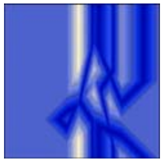
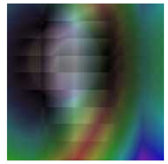


Cartesian Genetic Programming



Julian Francis Miller
Dept of Electronics
University of York, UK
jfm7@ohm.york.ac.uk

Simon Harding
Dept of Computer Science
Memorial University of Canada
slh@evolutioninmaterio.com

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-131-6/08/07.

1

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…

2

Cartesian Genetic Programming (CGP)

- Grew out of work in the evolution of digital circuits, Miller and Thomson 1997. First 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)
- The genotype is a list of integers that represent the program primitives and how they are connected together
  - CGP represents programs as *graphs* in which there are *non-coding genes*

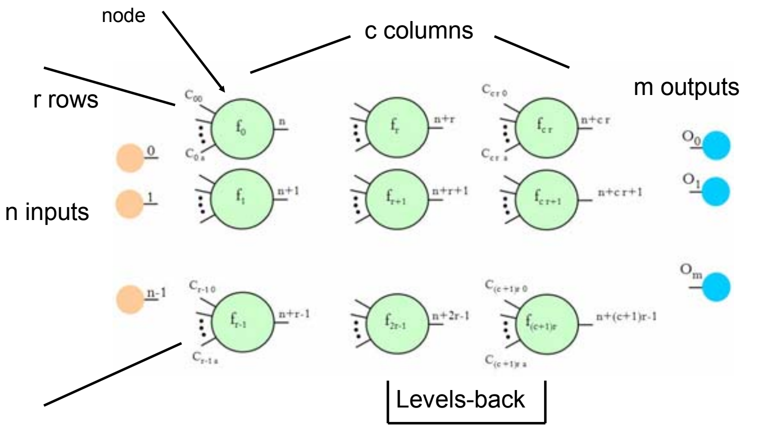
3

Types of CGP

- Classic
- Modular
- Self-modifying
- Developmental
- Cyclic

4

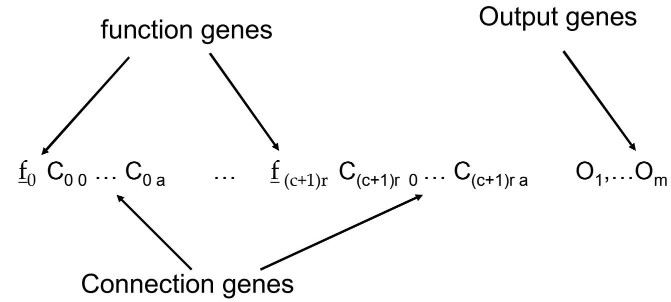
CGP General form



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

5

CGP genotype

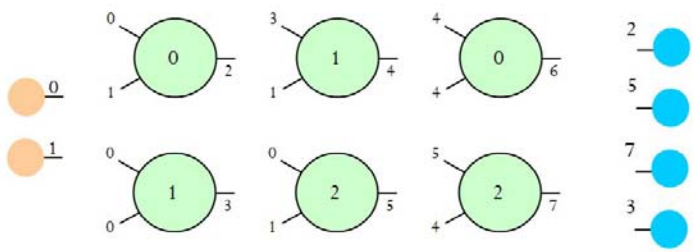


Usually, all functions have as many inputs as *maximum* function arity

Unused connections are ignored (see later)

6

Example

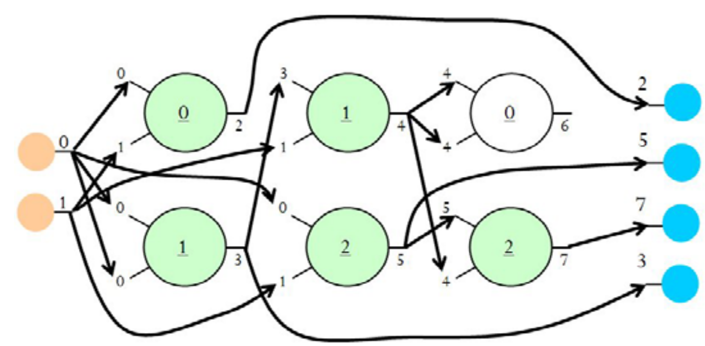


Encoding of graph as a list of integers (i.e. the genotype)

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

7

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

8

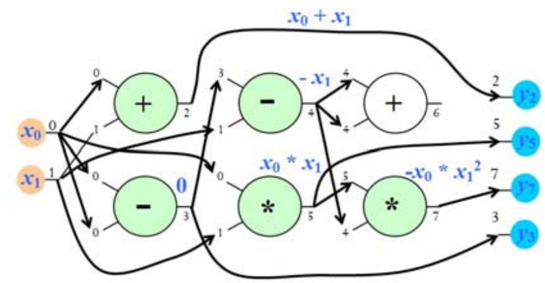
Example: Function look up table

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)

9

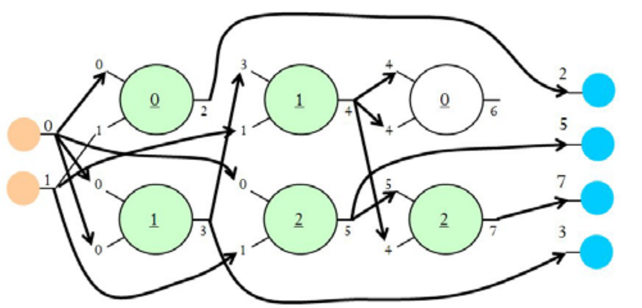
So what does the graph represent?



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

10

What happened to the node whose output label is 6?



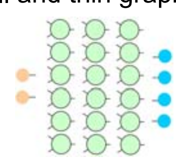
The node was not used so the genes are *silent* or *non-coding*

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

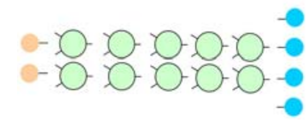
11

The role of the geometric parameters: rows, columns and level-back

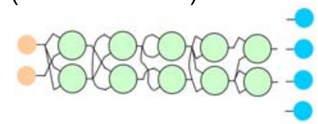
Tall and thin graphs



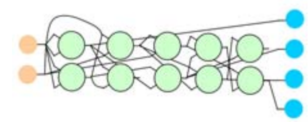
Short and wide graphs



Layered graphs (levels-back =1)



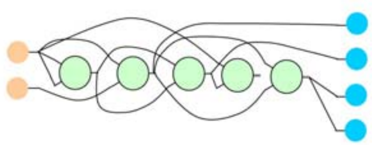
Less layered graphs (levels-back =3)



