Cartesian Genetic Programming *for* Evolution and Approximation *of* Digital Circuits

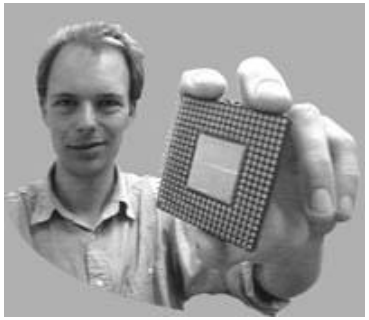
Lukáš Sekanina
Faculty of Information Technology
Brno University of Technology
sekanina@fit.vutbr.cz

IT4Innovations
national
supercomputing
center

Prague, MFF UK
November 5, 2015

Why to Evolve Designs?

New designs



"Evolutionary algorithms in practice can produce designs that are beyond the scope of conventional methods and are, in some sense, better."

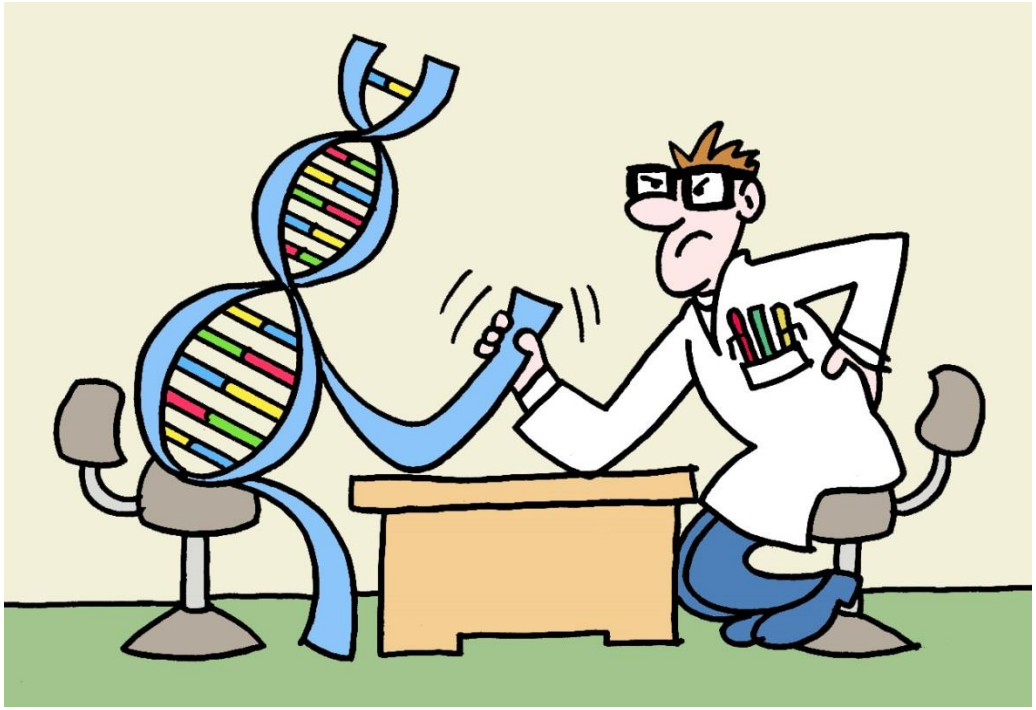
[Thompson et al: IEEE Trans. on Evol. Comp, 1999]

Adaptive systems

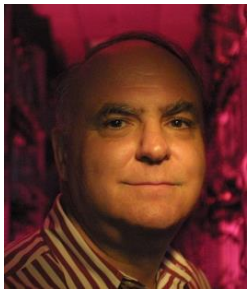
"The challenge of conventional design is replaced with that of designing an evolutionary process that automatically performs the design in our place. *This may be harder than doing the design directly*, but makes autonomy possible."

[Adrian Stoica, NASA JPL, 2004]

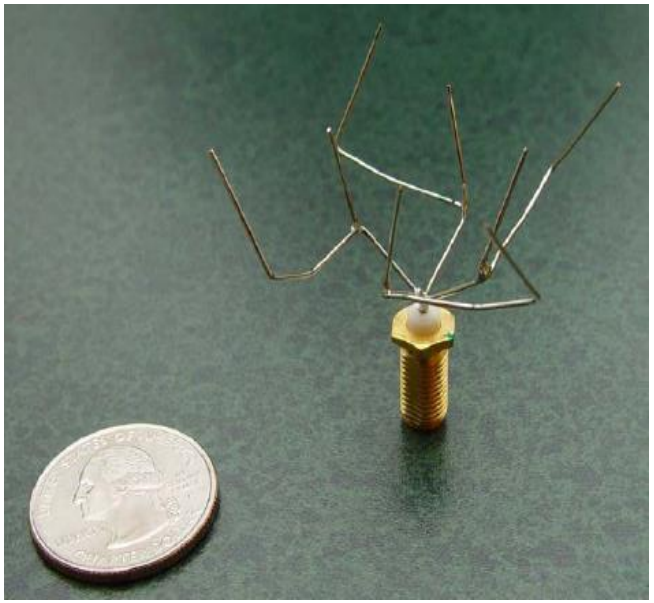
Human-Competitive Results Competition at GECCO



http://www.genetic-programming.org/combined.php



J. R. Koza



Humies at GECCO since 2004

J. Koza: We say that an automatically created result is "human-competitive" if it satisfies one or more of the eight criteria below.

- **(A)** The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.
- **(B)** The result is equal to or better than a result that was accepted as a new scientific result at the time when it was published in a peer-reviewed scientific journal.
- **(C)** The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.
- **(D)** The result is publishable in its own right as a new scientific result *independent* of the fact that the result was mechanically created.
- **(E)** The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.
- **(F)** The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.
- **(G)** The result solves a problem of indisputable difficulty in its field.
- **(H)** The result holds its own or wins a regulated competition involving human contestants (in the form of either live human players or human-written computer programs).

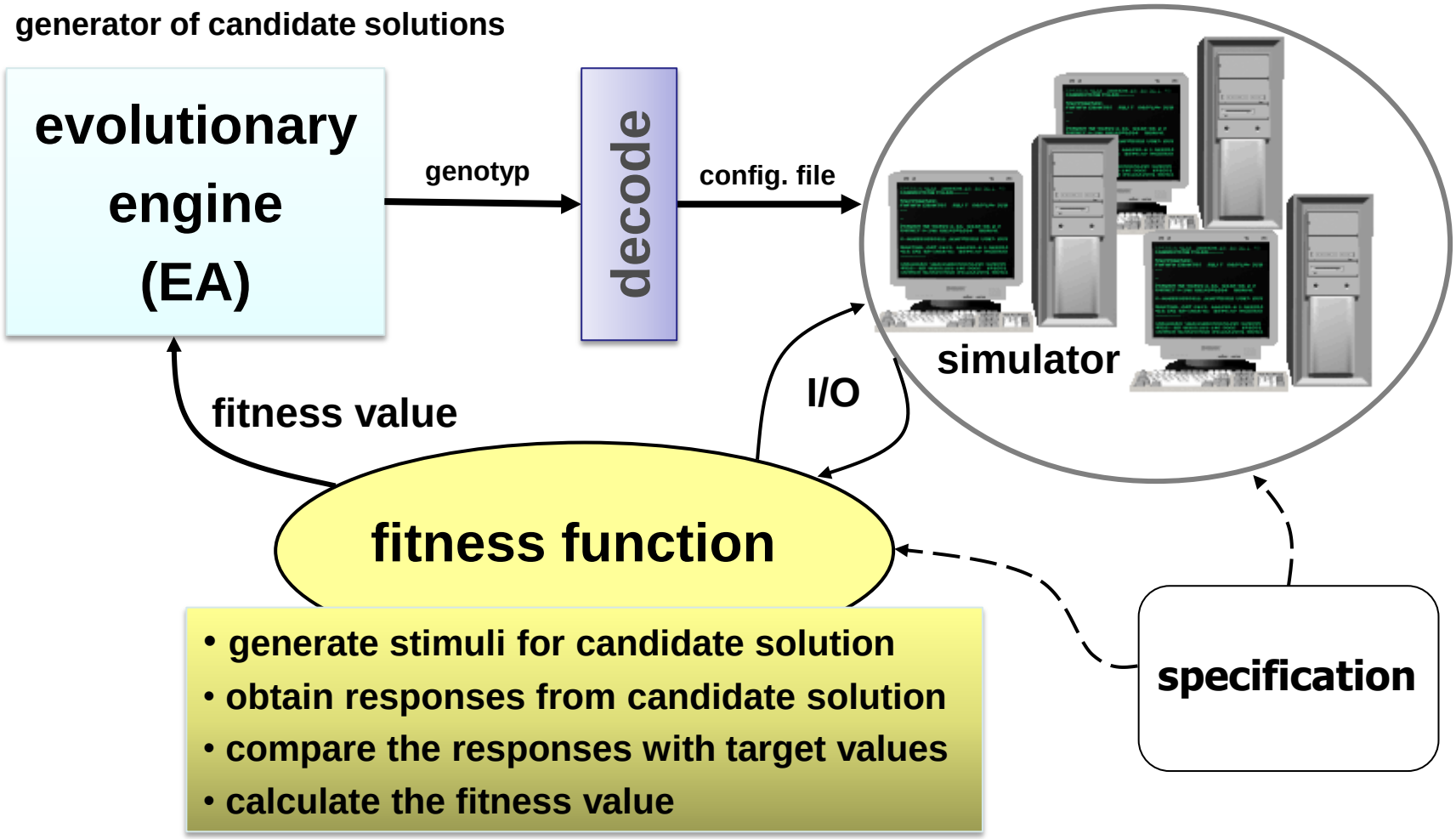
Winners of Humies 2004 - 2015

Year	Title	Institution
2004	An Evolved Antenna for Deployment on NASA's ST 5 Mission	NASA AMES
	Automatic Quantum Computer Programming: A GP Approach	Hampshire Coll., US
2005	Two-dimensional photonic crystals designed by evolutionary algorithms	Cornell U., US
	Shaped-pulse optimization of coherent soft-x-rays	Colorado S. Univ., US
2006	Catalogue of Variable Freq. and Single-Resistance-Controlled Oscillators	MIT, US
2007	Evol. Design of Single-Mode Microstructured Polymer Optical Fibers	UCL, UK
2008	Genetic Programming for Finite Algebras	Hampshire Coll., US
2009	A Genetic Programming Approach to Automated Software Repair	U. of New Mexico, US
2010	Evol. design of the energy function for protein structure prediction	U. of Nottingham, UK
2011	GA-FreeCell: Evolving Solvers for the Game of FreeCell	Ben-Gurion U., IL
2012	Evolutionary Game Design	ICL, UK
2013	Evolutionary Design of FreeCell Solvers	Ben-Gurion U., IL
	Search for a grand tour of the Jupiter Galilean moons	ESA
2014	Genetic Algorithms for Evolving Computer Chess Programs	Bar-Ilan University, IL
2015	Evolutionary Approach to Approximate Digital Circuits Design	Brno U. of Tech., CZ

Outline

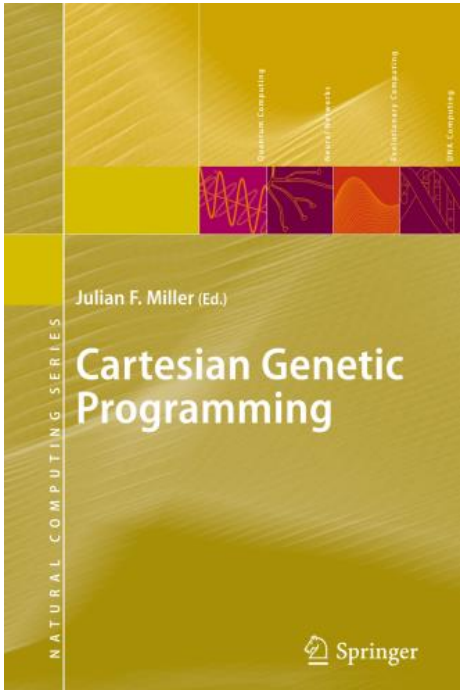
- Cartesian Genetic Programming (CGP)
- Formal methods in the fitness function
  - SAT-based approach
  - BDD-based approach
- Evolutionary design of image filters
- Approximate computing using CGP
- Conclusions

