEEE Computer Society (2000) 55-64.

Vassilev V. K., Miller J. F. The Advantages of Landscape Neutrality in Digital Circuit Evolution. Proceedings of the 3rd International Conference on Evolvable Systems: From Biology to Hardware. Springer LNCS 1801 (2000) 252-263.

Vassilev V. K., Miller J. F. Towards the Automatic Design of More Efficient Digital Circuits. Proceedings of the 2nd NASA/DOD Workshop on Evolvable Hardware. IEEE Computer Society (2000) 151-160.

Vassilev V. K., Miller J. F., Fogarty T. C. Digital Circuit Evolution and Fitness Landscapes. Proceedings of the Congress on Evolutionary Computation. IEEE Press (1999) 1299-1306.

Vassilev V. K., Miller J. F., Fogarty T. C. On the Nature of Two-Bit Multiplier Landscapes. Proceedings of the First NASA/DOD Workshop on Evolvable Hardware. IEEE Computer Society (1999) 36-45.

Voss M. S. Social programming using functional swarm optimization. In Proceedings of IEEE Swarm Intelligence Symposium (2003)

Voss M. S., Howland, J. C. III.Financial modelling using social programming. Financial Engineering and Applications (2003)

Völk K., Miller J. F., Smith, S. L. Multiple Networks CGP for the Classification of Mammograms. Proceedings of the 11th European Workshop on Image Analysis and Signal Processing (EvoIASP), Springer LNCS (2009).

Walker J. A., Liu Y., Tempesti G., Tyrrell A. M., “Automatic Code Generation on a MOVE Processor Using Cartesian Genetic Programming,” in Proceedings of the International Conference on Evolvable Systems: From Biology to Hardware, Springer LNCS vol. 6274 (2010) 238–249

Walker J.A., Völk, K. , Smith, S. L., Miller, J. F. Parallel evolution using multi-chromosome cartesian genetic programming, Genetic Programming and Evolvable Machines, 10 (4), (2009) pp 417-445

Walker J. A., Hilder, J. A., Tyrrell. A. M. Towards Evolving Industry-feasible Intrinsic Variability Tolerant CMOS Designs, Proceedings of Congress on Evolutionary Computation, IEEE Press (2009)

Walker J.A., Miller J.F. The Automatic Acquisition, Evolution and Re-use of Modules in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 12 (2008) 397-417.

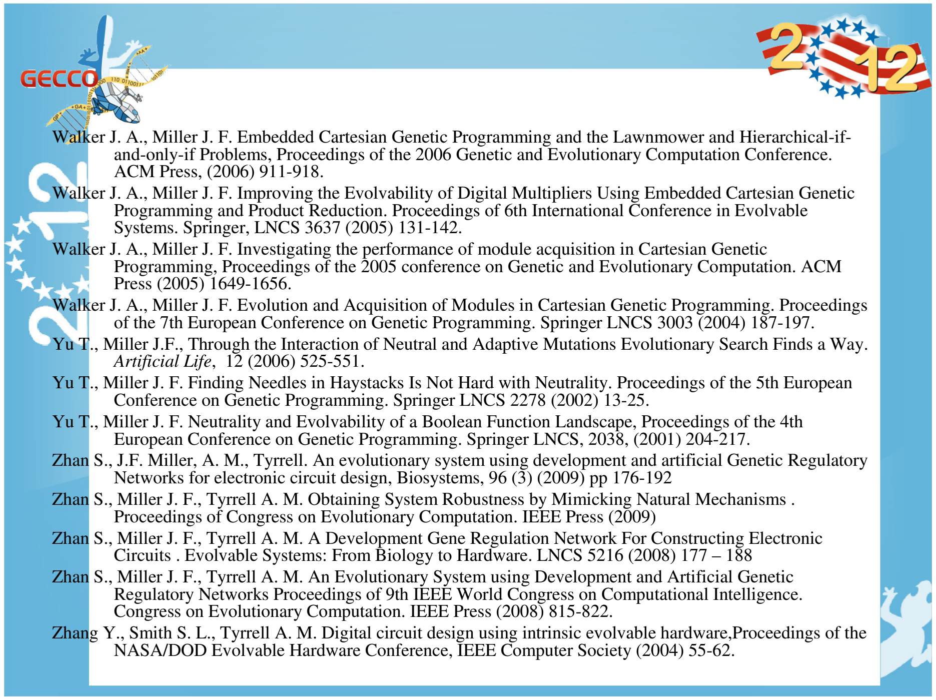
Walker J. A. Modular Cartesian Genetic Programming. PhD thesis, University of York, 2008.