12

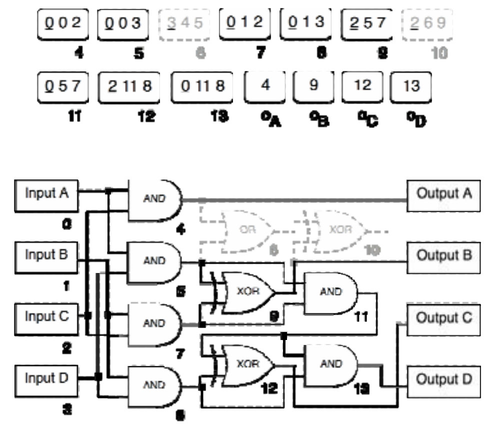
Types of graphs easily controlled

- 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

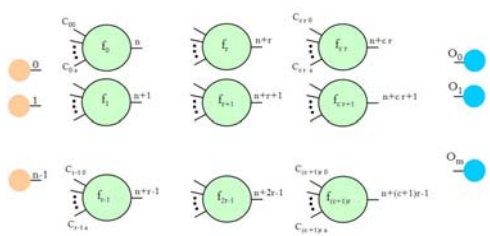


13

Arbitrary directed graph CGP Example



Allelic constraints



All function genes f_i must takes allowed function alleles

$$0 \le f_i \le n_f$$

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

$j \ge l$ $\quad n + (j-l)r \le C_{ij} \le n + jr$

$j < l$ $\quad 0 \le C_{ij} \le n + jr$

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

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

15

Non-coding genes in CGP

- Contains active and inactive regions (rather than coding or non-coding)
- Mutations can make active genes become inactive and inactive genes become active
- A single gene change can thus cause large phenotypic changes
- When a gene is changed by mutation several things can happen

16

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 (as in slide 14)

```
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
}
```

17

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 – see later (Walker, Miller Cavill 2006)
- Recent work using a floating point representation of CGP has suggested that crossover might be useful (Clegg, Walker, Miller 2007)

18

Program changes caused by mutations

Gene was	Gene is	Genotypic change	Phenotypic change	Fitness change
silent	silent	Yes	No	No
active	silent	Yes	Yes	Likely
silent	active	Yes	Yes	Likely
active	active	Yes	Yes	Likely

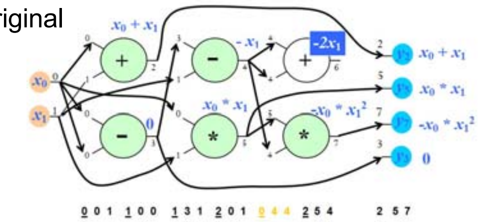
When genetic changes occur without any fitness change it is often referred to a *neutral* change.

The very interesting aspect is that in CGP most neutral change occurs externally to the phenotype, so it does not have to be processed in any fitness calculation (unlike many other forms of GP)

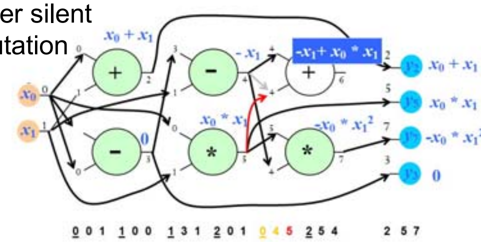
19

Silent mutations and their effects

Original



After silent mutation

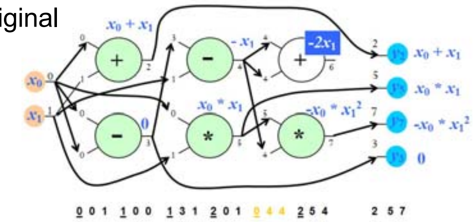


No change in phenotype but it changes the programs *accessible* through *subsequent* mutational change

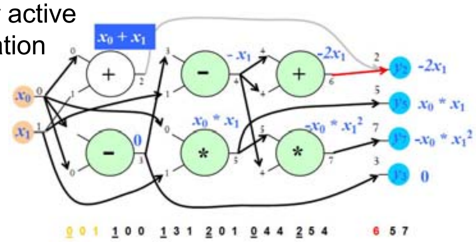
20

Non-silent mutations and their effects

Original



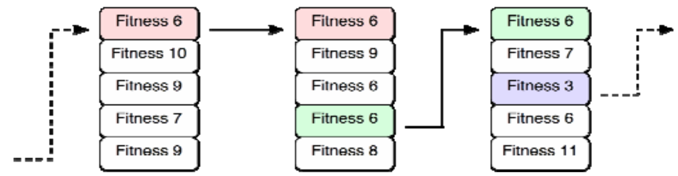
After active mutation



Massive change in phenotype is possible through simple mutation

21

Evolutionary Strategy



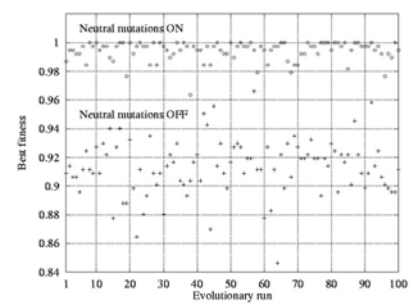
- CGP uses a variant of (1 + 4) Evolutionary Strategy
  - However, an offspring is always chosen if it *is equally as fit* or has better fitness than the parent

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)

23

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

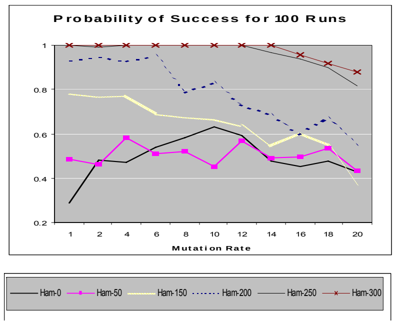


Importance of neutral search can be demonstrated by looking at the success rate in evolving a correct three-bit digital parallel multiplier circuit.

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

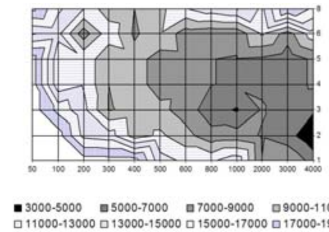
24

Effectiveness of Neutral Search as a function of mutation rate and Hamming bound (Yu and Miller 2001)

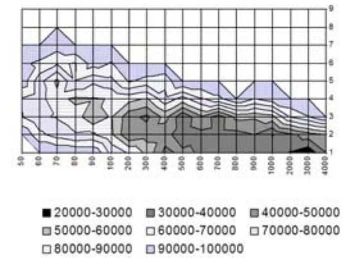


- Hamming Distance H(g,h)
  g1=213 012 130 432 159
  g2=202 033 132 502 652
  hamming distance H(g1,g2)=9.
- If genotypes are selected so that $H(g_{new},g_{old}) = 0$. No neutral drift is permitted.
- If genotypes are selected so that $H(g_{new},g_{old}) = \text{length}(g)$. Any amount of neutral drift is permitted.

Computational effort versus Genotype length and mutation rate



Even-3 parity



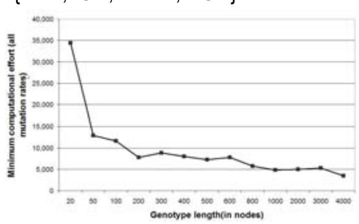
Two-bit multiplier

Evolutionary search is most effective at low mutation rates and large genotype lengths. The larger the genotype length, the lower should be the value chosen for mutation rate

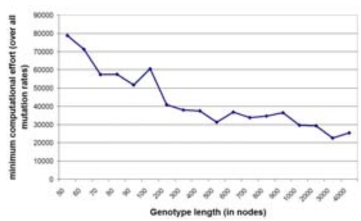
26

Minimum Computational Effort (over all mutation rates) versus genotype length (in nodes)

Even 3 parity with gate set {AND, OR, NAND, NOR}.