Evolutionary design



Cartesian Genetic Programming [Miller 1999]

- Cartesian Genetic Programming (CGP) is a form of Genetic Programming (GP)
  - GP: candidate program ~ syntactic tree (J. Koza)
  - CGP: candidate program ~ directed acyclic graph
- Features of CGP
  - genetic encoding is compact and simple
  - mutation-based search
  - easy to implement
- Implementations
  - standard CGP, modular CGP, self-modifying CGP, multichromosome CGP, multiobjective CGP
- CGP website
  - http://www.cartesiangp.co.uk/



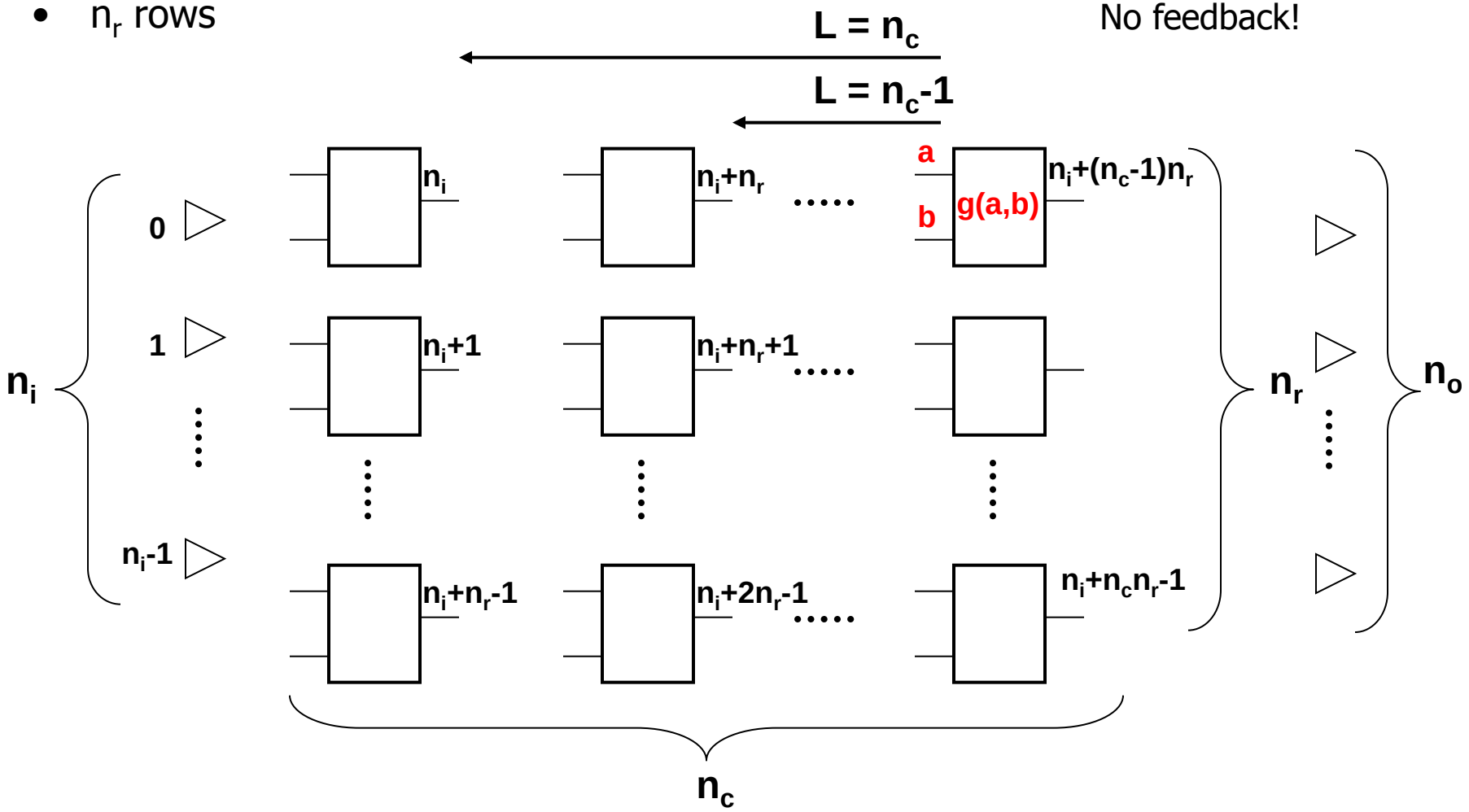
CGP: Standard form

- n_i primary inputs
- n_o primary outputs
- n_c columns
- n_r rows

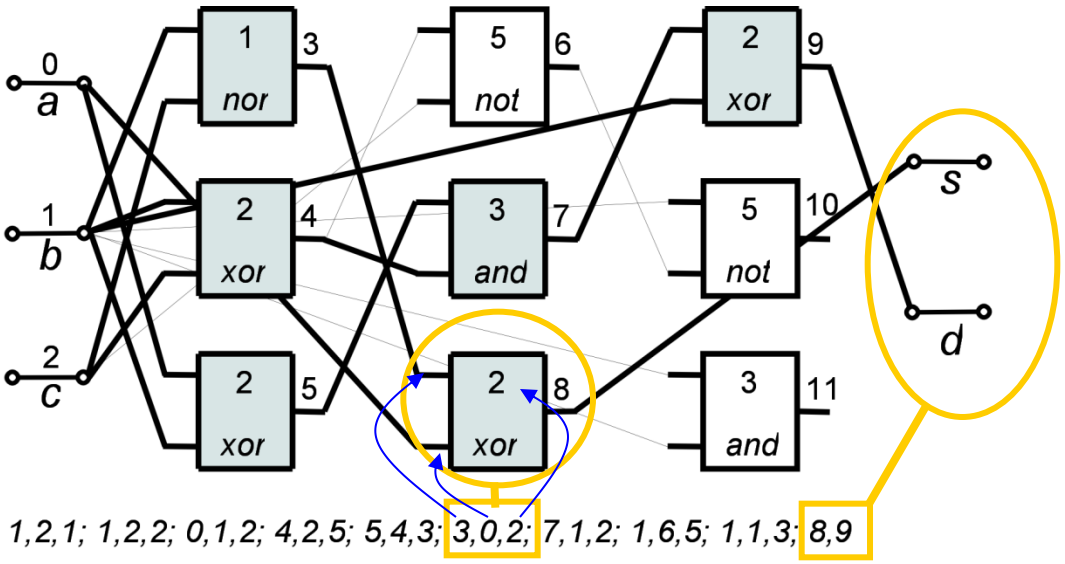
- n_a inputs of each node
- Γ function set
- L-back parameter

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

No feedback!



CGP: Representation for logic networks



- CGP parameters
  - n_r=3 (#rows)
  - n_c = 3 (#columns)
  - n_i = 3 (#inputs)
  - n_o = 2 (#outputs)
  - n_a = 2 (max. arity)
  - L = 3 (level-back parameter)
  - Γ= {NAND$^{(0)}$, NOR$^{(1)}$, XOR$^{(2)}$, AND$^{(3)}$, OR$^{(4)}$, NOT$^{(5)}$}

Genotype:

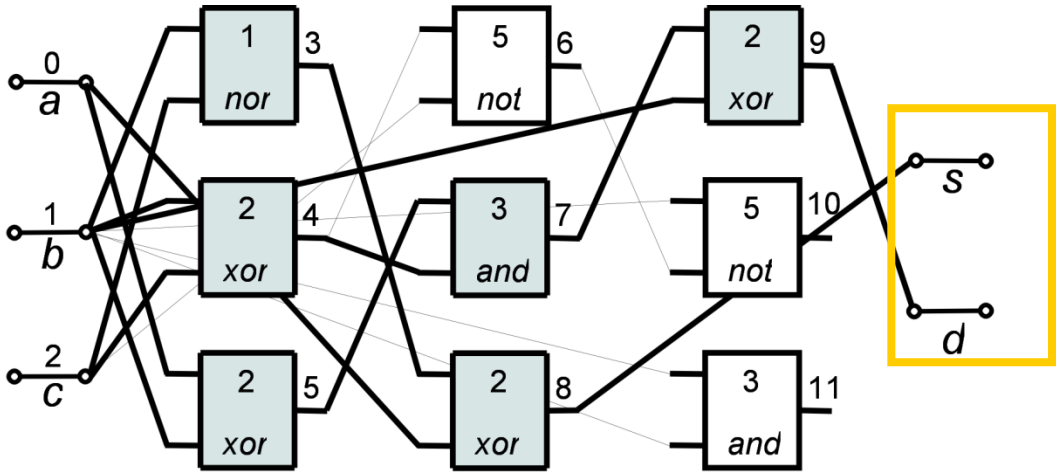
n_a+1 integers per node; n_o integers for outputs;

Constant size: $n_c n_r(n_a + 1) + n_o$ integers

Phenotype:

Variable size; unused nodes are ignored.

CGP: Fitness function for logic synthesis



1,2,1; 1,2,2; 0,1,2; 4,2,5; 5,4,3; 3,0,2; 7,1,2; 1,6,5; 1,1,3; 8,9

a	b	c	d	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

=> fitness = 16

Specification (1-bit adder), target table:

a	b	c	d	s
0	0	0	0	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=> fitness = 10

Typical fitness function (circuit functionality):

$$f = \sum_{i=1}^{K} |yi - wi|$$

K ← The number of test vectors

Desired response

Circuit response

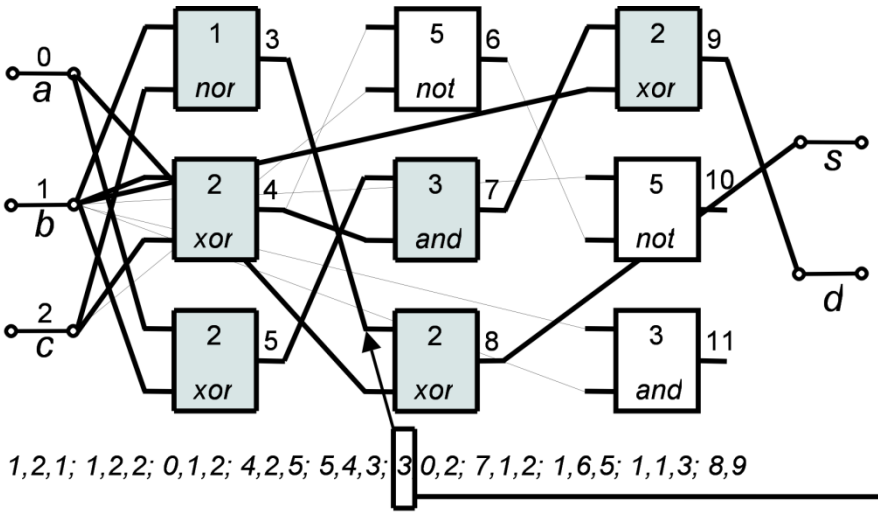
Additional objectives - minimize:

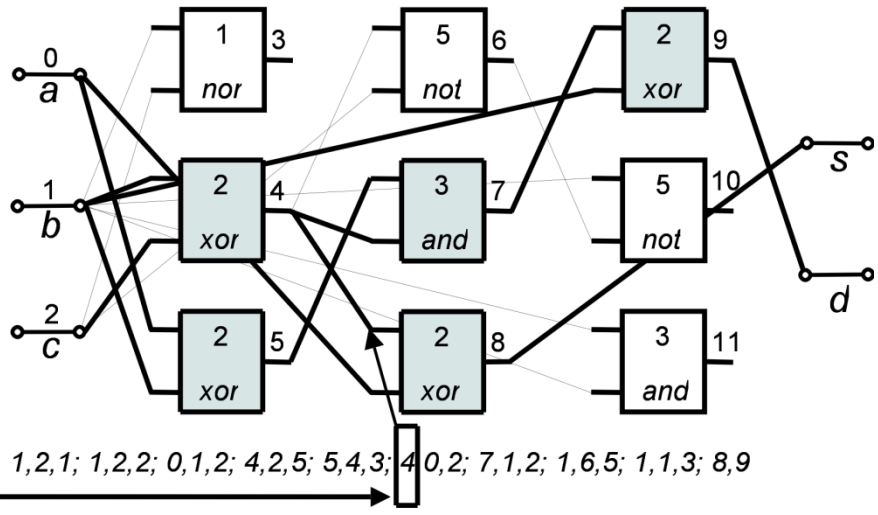
- the area (the number of gates)
- delay
- power consumption etc.