Walker J. A., Miller J. F. Solving Real-valued Optimisation Problems using Cartesian Genetic Programming. Proceedings of Genetic and Evolutionary Computation Conference, ACM Press (2007) 1724-1730.

Walker J. A., Miller J. F. Changing the Genospace: Solving GA Problems using Cartesian Genetic Programming, Proceedings of 10th European Conference on Genetic Programming, Springer LNCS 4445 (2007) 261-270.

Walker J. A., Miller J. F. Predicting Prime Numbers using Cartesian Genetic Programming, Proceedings of 10th European Conference on Genetic Programming. Springer LNCS 4445, (2007) 205-216

Walker J. A., Miller J. F., Cavill R. A Multi-chromosome Approach to Standard and Embedded Cartesian Genetic Programming, Proceedings of the 2006 Genetic and Evolutionary Computation Conference. ACM Press, (2006) 903-910.



Walker J. A., Miller J. F. Embedded Cartesian Genetic Programming and the Lawnmower and Hierarchical-if-and-only-if Problems, Proceedings of the 2006 Genetic and Evolutionary Computation Conference. ACM Press, (2006) 911-918.

Walker J. A., Miller J. F. Improving the Evolvability of Digital Multipliers Using Embedded Cartesian Genetic Programming and Product Reduction. Proceedings of 6th International Conference in Evolvable Systems. Springer, LNCS 3637 (2005) 131-142.

Walker J. A., Miller J. F. Investigating the performance of module acquisition in Cartesian Genetic Programming, Proceedings of the 2005 conference on Genetic and Evolutionary Computation. ACM Press (2005) 1649-1656.

Walker J. A., Miller J. F. Evolution and Acquisition of Modules in Cartesian Genetic Programming. Proceedings of the 7th European Conference on Genetic Programming. Springer LNCS 3003 (2004) 187-197.

Yu T., Miller J.F., Through the Interaction of Neutral and Adaptive Mutations Evolutionary Search Finds a Way. *Artificial Life*, 12 (2006) 525-551.

Yu T., Miller J. F. Finding Needles in Haystacks Is Not Hard with Neutrality. Proceedings of the 5th European Conference on Genetic Programming. Springer LNCS 2278 (2002) 13-25.

Yu T., Miller J. F. Neutrality and Evolvability of a Boolean Function Landscape, Proceedings of the 4th European Conference on Genetic Programming. Springer LNCS, 2038, (2001) 204-217.

Zhan S., J.F. Miller, A. M., Tyrrell. An evolutionary system using development and artificial Genetic Regulatory Networks for electronic circuit design, Biosystems, 96 (3) (2009) pp 176-192

Zhan S., Miller J. F., Tyrrell A. M. Obtaining System Robustness by Mimicking Natural Mechanisms . Proceedings of Congress on Evolutionary Computation. IEEE Press (2009)

Zhan S., Miller J. F., Tyrrell A. M. A Development Gene Regulation Network For Constructing Electronic Circuits . Evolvable Systems: From Biology to Hardware. LNCS 5216 (2008) 177 – 188

Zhan S., Miller J. F., Tyrrell A. M. An Evolutionary System using Development and Artificial Genetic Regulatory Networks Proceedings of 9th IEEE World Congress on Computational Intelligence. Congress on Evolutionary Computation. IEEE Press (2008) 815-822.

Zhang Y., Smith S. L., Tyrrell A. M. Digital circuit design using intrinsic evolvable hardware,Proceedings of the NASA/DOD Evolvable Hardware Conference, IEEE Computer Society (2004) 55-62.