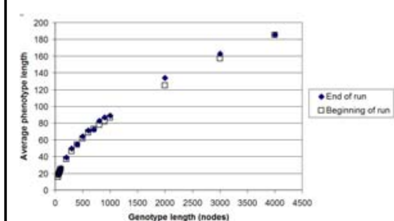
Two-bit multiplier with gate set {AND, OR, NAND, NOR}.



So provided you choose the 'best' mutation rate, problems are more easily solved with large genotypes. However big genotypes does NOT mean big phenotypes (programs)….

27

Phenotype length versus genotype length (two-bit multiplier)

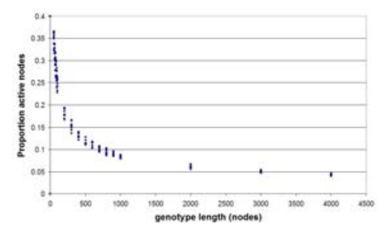


Average phenotype length for the initial population contrasted with the average phenotype length at conclusion of evolutionary run versus genotype length with 1% mutation

NO BLOAT

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!!



28

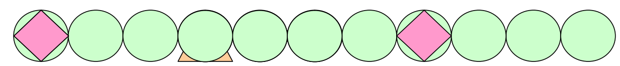
Modular/Embedded CGP (Walker, Miller 2004)

- 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

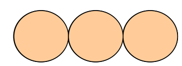
29

MCGP Example

Genotype



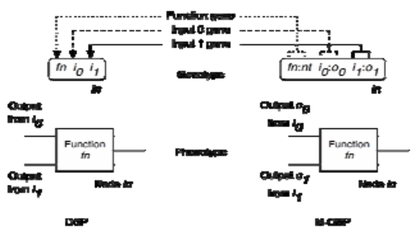
Module List



Module

30

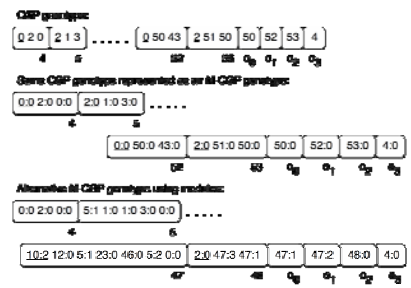
Representation Modification 1



- 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

31

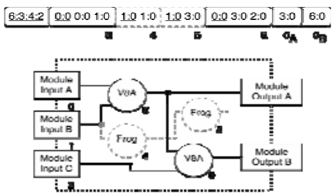
Representation Modification 2



- 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

32

Modules

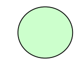


- Same characteristics as M-CGP
  - Bounded variable length genotype
  - Bounded variable length phenotype
- Modules also contain inactive genes as in CGP
- Modules can not contain other modules!

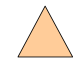
33

Node Types

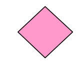
- Three node types:
  - Type 0
    - Primitive function



  - Type I
    - Module created by compress operator



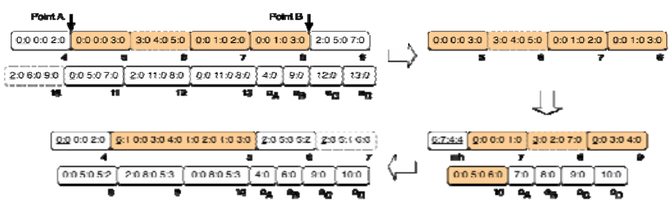
  - Type II
    - Module replicated by genotype point-mutation



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

34

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

35

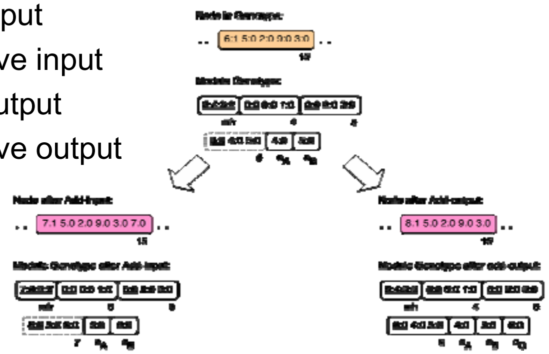
Module Survival

- 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

36

Evolving a Module I

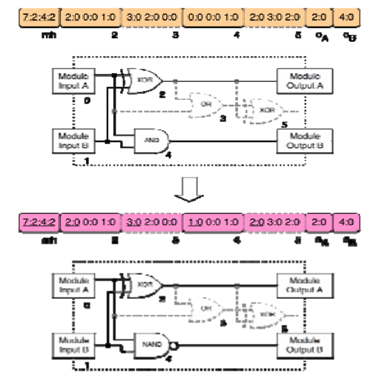
– Structural mutation
- Add input
- Remove input
- Add output
- Remove output



37

Evolving a Module II

– Module point-mutation operator
- Restricted version of genotype point-mutation operator
- Only uses primitive functions



38

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

39

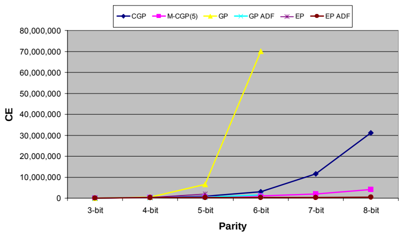
Experimental parameters

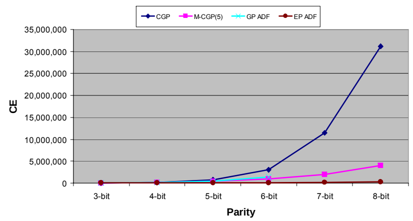
Parameter	Value
Population size	5
Initial genotype size	100 nodes (300 genes)
Genotype point mutation rate	3% (9 genes)
Genotype point mutation probability	1
Compress/Expand probability ◊	0.1/0.2
Module point mutation probability ◊	0.04
Add/Remove input probability ◊	0.01/0.02
Add/Remove output probability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

NOTES: ◊ these parameters only apply to Modular (Embedded) CGP

The results are heavily dependent on the maximum number of nodes allowed. Much better results are obtained when larger genotype lengths are used.