$K = 2^{inputs}$ for combinational circuits. Not scalable!!!

CGP: Mutation-based search

- Mutation: Randomly select *h* integers and replace them by randomly generated (but legal) values:





mutation

(a)

a	b	c	d	s
0	0	0	0	1
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=> fitness = 10

(b)

a	b	c	d	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

=> fitness = 16
(for full adder)

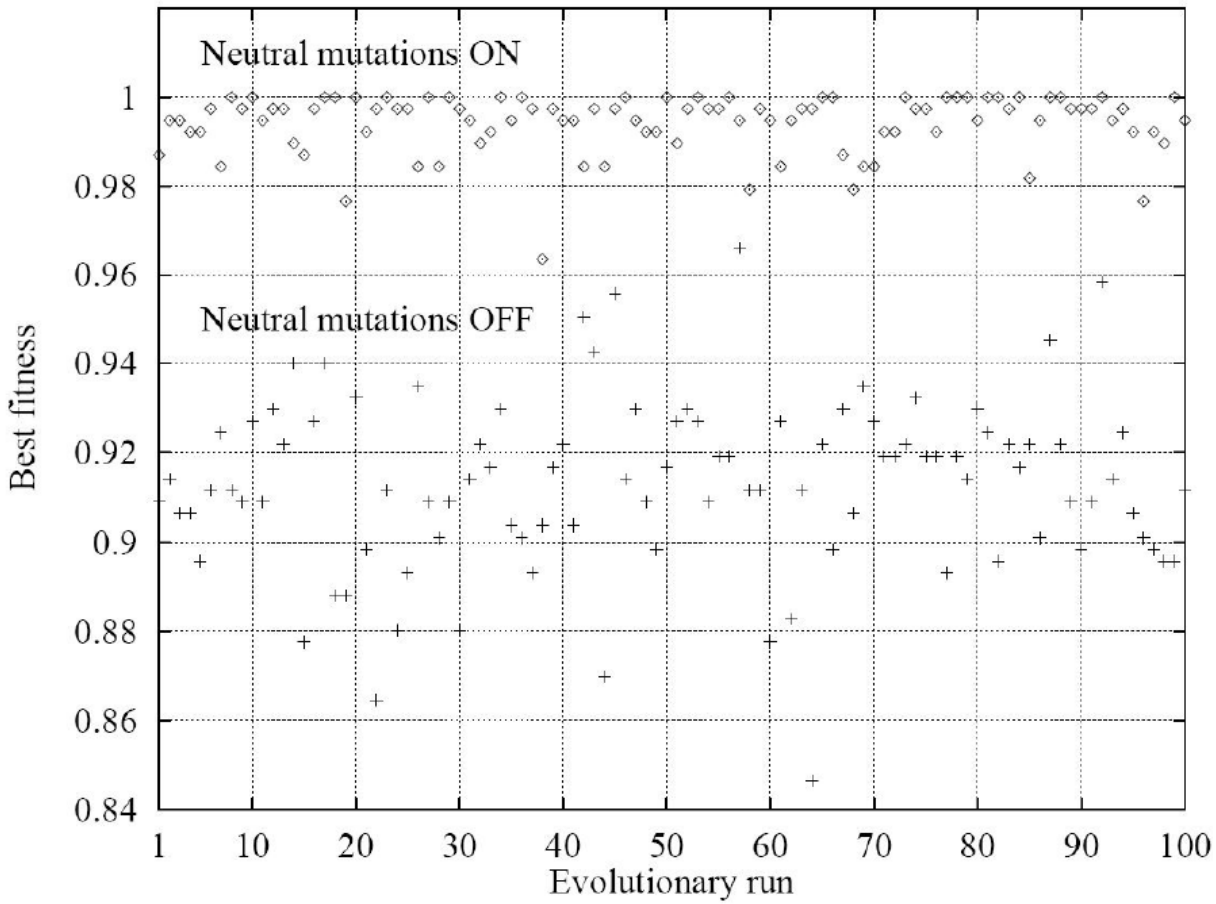
CGP: Search algorithm (1 + λ)

Algorithm 1: CGP

Input: CGP parameters, fitness function
Output: The highest scored individual p and its fitness

```
1 P ← randomly generate parent p and its λ offspring;
2 EvaluatePopulation(P);
3 while ⟨terminating condition not satisfied⟩ do
4     α ← highest-scored-individual(P);
5     if fitness(α) ≥ fitness(p) then
6         p ← α;
7     P ← create λ offspring of p using mutation;
8     EvaluatePopulation(P);
9 return p, fitness(p);
```

CGP: The role of neutrality



Evolutionary design of the 3bx3b multiplier (cf Vassilev, Miller: ICES 2000)
Normalized fitness for 100 independent runs (10M generations).
ON: A new parent replaces the former parent if its fitness is higher or equal.
OFF: A new parent replaces the former parent only if its fitness is higher.

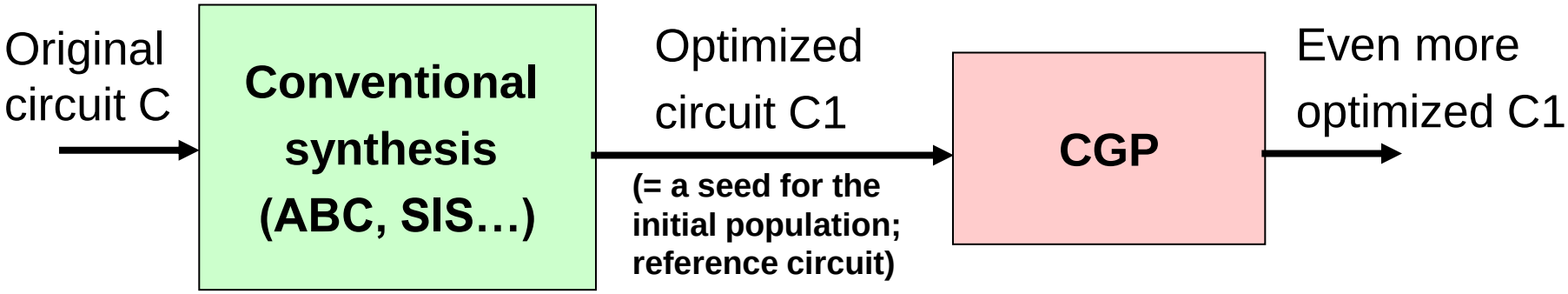
Evolutionary design at FIT BUT

- Image filters
- Logic circuits
  - complex general logic
  - bent functions
  - application protocol classifiers
  - polymorphic circuits
- Transistor-level circuits
- Benchmark circuits for diagnostics CAD tools
- Image compression
- Microprogram-controlled HW
- Protocols for WSN
- Approximate circuits and SW

- Accelerators for EA
  - FPGA, GPU, many core chips (Xeon Phi), cluster, supercomputer
- Advanced GP
  - development
  - Co-evolution
  - ALPS
  - multi-objective
  - parallel

- Other applications
  - GA in calibration of microscopic traffic models
  - Multi-objective GA pro robust prediction of traffic data

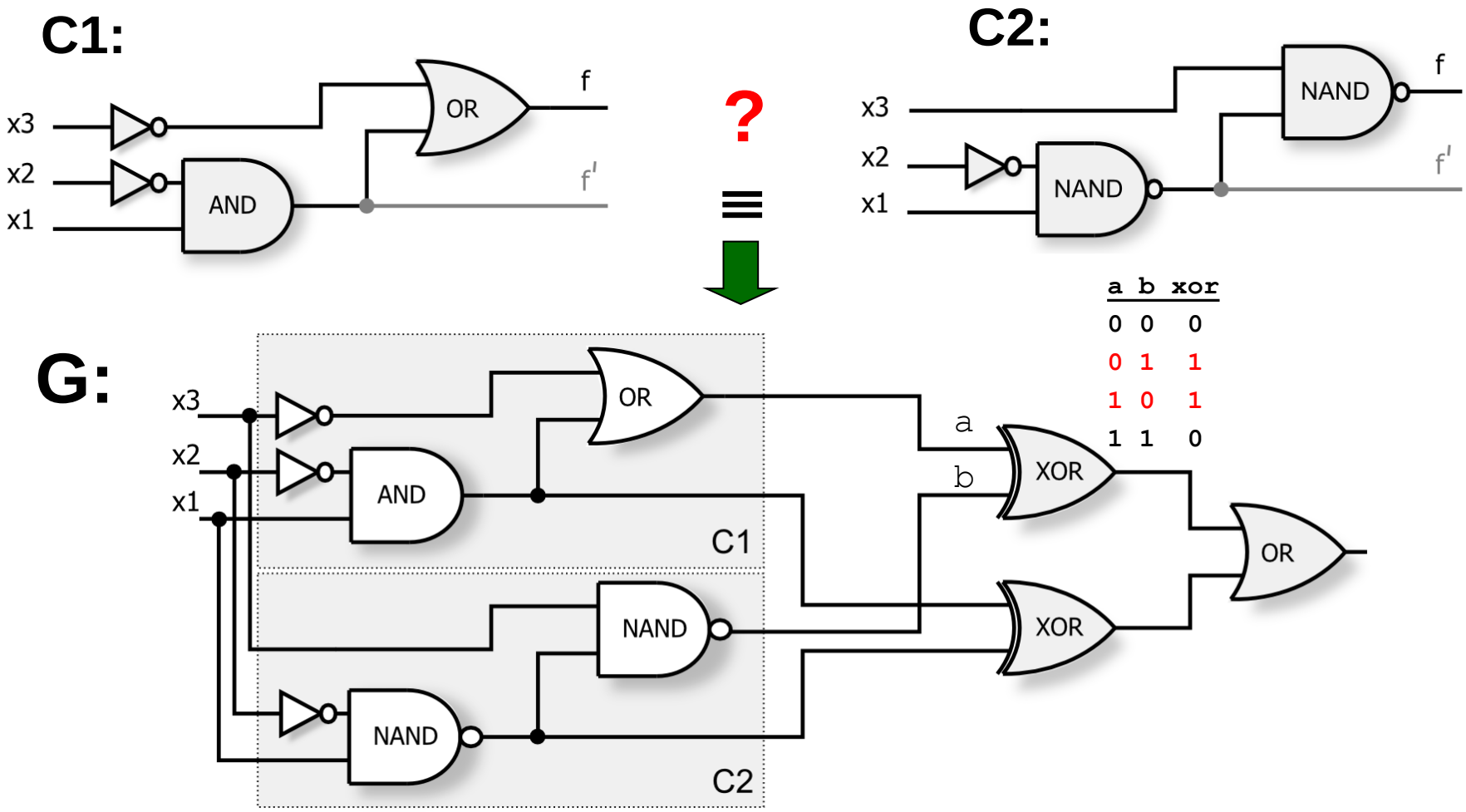
CGP for optimization of complex circuits



- SAT solver is used to decide whether candidate circuit C_i and reference circuit C1 are functionally equivalent.
  - If so, then fitness(C_i) = the number of gates in C_i;
  - Otherwise: discard C_i.

[Vašíček, Sekanina: Genetic Programming and Evolvable Machines 12(3), 2011]

SAT solver in the fitness function (1)



If C1 and C2 are not functionally equivalent then there is at least one assignment to the inputs for which the output of G is 1.

SAT solver in the fitness function (2)

- Circuit G is transformed to conjunctive normal form (CNF) using the Tseitin transform.
- The CNF representation captures the valid assignments between the gate inputs and outputs.
  - Consider a gate y = OP(a, b)
  - Hence, a CNF formula $\varphi(y, a, b) = 1$ if the predicate y = OP(a, b) holds true.