Even Parity Results

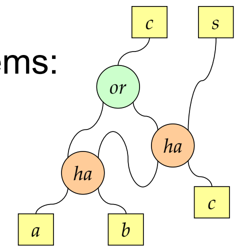




41

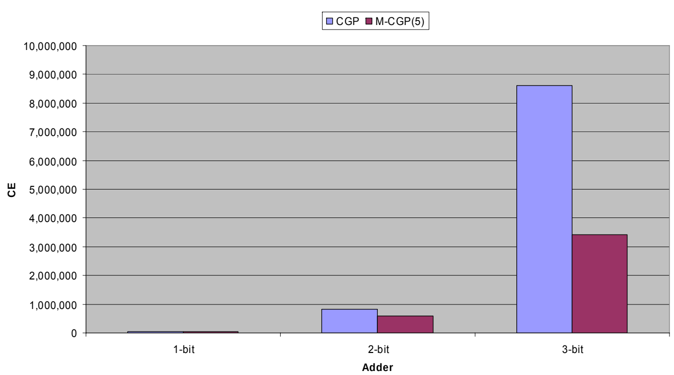
Digital Adder

- Three digital adder problems:
  - 1-bit, 2-bit, and 3-bit
- Function set:
  - AND, NAND, OR, NOR
- Fitness Function:
  - Number of phenotype output bits that differ from the perfect *n*-bit digital adder solution
  - Perfect solution has a fitness of zero



42

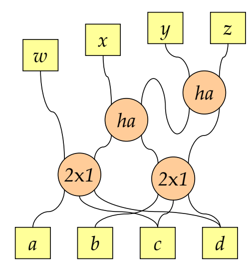
Adder Results



43

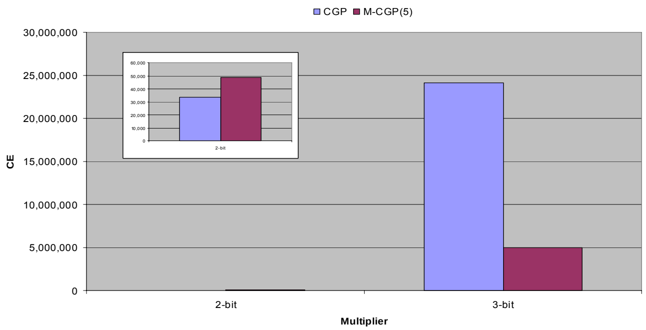
Digital Multiplier

- Two digital multiplier problems:
  - 2-bit and 3-bit
- Function set:
  - AND, AND (on 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



44

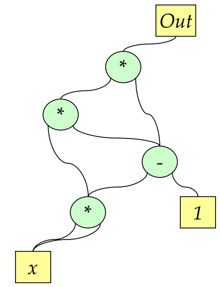
Multiplier Results



45

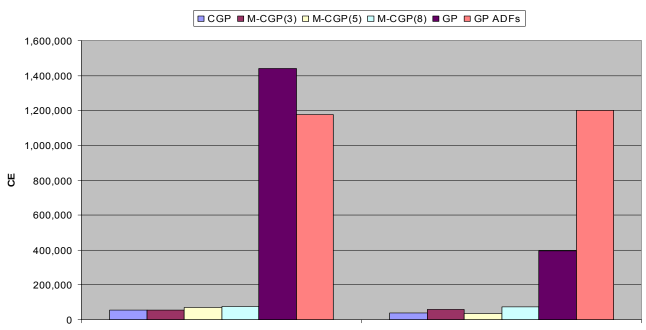
Symbolic Regression

- 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



46

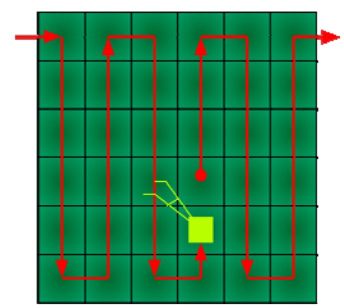
Symbolic Regression Results



47

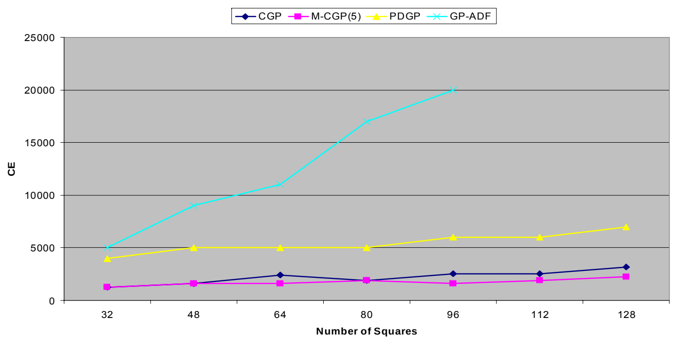
Lawnmower Problem

- Guide a lawnmower around a lawn cutting the grass
  - Lawn divided into *n* x *m* squares
  - Cuts all the grass in a square, when the square is visited
  - Starts in the centre square
  - If the lawnmower leaves one side, it reappears on the opposite side
- Problem solved when all squares have been visited



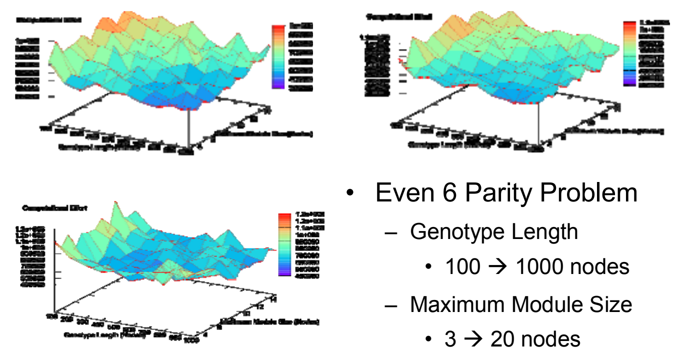
48

Lawnmower Problem Results



49

Parameter Sweeps



- Even 6 Parity Problem
  - Genotype Length
    - 100 → 1000 nodes
  - Maximum Module Size
    - 3 → 20 nodes
  - Mutation Rate
    - 2% → 4%

Multi-chromosome Approach

- A multi-chromosome genotype is divided up into *n* equal length sections called "chromosomes"
  - Each chromosome contains an equal number of nodes
- The no. of chromosomes (*n*) is dictated by the no. of outputs of the given problem
  - Each chromosome has a single output
- The entire problem is still represented in a single genotype

Multi-chromosome CGP Example

- A node in a chromosome can connect to:
  - A program input
  - The output of a previous node in the SAME chromosome
- Creates a form of compartmentalisation in the genotype
  - Removes any connections between the smaller problems in each chromosome

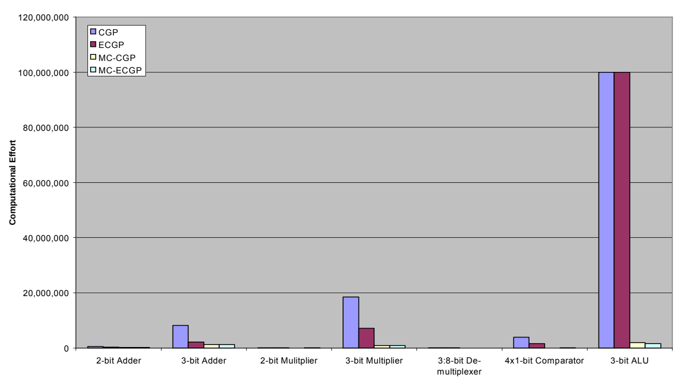
Multi-chromosome (1 + 4) ES

- Calculate fitness for each chromosome
- Select best chromosome from each position
  - If tied, choose offspring over parent
- Promoted individual consists of the best chromosomes
  - May not have been in the original population
  - May have a higher overall fitness than parents

Multi-chromosome experiments and Parameters

- Adder †
  - 2-bit (3 chromosomes)
  - 3-bit (5 chromosomes)
- Multiplier *
  - 2-bit (4 chromosomes)
  - 3-bit (6 chromosomes)
- De-multiplexer †
  - 3:8-bit (8 chromosomes)
- Comparator †
  - 4 x 1-bit (18 chromosomes)
- Arithmetic Logic Unit *
  - 3-bit (17 chromosomes)
- Each chromosome contained 100 nodes (300 genes)
- Function set 1 (*)
  - AND, AND (one input inverted), XOR, OR
- Function set 2 (†)
  - AND, NAND, OR, NOR

Results



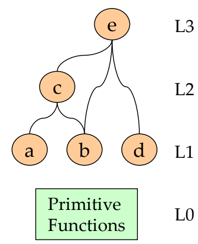
Modules within Modules?