Gate	Corresponding CNF representation
$y = \mathrm{NOT}(x_1)$	$(\neg y \vee \neg x_1) \wedge (y \vee x_1)$
$y = \mathrm{AND}(x_1, x_2)$	$(y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1) \wedge (\neg y \vee x_2)$
$y = \mathrm{OR}(x_1, x_2)$	$(\neg y \vee x_1 \vee x_2) \wedge (y \vee \neg x_1) \wedge (y \vee \neg x_2)$
$y = \mathrm{XOR}(x_1, x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (\neg y \vee x_1 \vee x_2) \wedge$ $(y \vee \neg x_1 \vee x_2) \wedge (y \vee x_1 \vee \neg x_2)$
$y = \mathrm{NAND}(x_1, x_2)$	$(\neg y \vee \neg x_1 \vee \neg x_2) \wedge (y \vee x_1) \wedge (y \vee x_2)$
$y = \mathrm{NOR}(x_1, x_2)$	$(y \vee x_1 \vee x_2) \wedge (\neg y \vee \neg x_1) \wedge (\neg y \vee \neg x_2)$

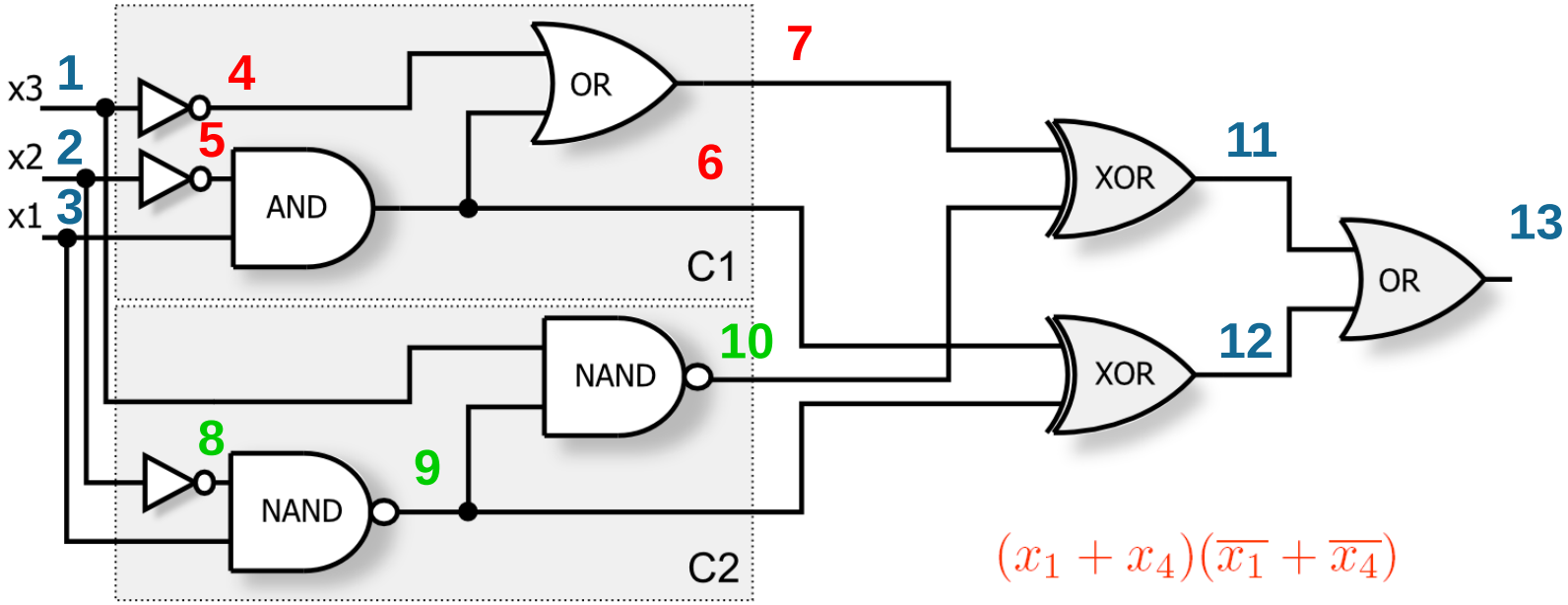
Example: `y = not (x)`

x	y	φ
0	0	0
0	1	1
1	0	1
1	1	0

CNF: (~x ∨ y) (x ∨ ~y)

- SAT solving can be combined with circuit simulation and optimized using various methods.

SAT solver in the fitness function (3)



$(x_2 + x_8)(\overline{x_2} + \overline{x_8})$

$(x_8 + x_9)(x_3 + x_9)(\overline{x_8} + \overline{x_3} + \overline{x_9})$

$(x_1 + x_{10})(x_9 + x_{10})(\overline{x_1} + \overline{x_9} + \overline{x_{10}})$

$(x_1 + x_4)(\overline{x_1} + \overline{x_4})$

$(x_2 + x_5)(\overline{x_2} + \overline{x_5})$

$(x_5 + \overline{x_6})(x_3 + \overline{x_6})(\overline{x_5} + \overline{x_3} + x_6)$

$(\overline{x_4} + x_7)(\overline{x_6} + x_7)(x_4 + x_6 + \overline{x_7})$

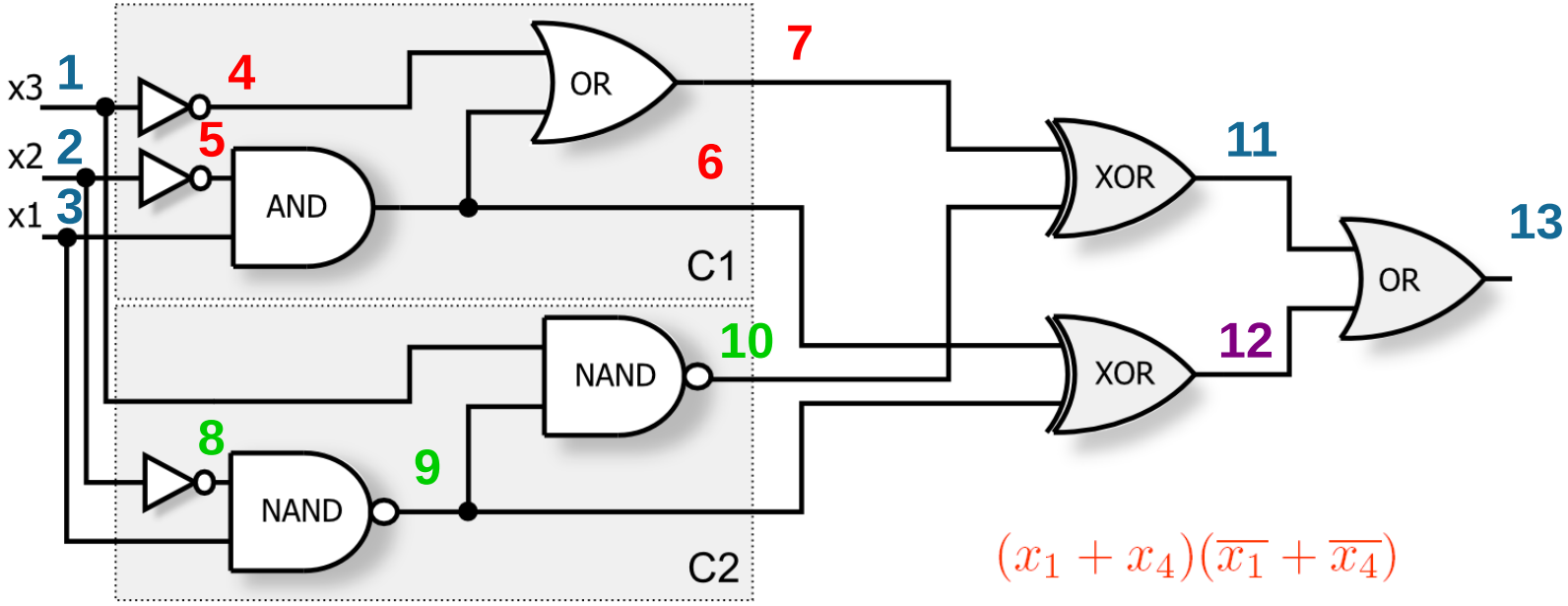
$(\overline{x_7} + x_{10} + x_{11})(x_7 + \overline{x_{10}} + x_{11})(\overline{x_7} + \overline{x_{10}} + \overline{x_{11}})(x_7 + x_{10} + \overline{x_{11}})$

$(\overline{x_6} + x_9 + x_{12})(x_6 + \overline{x_9} + x_{12})(\overline{x_6} + \overline{x_9} + \overline{x_{12}})(x_6 + x_9 + \overline{x_{12}})$

$(x_{11} + x_{12} + \overline{x_{13}})(x_{13} + \overline{x_{11}})(x_{13} + \overline{x_{12}})$

(x_{13})

SAT solver in the fitness function (3)



$(x_1 + x_4)(\overline{x_1} + \overline{x_4})$

$(x_2 + x_5)(\overline{x_2} + \overline{x_5})$

$(x_2 + x_8)(\overline{x_2} + \overline{x_8})$

$x_6)$

$\overline{x_7})$

SAT solver

variables: 13, clauses: 30, time elapsed: 0.03ms

result: SATISFIABLE / NONEQUIVALENT

model / counter example: 0011111101011

$(x_{11} + x_{12} + \overline{x_{13}})(x_{13} + \overline{x_{11}})(x_{13} + \overline{x_{12}})$

(x_{13})

Experiment 1: Minimization of the number of gates

circuit	PI	PO	SIS	ABC	C1	C2	C3	CGP	impr.
apex1	45	45	1394	1862	1439	**1272**	1368	847	33,4%
apex2	39	3	**151**	225	221	195	299	90	40,4%
apex3	54	50	1405	1737	1494	**1332**	1515	1038	22,1%
apex5	117	88	751	768	728	**609**	921	613	-0,7%
cordic	23	2	67	61	67	**49**	90	32	34,7%
cps	24	109	1128	1109	1150	975	**967**	585	39,5%
duke2	22	29	406	**356**	417	366	357	260	27,0%
e64	65	65	192	384	**183**	191	255	129	29,5%
ex4p	128	28	488	523	468	**467**	555	349	25,3%
misex2	25	18	111	121	94	**89**	108	71	20,2%
vg2	25	8	95	113	88	**83**	109	78	6,0%

CGP + SAT solver:

ES(1+1), 1 mut/chrom, seed: SIS, Gate set: {AND, OR, NOT, NAND, NOR, XOR}, 100 runs

Average area improvement: 25%

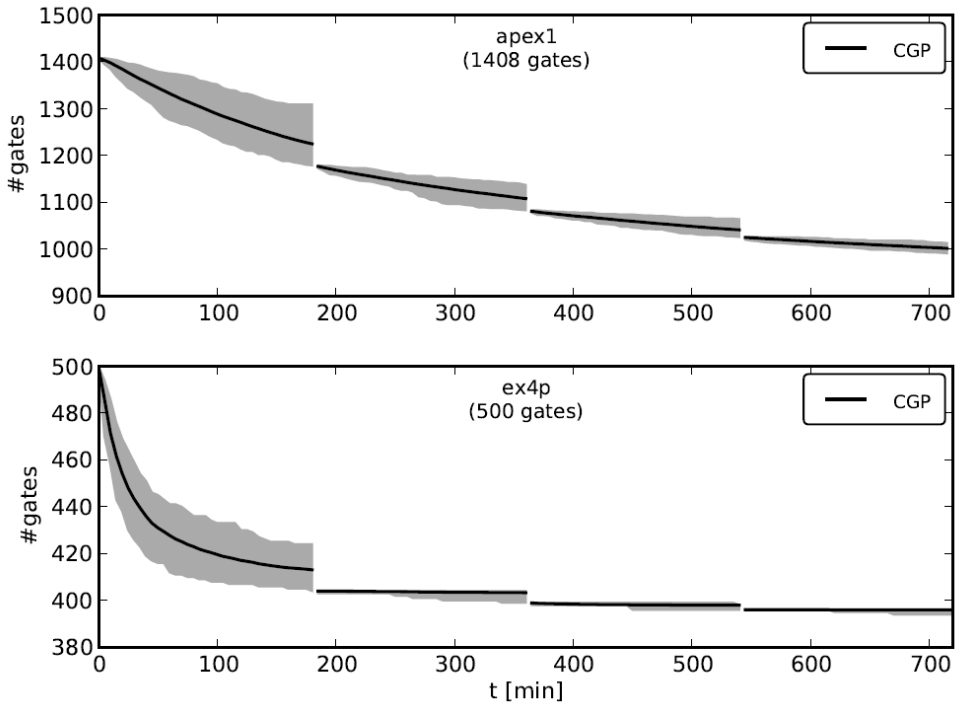
Delay increased: +3 logic levels (but not optimized!)

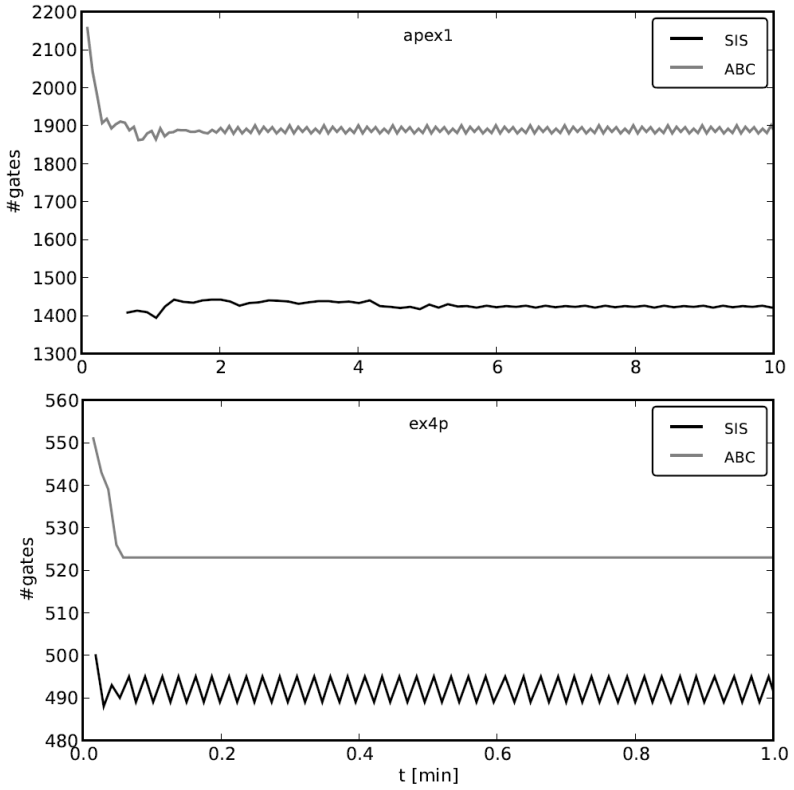
ABC, SIS – conventional open academic synthesis tools

C1, C2, C3 – commercial synthesis tools

[Vašíček, Sekanina: DATE 2011]

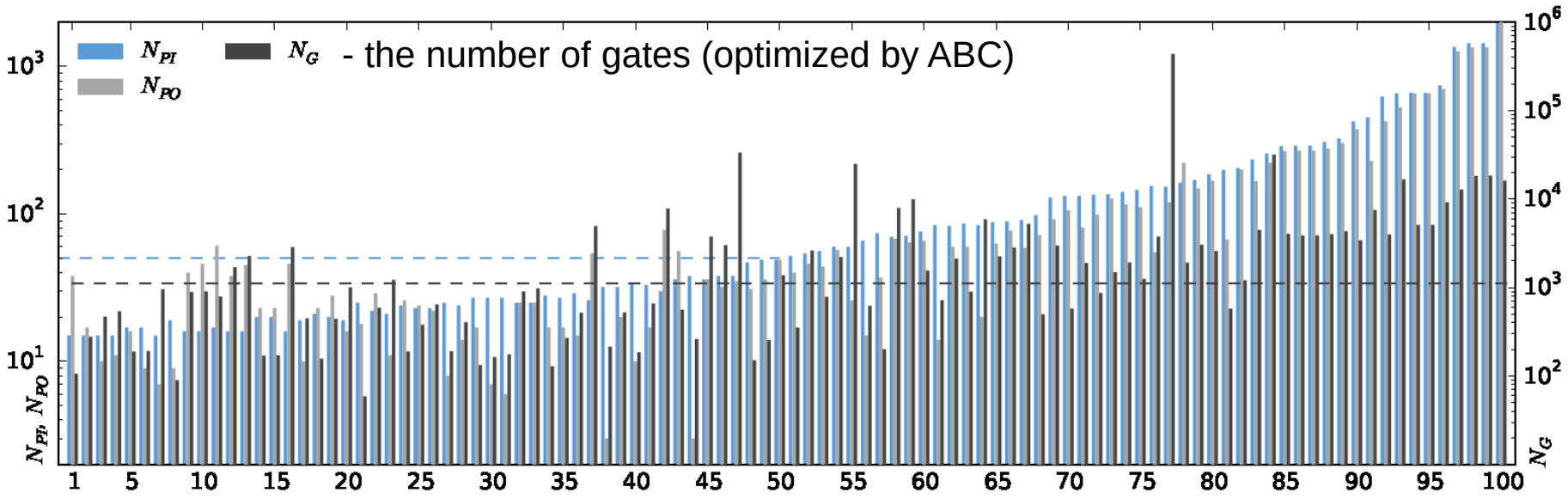
Experiment 1: Convergence curves





- Summary
  - More time ⇒ better results
  - Current circuit synthesis and optimization tools provide far from optimum circuits!

Exp. 2: More complex circuits



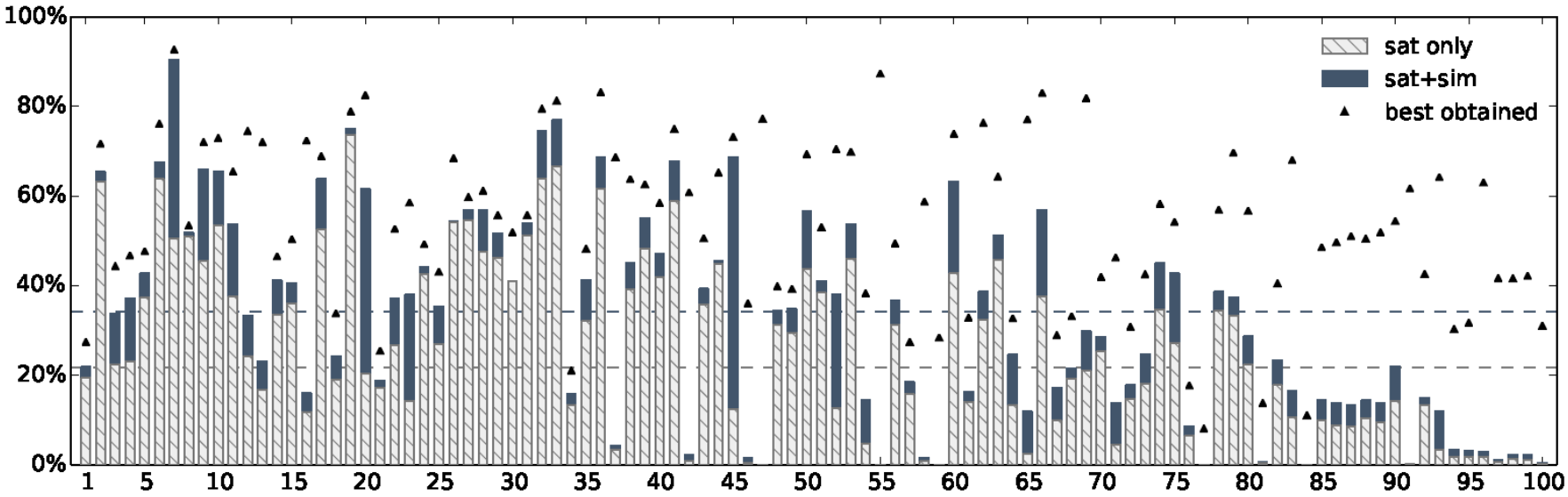
100 combinational circuits (≥15 inputs) - IWLS2005, MCNC, QUIP benchmarks

Heavily optimized by ABC

1: alcom (N_G = 106 gates; N_{PI} = 15 inputs; N_{PO} = 38 outputs)

100: ac97ctrl (N_G = 16,158; N_{PI} = 2,176; N_{PO} = 2,136)

Exp. 2: More complex circuits



CGP + SAT solver + circuit simulation

Y-axis: Gate reduction w.r.t. ABC after 15 minutes, 34% on average

▲ Gate reduction w.r.t. ABC after 24 hours

[Vašíček Z.: EuroGP 2015]

BDD in the fitness function [Vašíček, Sekanina: ICES 2014]

Objective

- To evolve complex circuits from scratch (to avoid biasing by conventional solutions)
- To minimize the number of gates

The method: CGP

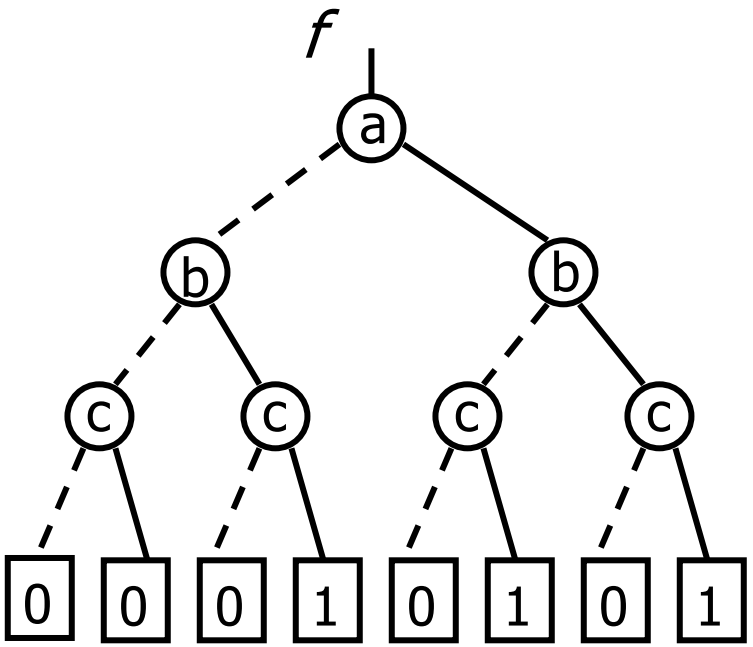
- The specification is given as a Binary Decision Diagram (BDD).
- Every candidate circuit is converted from CGP representation to BDD and its functionality is compared against the specification (BDD).
- The operations over BDDs are implemented using Buddy.

Standard CGP will be compared with CGP-BDD.