- Currently only allow primitive functions in modules
  - Single level hierarchy
- Allow modules within modules
  - Multi-level hierarchy
  - Produce larger building blocks
  - Improve performance
  - Evolve solutions to larger, more complex problems
- ADFs occur inside ADFs in GP, why not have modules inside modules?

56

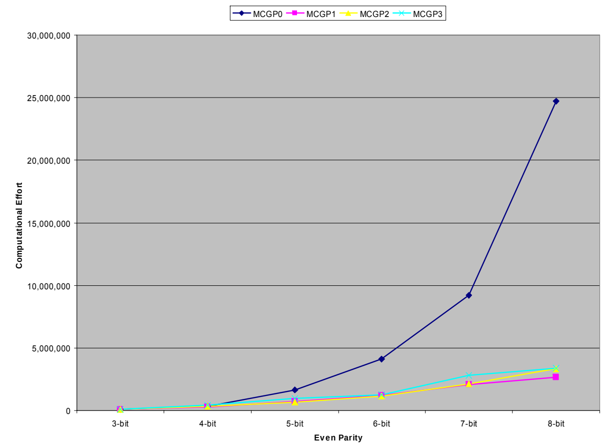
Multi-level Hierarchy (Walker 2008)

- Introduce level types into modules
  - L1 → primitive functions
  - L2 → primitive functions, L1 modules
  - L3 → primitive functions, L1 and L2 modules
  - etc…
- Each level is created by the compress operator
  - User sets the number of levels in the hierarchy
- Nodes in a module can only be mutated to functions with a lower level type
- Nodes in a genotype can be mutated to a function of any level

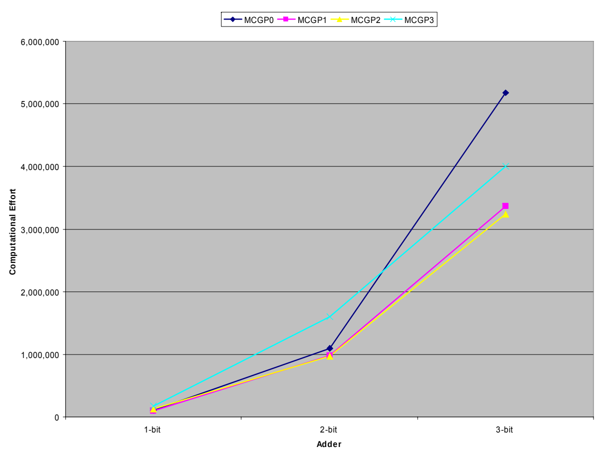


57

Multi-level modular CGP Even Parity



Multi-level modular CGP Adder



Self-modifying CGP (Harding, Miller and Banzhaf 2007)

- SMCGP is a form of developmental CGP
- To be more specific: a form of genetic programming where an individual's phenotype can vary over time
  - It is iterated

60

Representational differences

- In CGP nodes connect explicitly
  - i.e This node connects to node 12.
- In SMCGP nodes have a relative address
  - i.e. This node connects to one 4 nodes back.
  - Useful for moving pieces of cgp code around
- CGP node :
  - function & connections
- SMCGP node :
  - function, connections & 3 parameters.

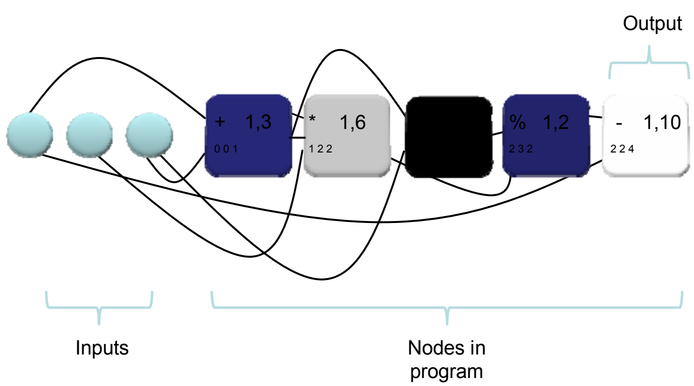
61

Other representational differences

- Input/Outputs handled differently.
  - In SMCGP typically the last N-nodes in the graph are used as output nodes
- If a node addresses a node of a negative index, then this is mapped to an input (using modulo arithmetic)

62

Visualization



63

Self-modification

- In addition to functional nodes, SMCGP contains nodes that modify its own graph
  - For example, a function may add, delete or move a section of the program
- Self modification nodes pass the larger numerical input unchanged
- Phenotype is initially the same as the genotype graph, however
  - It is iterated, which with modification, causes it to diverge from the original graph

64

Self-modification process

1. Evaluate CGP graph
   - Get computational output
2. If a node is a modification node and it is activated, add to 'ToDo' list
   - Activated means: If the first input is greater or equal to value to the second input
3. When finished evaluating entire graph, parse 'ToDo' list.
4. Perform each operation to build modified graph for next iteration

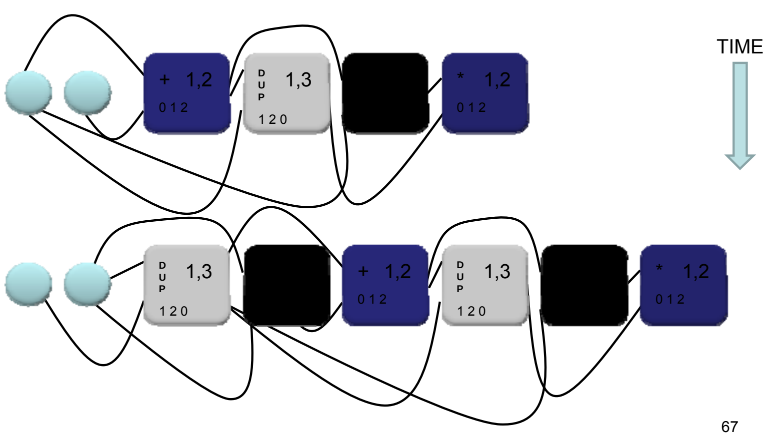
65

Some SMCGP operators

Operator	Parameters	Function
MOVE	Start, End, Insert	Moves each of the nodes between Start and End into the position specified by Insert
DUPE	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
ADD	Insert, Count	Adds Count number of NOP nodes at position Insert
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
CHP	Node, Parameter, New Value	Changes the specified parameter and a given node
FLR		Clears any entries in the pending modifications list
OVR	Start, End, Insert	Moves each of the nodes between Start and End into the position specified by Insert, overwriting existing nodes
DU2	Start, End, Insert	Similar to DUPE, but connections are considered to absolute, rather than relative