$f = ac + bc$

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Truth table

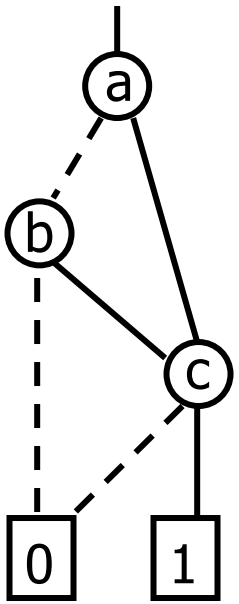


Decision tree

1 edge

0 edge

$f = (a+b)c$



Reduced Ordered BDD (ROBDD)

"Apply" operation

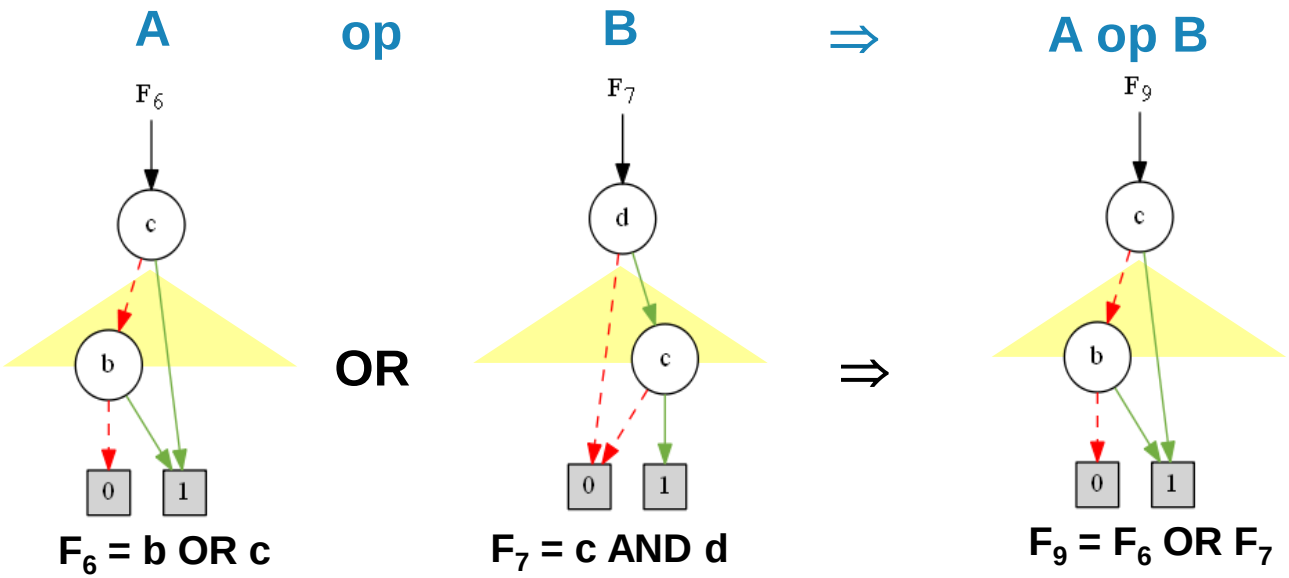
- Is a basic technique for building ROBDD from a Boolean formula

Arguments:

- A, B: Boolean functions, represented as ROBDDs
- op: Boolean operation (e.g., AND, OR, XOR)

Result:

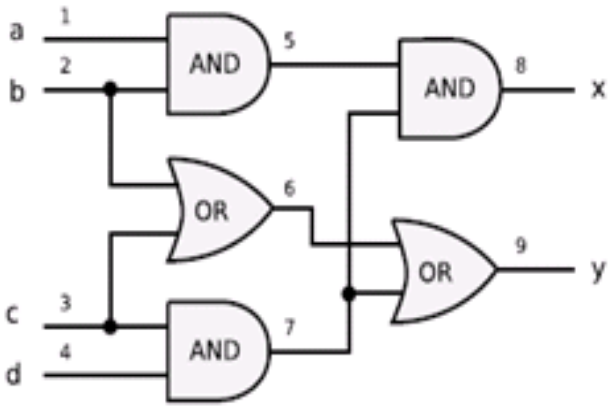
- ROBDD representing composite function A op B

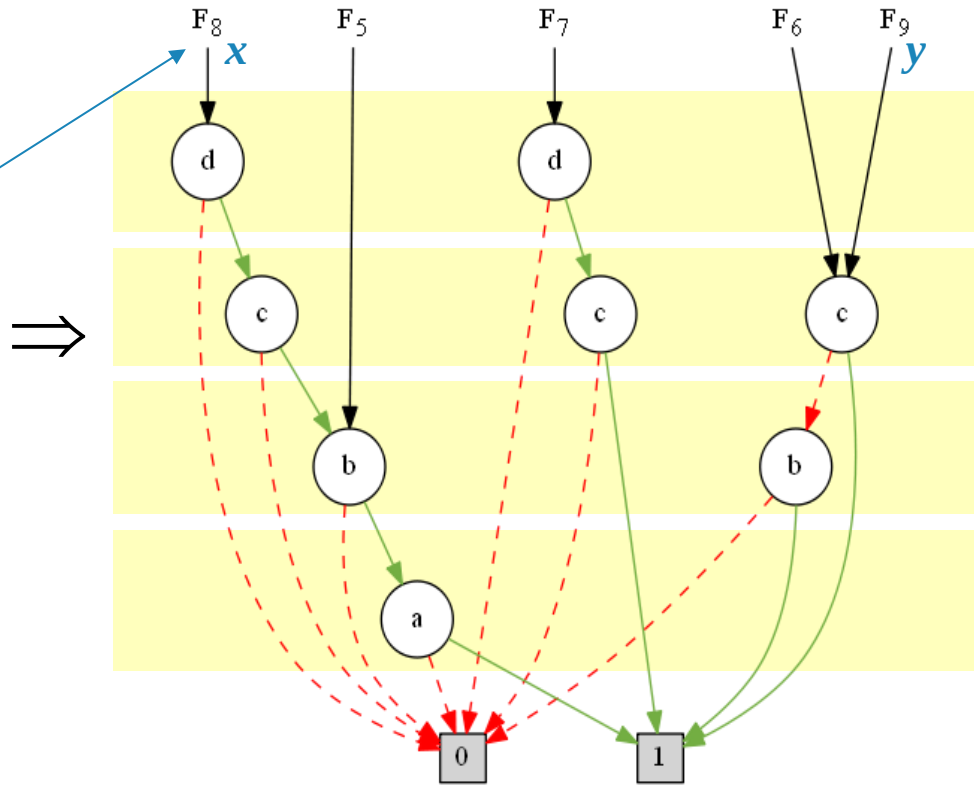


From CGP representation to ROBDD

1. Inform a BDD library that n_i variables are required. Specify ordering.
2. Visit each CGP node encoded as (i_1, i_2, f) and perform **Apply** operation using Boolean operation f on top of the variables i_1 and i_2

Example:

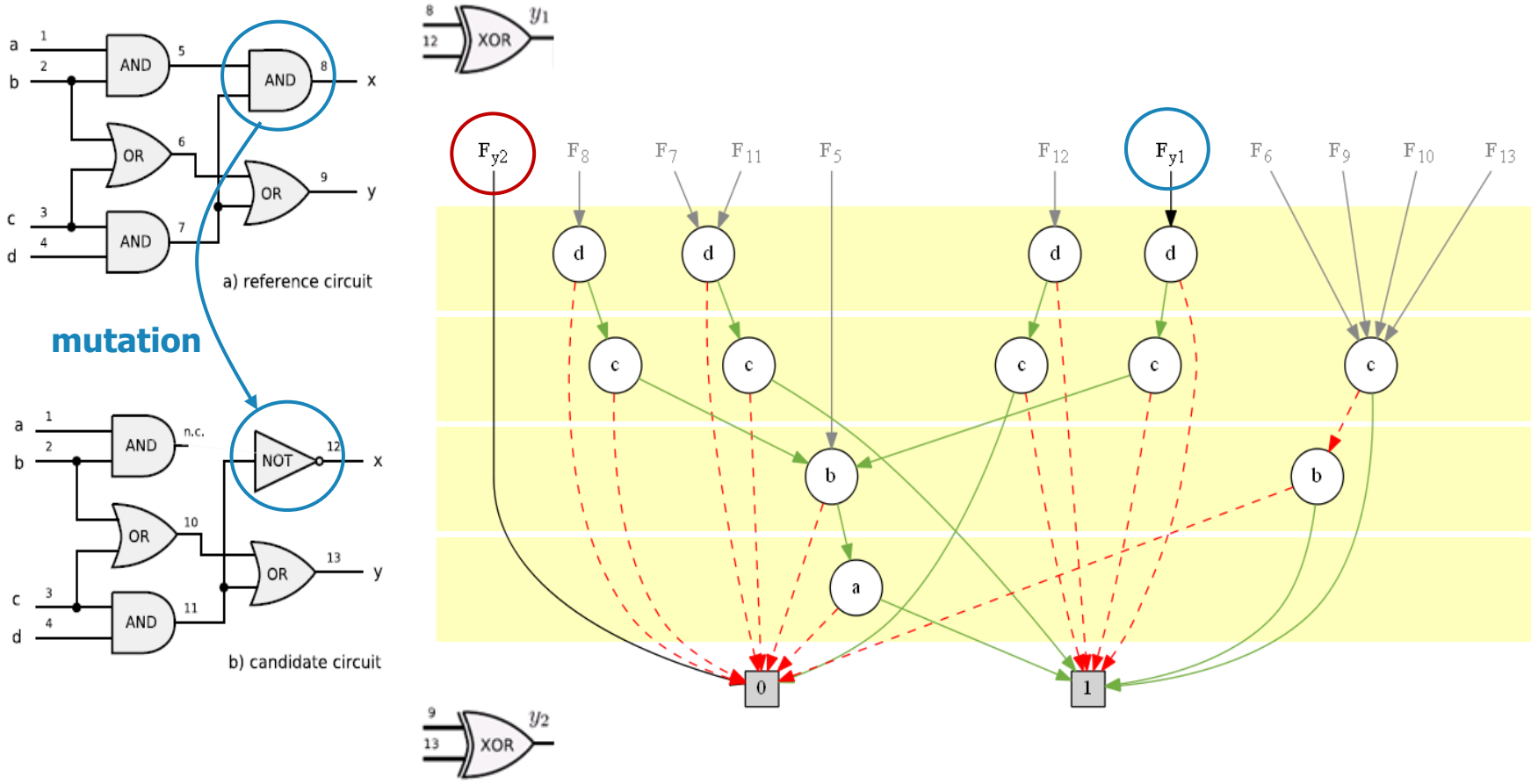




Ordering: d > c > b > a

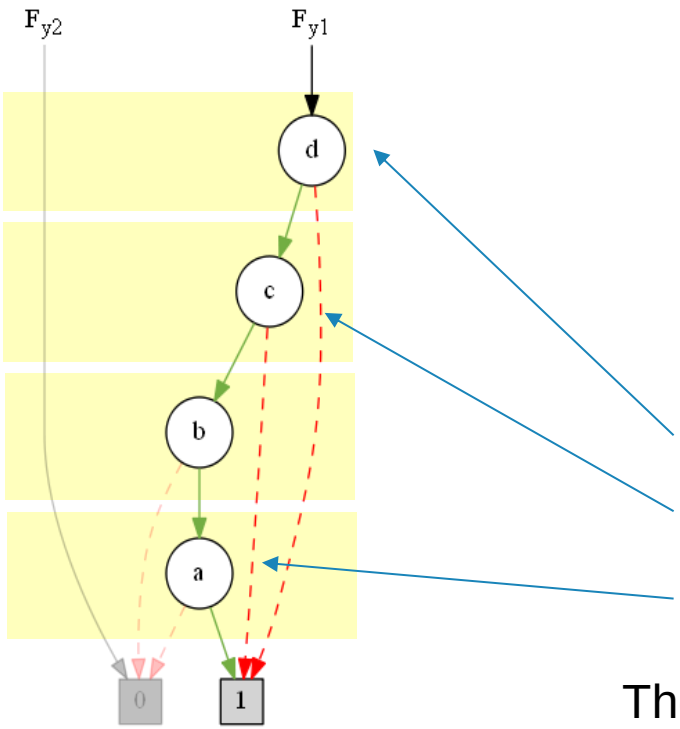
How to calculate the fitness value – 1/2

1. For a candidate circuit, apply the same approach as for a reference
2. Combine all the corresponding outputs with XOR operation



How to calculate the fitness value – 2/2

3. **SatCount** operation is applied on every output y_i (linear time complexity)
4. The sum of the obtained values represents the Hamming distance between a candidate solution and a chosen reference implementation.



1. SatCount(F_{y1}) = 13
2. SatCount(F_{y2}) = 0
3. Fitness value = 13 + 0 = 13

The assignments evaluating F_{y1} to 1:

a	b	c	d	# combinations
-	-	-	0	8
-	-	0	1	4
1	1	1	1	1

The zero fitness value means that a fully working solution was discovered.

Benchmark Circuits

Circuit	PI	PO	Gates	Levels	CE	|BDD|
x2	10	7	41	7	6.6×10^2	38
cm151a	12	2	28	8	1.8×10^3	32
9sym	9	1	228	14	1.8×10^3	33
ex5	8	63	505	10	2.0×10^3	293
cm162a	14	5	41	8	1.0×10^4	43
cu	14	11	47	8	1.2×10^4	52
apex4	9	19	2868	17	2.3×10^4	1011
b12	15	9	60	7	3.1×10^4	72.0
cm163a	16	5	42	7	4.3×10^4	31
t481	16	1	66	11	6.8×10^4	32
tcon	17	16	33	3	6.8×10^4	24
alu4	14	8	1059	22	2.7×10^5	769
table5	17	15	1500	24	3.1×10^6	753
cordic	23	2	63	11	8.3×10^6	83
frg1	28	3	103	12	4.3×10^8	91
C499	41	32	185	12	6.4×10^{12}	31699

- Pre-optimized by ABC
- Computational (simulation) Effort: CE = gates * 2^{inp} (64-bit CPU)
- Sorted according to CE

CGP parameters

- n_c = # of gates from ABC
- n_r = 1
- L-back = max.
- ES (1+4)
- 5 mutations/chromosome
- Gate set: AND, OR, NOT, NAND, NOR, XOR, NXOR, BUF
- 3-hours/run

CGP: a well optimized implementation of the standard CGP

CGP-BDD: CGP utilizing the Buddy BDD library

Evolutionary design from scratch

Circuit	# generations CGP	# generations CGP-BDD	speedup	success rate CGP	success rate CGP-BDD	PI	Gates	|BDD|
x2	1.6×10^8	1.6×10^8	1.0	4.0%	4.0%	10	41	38
cm151a	5.8×10^7	1.2×10^8	2.1	97.0%	**98.0%**	12	28	32
9sym	1.9×10^8	8.0×10^7	0.4	100.0%	100.0%	9	228	33
ex5	6.1×10^7	4.1×10^7	0.7	**13.0%**	7.0%	8	505	293
cm162a	1.2×10^7	1.2×10^8	10.4	44.0%	**100.0%**	14	41	43
cu	1.2×10^7	1.6×10^8	13.2	0.0%	**7.0%**	14	47	52
apex4	1.5×10^7	9.9×10^6	0.7	0.0%	0.0% →	9	2868	1011
b12	4.6×10^6	1.3×10^8	28.1	0.0%	**19.0%**	15	60	72.0
cm163a	3.4×10^6	1.6×10^8	47.8	0.0%	**20.0%**	16	42	31
t481	3.7×10^6	1.7×10^8	45.9	92.0%	**100.0%**	16	66	32
tcon	1.6×10^6	1.8×10^8	113.0	15.0%	**100.0%**	17	33	24
alu4	2.0×10^6	1.8×10^7	8.5	0.0%	**58.0%**	14	1059	769
table5	3.0×10^5	2.7×10^7	90.2	0.0%	0.0% →	17	1500	753
cordic	n.a.	2.3×10^8	n.a.	0.0%	0.0% →	23	63	83
frg1	n.a.	1.1×10^8	n.a.	0.0%	**54.0%**	28	103	91
C499	n.a.	6.6×10^6	n.a.	0.0%	0.0% →	41	185	31699

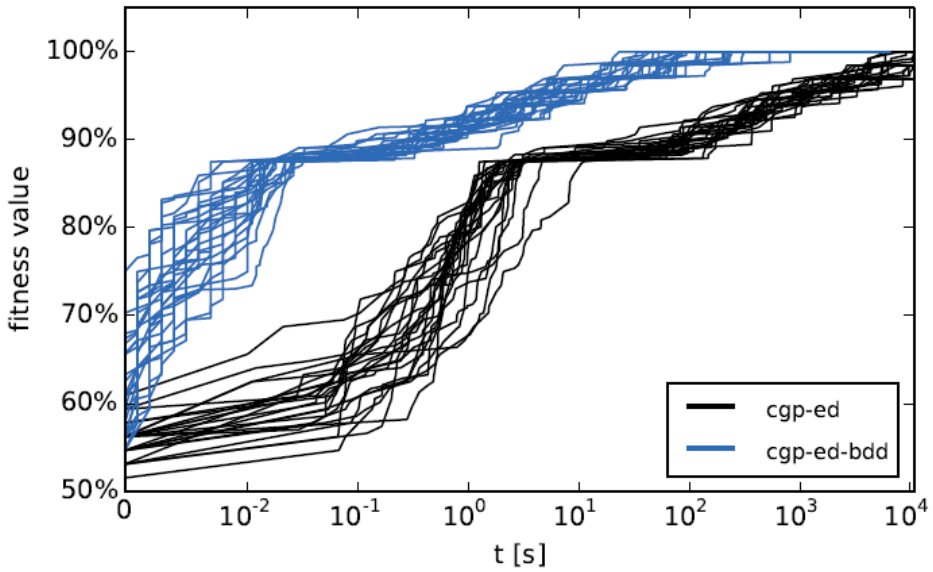
- CGP-BDD gives the speedup 1 – 2 orders of magnitude
- The success rate calculated from 100 independent runs.
- Is the BDD size the main indicator of difficulty?

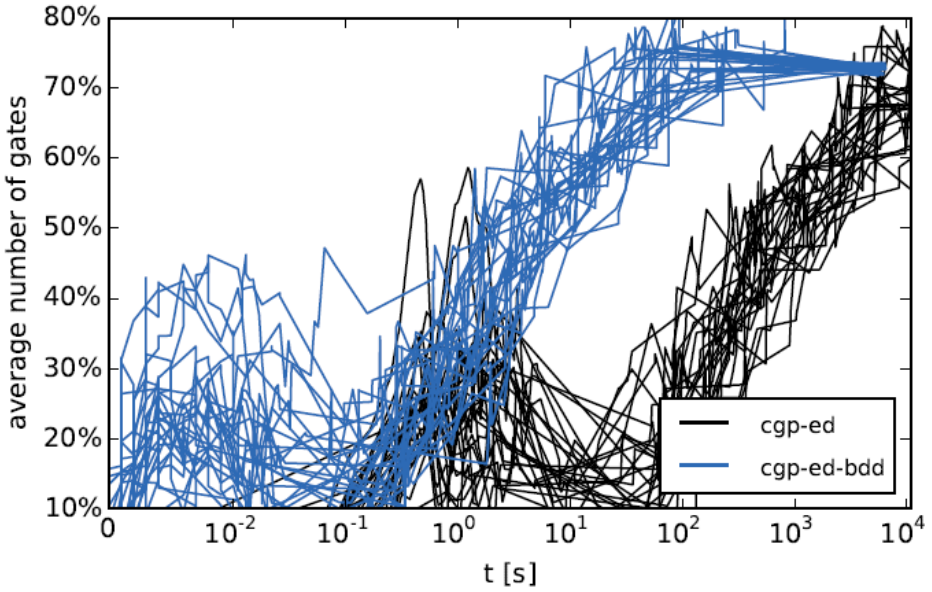
Parameters of evolved circuits

	CGP			CGP-BDD		
Circuit	f_{bst}	g_{bst}	Improv.	f_{bst}	g_{bst}	Improv.
x2	100.0%	27.0	34.1%	100.0%	26.0	**36.6%**
cm151a	100.0%	23.0	17.9%	100.0%	23.0	17.9%
9sym	100.0%	23.0	89.9%	100.0%	23.0	89.9%
ex5	100.0%	152.0	**69.9%**	100.0%	155.0	69.3%
cm162a	100.0%	26.0	36.6%	100.0%	26.0	36.6%
cu	99.8%	24.9	n.a.	**100.0%**	34.0	**27.7%**
apex4	**89.7%**	648.1	n.a.	89.2%	594.0	n.a.
b12	99.3%	35.7	n.a.	**100.0%**	45.0	**25.0%**
cm163a	99.4%	23.6	n.a.	**100.0%**	26.0	**38.1%**
t481	100.0%	22.0	66.7%	100.0%	21.0	**68.2%**
tcon	100.0%	25.0	24.2%	100.0%	25.0	24.2%
alu4	99.3%	178.2	n.a.	**100.0%**	70.0	**93.4%**
table5	98.1%	134.3	n.a.	**99.4%**	157.5	n.a.
cordic	n.a.	n.a.	n.a.	**99.4%**	12.2	n.a.
frg1	n.a.	n.a.	n.a.	**100.0%**	44.0	**57.3%**
C499	n.a.	n.a.	n.a.	**99.6%**	36.2	n.a.

- If f_{bst} < 100% (functionality) then g_{bst} is the average number of gates in the last 100 generations across all the runs.
- Improv. is the improvement against ABC.

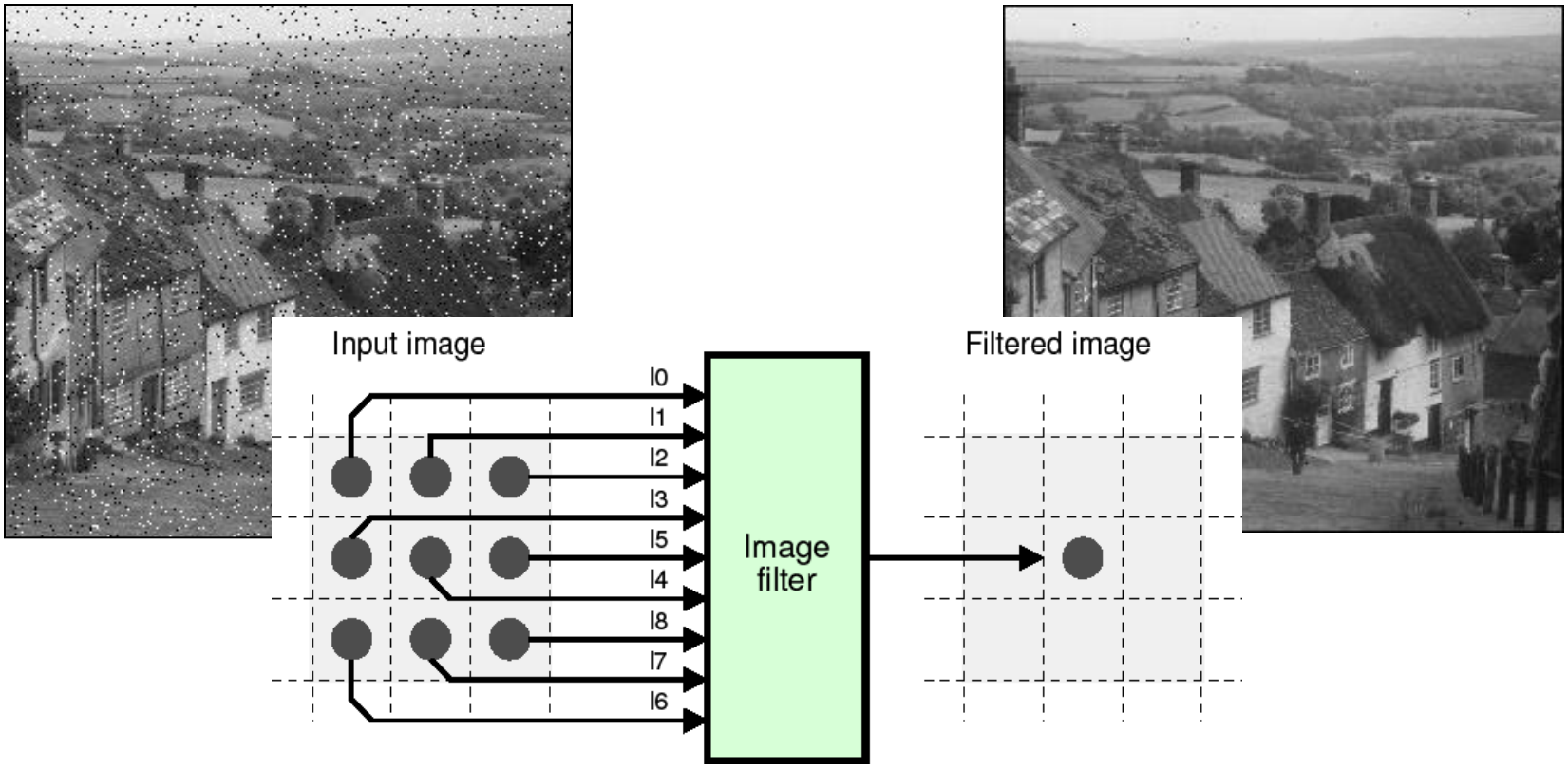
Convergence curves – tcon (25 runs)





- The average number of gates in all phenotypes w.r.t. ABC.