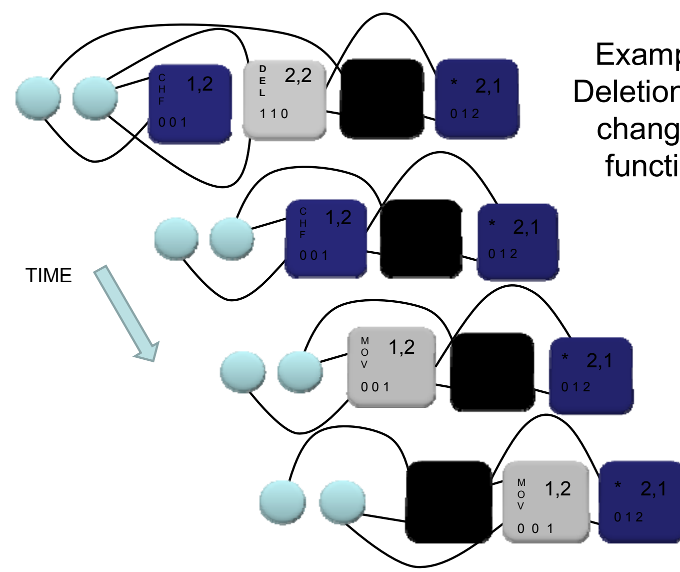
66

Example: Duplication



67

Example: Deletion and changing function



68

Modules in SMCGP

Operator	Parameters	Function
PRC	Start, End	Executes the nodes specified as a procedure.

- A special function can call another part of the graph as a procedure
- This section of graph could be made up of active nodes, or nodes neutral to the main graph
- Procedures can call other procedures.
- Procedures can self modify
- Inputs to this procedure are the inputs to the calling node

69

Example: Generating Sequences

- We limit the functional nodes to + and –
- The task is on each iteration (0,1,2,…) to produce the next number in a sequence
  - Here, we ask for the squares: 1,4,9,16,25 etc.
- The only input was the iteration, i
- Fitness calculated by iterating from 0 to 9 and counting the longest sequence from zero that were correct
- Without self-modification this task is *impossible*

70

Squares program

Input, *i*	**Evolved program**	**Output**
0	0 + *i*	0
1	0 + *i*	1
2	0 + *i*+ *i*	4
3	0 + *i* + *i* + *i*	9
4	0 + *i* + *i*+ *i*+ *i*	16

71

Evolving Digital Circuits

- In this experiment we tackled the well known problem of evolving circuits for solving even–parity
- We used a restricted function set, that is well studied in the literature

  AND OR NAND NOR
- This set of functions make the problem *very hard* to solve

72

Evolving Parity Circuits

Number of inputs	4	5	6	7	8
SMCGP	28811	58194	191493	352901	583712
Speedup compared with MCGP	2.27	3.13	1.44	0.76	0.5
Speedup over CGP	2.84	5.04	4.88	8.53	10.13

73

Evolving big parity functions

- The largest parity problem solved to date with a direct GP approach appears to be 22 inputs
  - Although general solutions have been found.
  - The function set used includes many more bit-wise operations – including XOR
- We attempted to produce a solution that could be iterated to find *any* size circuit

74

Evolving big parity functions

- The challenge:
  - Evolve an SMCGP program that solves 2-input even parity. After iterating the growth algorithm once, it should solve 3-input. After a second time, 4-input and so on
- We were able to evolve (and test) to 24 bit input. We think it is a general solution but haven't verified this yet

75

SMCGP Conclusions

- Discussed a promising new variant of CGP that bridges the divide between artificial developmental systems and genetic programming
- This is done by directly producing a phenotype capable of performing a computation
- We have shown we can solve problems that cannot be solved by a conventional GP system.
- In other experiments we have shown that performance appears to be similar on problems where there is no inherent advantage for the self-modification

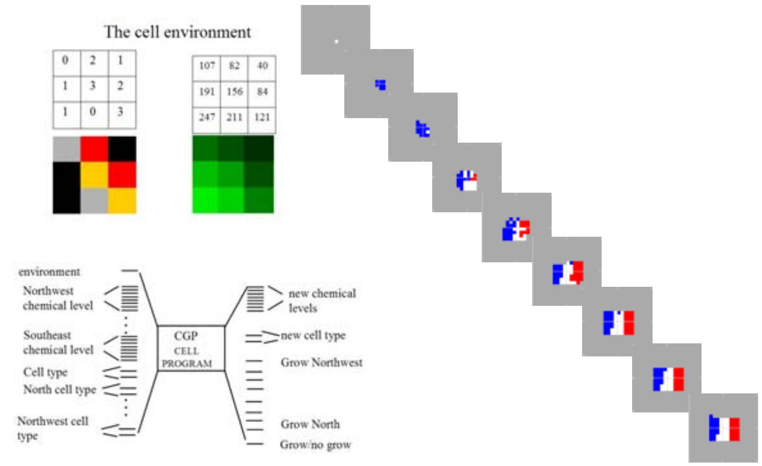
76

Developmental CGP

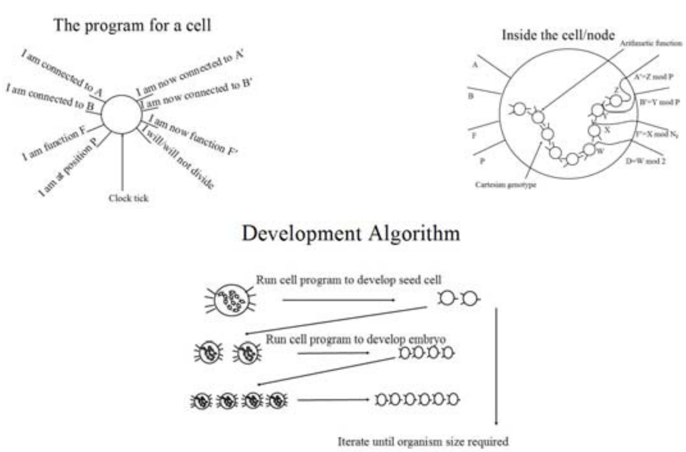
- Various types of CGP inspired by biological development, graph re-writing and neuro-develop have been devised
  - Biological developmental (Miller 2003, 2004)
  - Graph re-writing (Miller 2003)
  - Neuro-developmental (Khan, Miller and Halliday 2007, 2008)

77

Bio-inspired developmental CGP

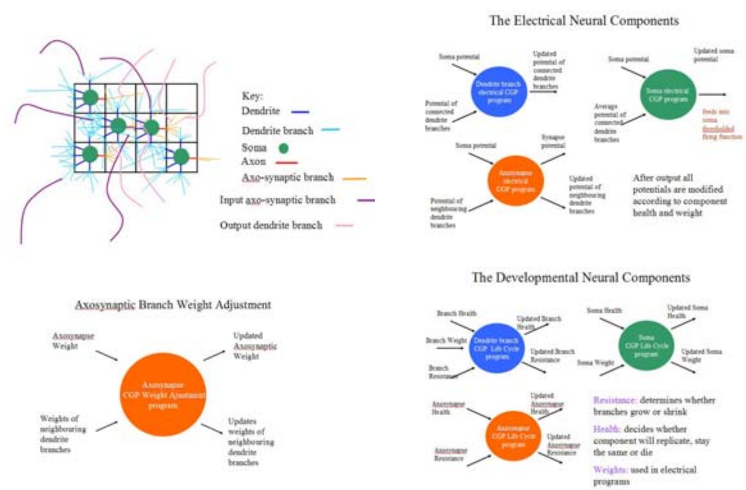


Graph-rewriting CGP



79

Neuro-inspired developmental CGP



Cyclic CGP

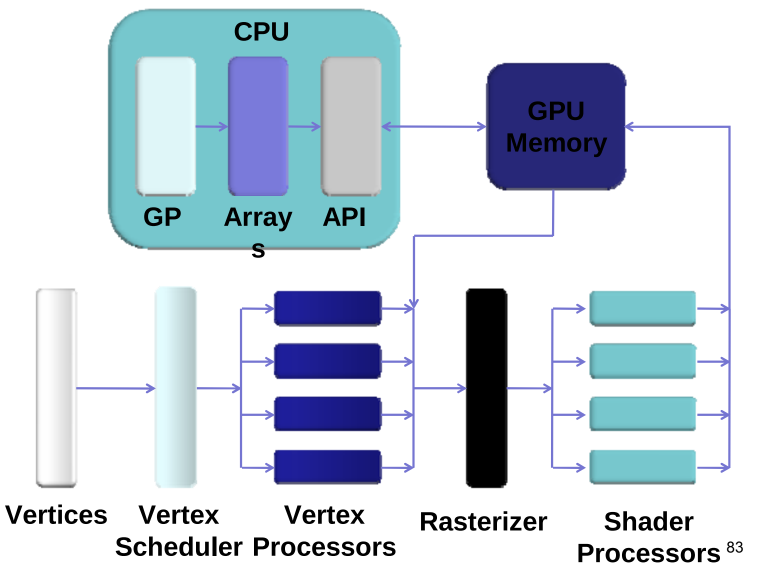
- When outputs are allowed to connect to inputs through a clocked delay (flip-flop) it is possible to allow CGP to include feedback.
- By feeding back outputs generated by CGP to an input, it is possible to get CGP to generate sequences

81

GPU Implementation (Harding and Banzhaf 2007)

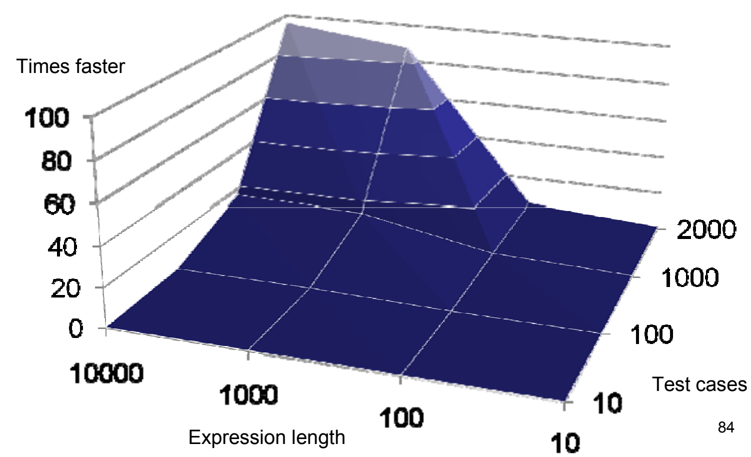
- A guaranteed maximum program length makes it easy to use CGP on more limited platforms.
- We have developed a version of CGP that runs on Graphics Processing Units
  - Limited program length
  - Memory constraints
  - But fast, parallel architecture

82



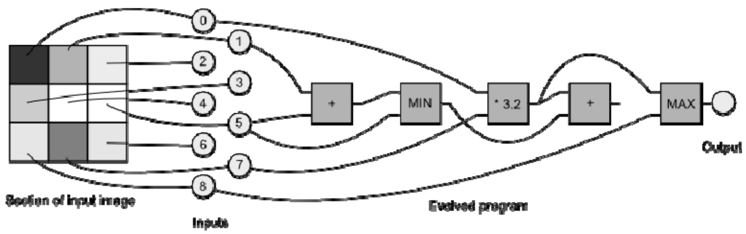
83

GPU Speed up on a regression problem



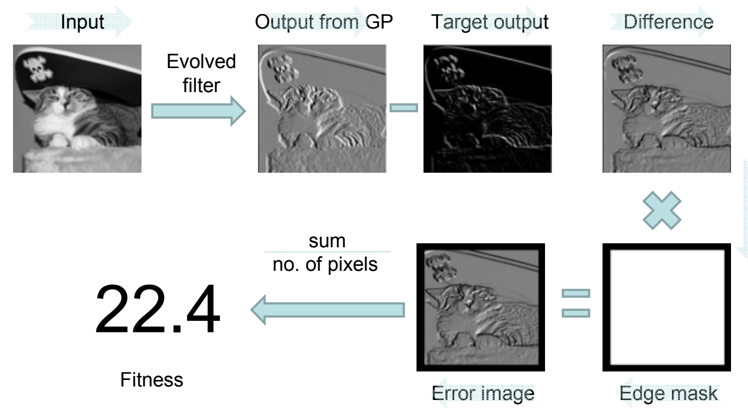
84

Image Processing



85

Image processing



86

Image Processing : Example of Evolved Filters

Sobel filter

Evolved filter

Target filter

87

Applications of CGP

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

88

CGP Web Resources

- Home site:
  http://www.cartesiangp.co.uk
- Julian Miller:
  http://www.elec.york.ac.uk/intsys/users/jfm7/
- Simon Harding:
  http://www.evolutioninmaterio.com/
  http://www.gpgpgpu.com

89

Conclusions

- Cartesian Genetic Programming is a graph based GP method
- 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
- The CGP method is still being developed (i.e. modular CGP, self-modifying CGP, neuro-developmental CGP)
- A method has been developed for CGP to output lists of numbers so that it can be applied to any problem that genetic algorithms can be applied to (see Walker and Miller 2007)

90

References

1. Walker J. A. Modular Cartesian Genetic Programming. PhD thesis, University of York, 2008.
2. Walker J.A., Miller J.F. The Automatic Acquisition, Evolution and Re-use of Modules in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, (2008) in press.
3. Hirayama Y., Clarke T, Miller J. F. Fault Tolerant Control Using Cartesian Genetic Programming, Proceedings of Genetic and Evolutionary Computation Conference, ACM Press, (2008) see this conference.
4. Khan G. M., Miller J. F., Halliday D. M. Co-evolution of neuro-developmental programs that play checkers, Proceedings of Genetic and Evolutionary Computation Conference, ACM Press, (2008) see this conference.
5. 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.
6. 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.
7. 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.
8. 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.
9. Walker J. A., Miller J. F. Predicting Prime Numbers using Cartesian Genetic Programming, Proceedings of 10th European Conference on Genetic Programming (EuroGP 2007), Springer LNCS 4445 (2007) 205-216

91

References

10. Harding S., Banzhaf W. Fast Genetic Programming on GPUs. Proceedings of 10th European Conference on Genetic Programming (EuroGP 2007), Springer LNCS 4445 (2007) 90-101
11. Walker J. A., Miller J. F. Changing the Genospace: Solving GA Problems using Cartesian Genetic Programming, Proceedings of 10th European Conference on Genetic Programming (EuroGP 2007), Springer LNCS 4445 (2007) 261-270.
12. Zbyšek G., Lukáš S. Reducing the Number of Transistors in Digital Circuits Using Gate-Level Evolutionary Design, Proceedings of Genetic and Evolutionary Computation Conference (GECCO2007), ACM, (2007) 245-252.
13. DiPaola S., Gabora L. Incorporating characteristics of human creativity into an evolutionary art algorithm, Late Breaking papers at Genetic and Evolutionary Computation Conference, ACM Press, (2007) 2450-2456. For further info see: http://dipaola.org/evolve/
14. Yu T., Miller J.F., Through the Interaction of Neutral and Adaptive Mutations Evolutionary Search Finds a Way. *Artificial Life*, 12 (2006) 525-551.
15. Miller J.F., Smith S.L. Redundancy and Computational Efficiency in Cartesian Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 10 (2006) 167-174.
16. 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 (GECCO 2006), ACM Press, (2006) 903-910.
17. 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 (GECCO 2006), ACM Press, (2006) 911-918.
18. Lukáš S., Vašíček Zdeněk V. On the Practical Limits of the Evolutionary Digital Filter Design at the Gate Level, Proceedings of EvoHOT, Springer, LNCS 3907 (2006) 344-355.

92

References

19. 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 (ICES 2005), Springer, LNCS 3637 (2005) 131-142.
20. 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.
21. 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 (GECCO 2005), ACM Press (2005) 1649-1656.
22. Harding S. L., Miller J. F. Evolution of Robot Controller Using Cartesian Proceedings of the 6th European Conference on Genetic Programming (EuroGP 2005) Springer LNCS 3447 (2005) 62-72.
23. 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
24. DiPaolo S. Evolving Creative Portrait Painter Programs using Darwinian Techniques with an Automatic Fitness Function. Electronic Visualizationa and the Arts Conference (2005)
25. 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).
26. 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).
27. 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.

93

References

28. Miller J. F. Evolving a self-repairing, self-regulating, French flag organism. Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2004), Springer LNCS 3102 (2004) 129-139.
29. Walker J. A., Miller J. F. Evolution and Acquisition of Modules in Cartesian Genetic Programming. Genetic Programming 7th European Conference, EuroGP 2004, Proceedings. Springer LNCS 3003 (2004) 187-197.
30. 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).
31. Rothermich J., Wang F., Miller J. F. Adaptivity in Cell Based Optimization for Information Ecosystems. Proceedings of the 2003 Congress on Evolutionary Computation (CEC03) IEEE Press (2003) 490-497.
32. 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.
33. 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.
34. 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).
35. Lukáš S. Evolvable Components - From Theory to Hardware Implementations, Berlin, Springer, 2003, ISBN 3-540-40377-9

94

References

36. Voss, Mark S. (2003). Social programming using functional swarm optimization. In Proceedings of *IEEE Swarm Intelligence Symposium (SIS03)*.
37. Voss, Mark S. and James C. Howland, III (2003). Financial modelling using social programming. In *FEA 2003: Financial Engineering and Applications*, Banff, Alberta.
38. 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).
39. Yu T., Miller J. F. Finding Needles in Haystacks Is Not Hard with Neutrality. Proceedings of the 5th European Conference on Genetic Programming (EuroGP2002), Springer LNCS 2278 (2002) 13-25.
40. Lukáš S. Image Filter Design with Evolvable Hardware, Proceedings of Evolutionary Image Analysis and Signal Processing (EvoIASP2002), Springer LNCS 2279 (2002) 255-266.
41. Yu T., Miller J. F. Neutrality and Evolvability of a Boolean Function Landscape, Proceedings of the 4th European Conference on Genetic Programming (EuroGP2001). Springer LNCS, 2038, (2001) 204-217.
42. Miller J. F., Hartmann M. Evolving *messy* gates for fault tolerance: some preliminary findings. Proceedings of the 3rd NASA/DOD Workshop on Evolvable Hardware (EH'01). IEEE Computer Society (2001) 116-123.
43. 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.

95

References

44. Miller J. F. What bloat? Cartesian Genetic Programming on Boolean problems. Genetic and Evolutionary Computation Conference, Late breaking paper (2001) 295 - 302.
45. 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).
46. 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.
47. 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.
48. 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.
49. 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.
50. Miller J. F., Thomson P. Cartesian Genetic Programming. Proceedings of the 3rd European Conference on Genetic Programming. Springer LNCS 1802 (2000) 121-132.
51. 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.

96

References

52. 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)
53. Miller J. F. Evolution of Digital Filters using a Gate Array Model. Proceedings of the First EvoIASP'99 Workshop on Image Analysis and Signal Processing. Springer LNCS 1596 (1999) 17-30.
54. Miller J. F. Digital Filter Design at Gate-level using Evolutionary Algorithms. Proceedings of the 1st Genetic and Evolutionary Computation Conference (GECCO'99). Morgan Kaufmann (1999) 1127-1134.
55. 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 (GECCO'99). Morgan Kaufmann (1999) 1135-1142.
56. Miller J. F. On the filtering properties of evolved gate arrays. Proceedings of the First NASA/DOD Workshop on Evolvable Hardware (EH'99). IEEE Computer Society (1999) 2-11.
57. 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 (EH'99). IEEE Computer Society (1999) 36-45.
58. 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 (CES'99) (1999) 65-74.
59. 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.

97

References

60. 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 (EH'99). IEEE Computer Society (1999) 54-63.
61. Miller J. F., Thomson P. Evolving Digital Electronic Circuits for Real-Valued Function Generation using a Genetic Algorithm . Proceedings of the Third Annual Conference on Genetic Programming. Morgan Kaufmann (1998) 863-868.
62. Miller J. F., Thomson P. Aspects of Digital Evolution: Evolvability and Architecture. Proceedings of The Fifth International Conference on Parallel Problem Solving from Nature (PPSNV). Springer LNCS 1498 (1998) 927-936.
63. 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) 25-35.
64. 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.
65. 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)

98