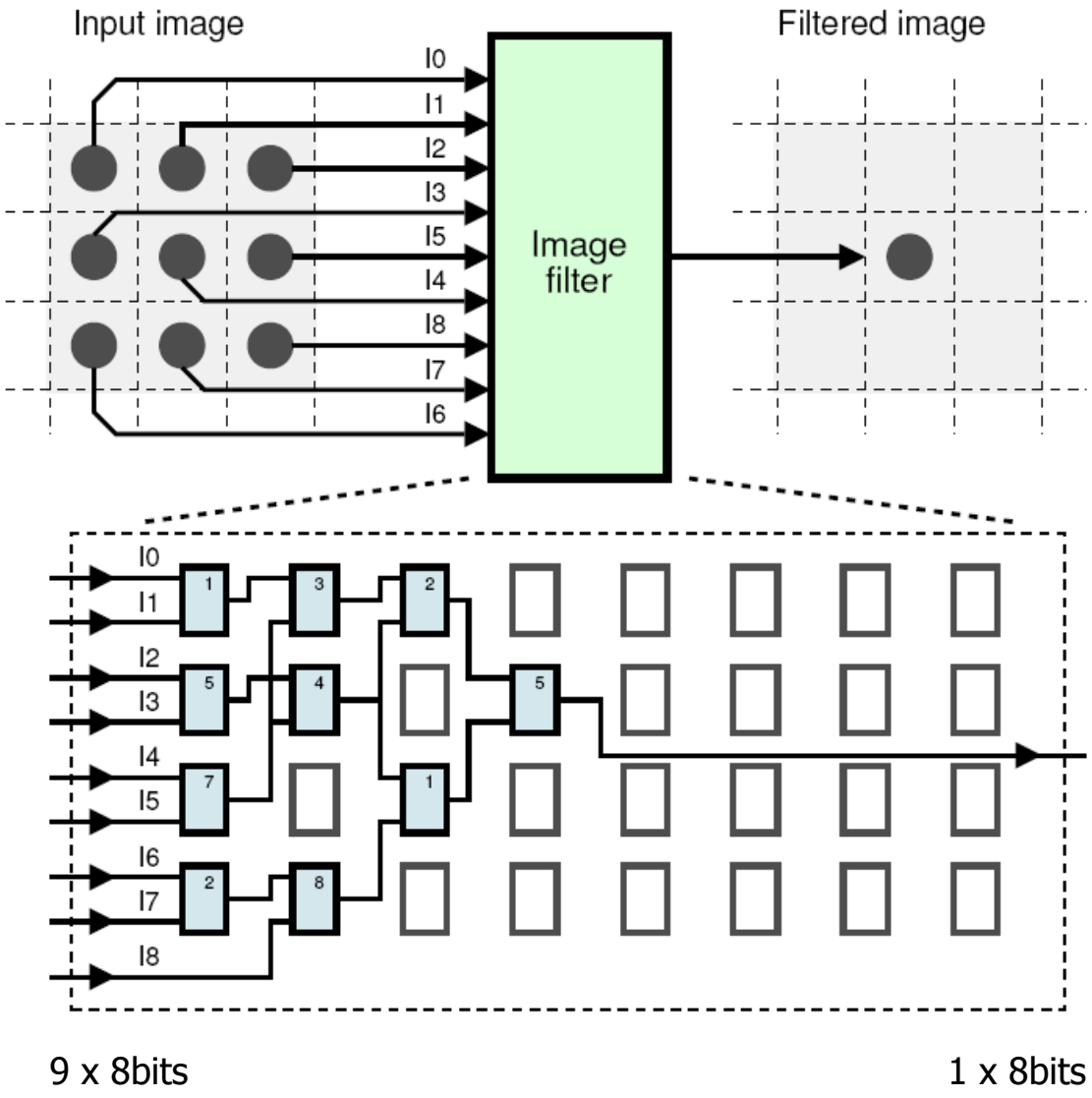
Evolutionary design of image filters



Can EA design an image filter which exhibits better filtering properties and lower implementation cost w.r.t. conventional solutions?

Target domain: filters suppressing shot noise, Gaussian noise, burst noise, edge detectors, ...

Image filter in CGP



- Array of programmable. elements (PE).
- No feedbacks.
- All I/O and connections on 8 bits.

- PE over 8 bits:
  - Minimum
  - Maximum
  - Average
  - Constants
  - logic operators
  - shift

Sekanina L.: EvoAISP 2002

Fitness function

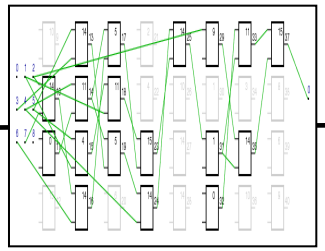
$$MAE = \frac{1}{RC}\sum_{r=0}^{R-1}\sum_{c=0}^{C-1}\left|I(r,c)-K(r,c)\right|$$

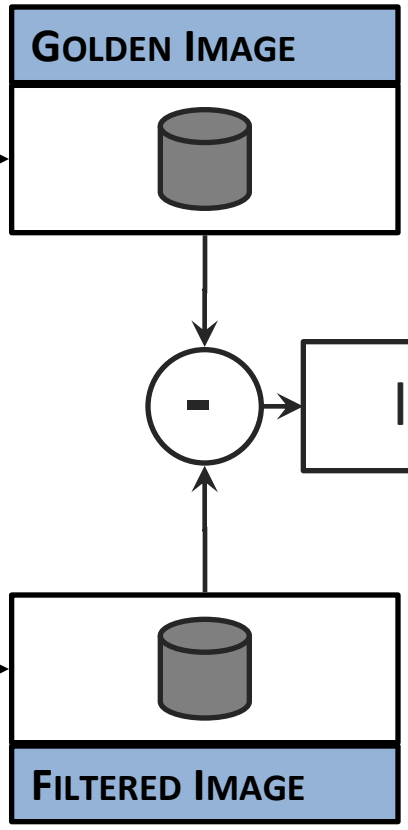
I: golden image

K: filtered image

R: rows

C: columns





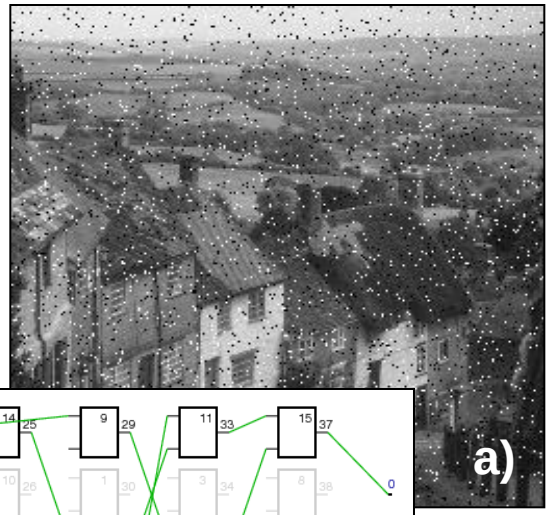
Example of evolved filter behavior

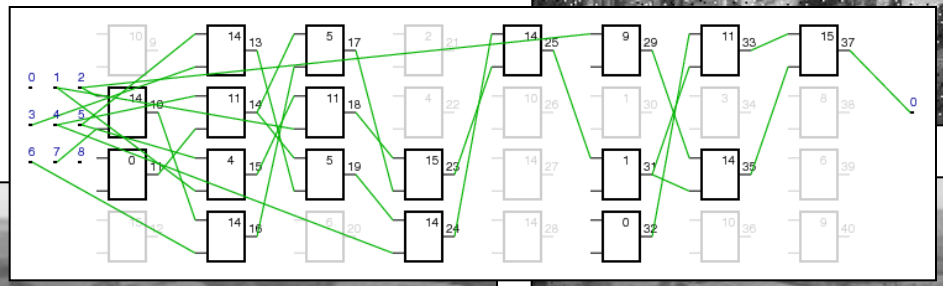
a) Image corrupted by 5% salt-and-pepper noise
PSNR: 18.43 dB (peak signal to noise ratio)

b) Original image

c) Median filter (kernel 3x3)
PSNR: 27.92 dB
268 FPGA slices; 305 MHz

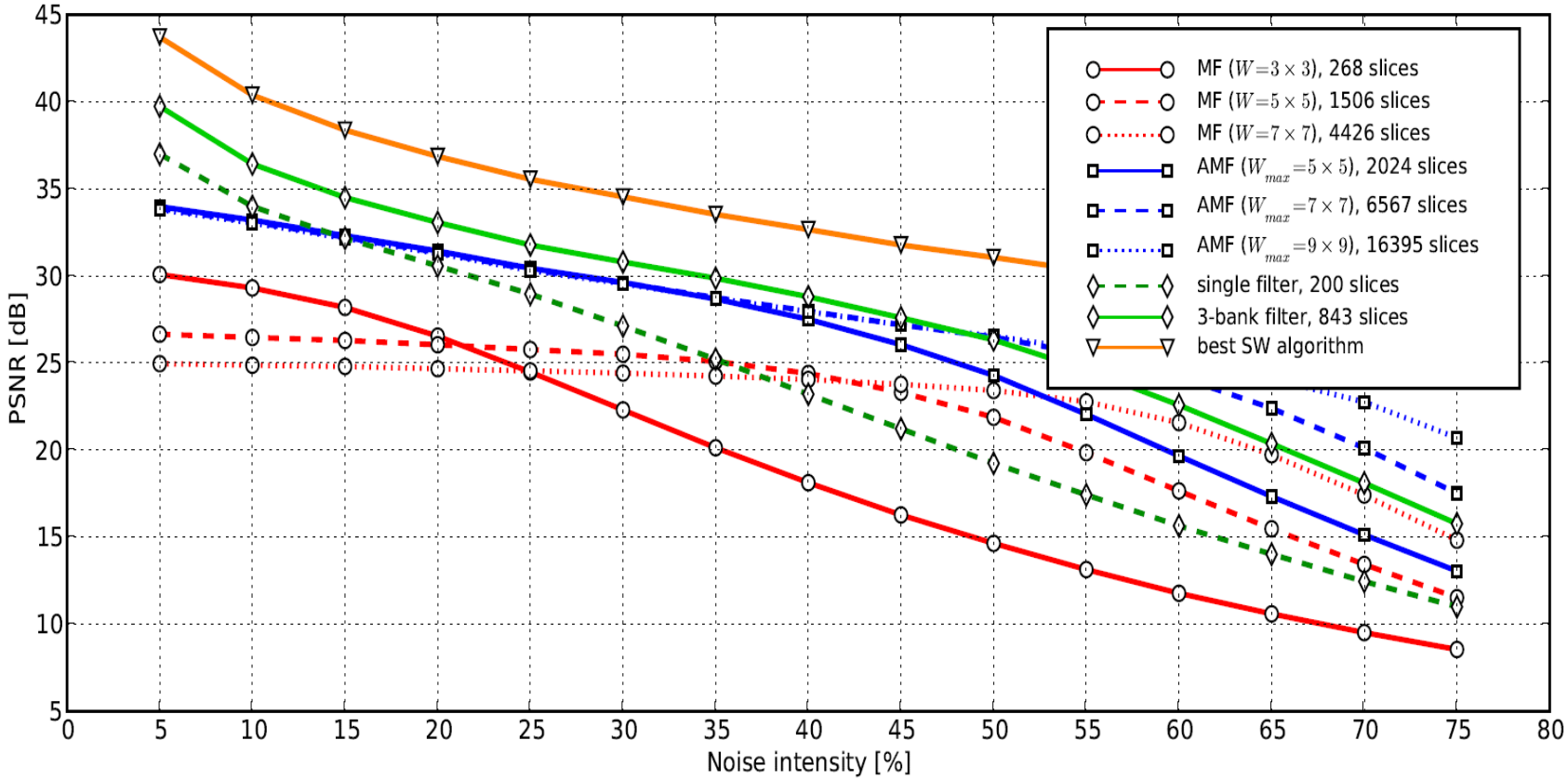
d) Evolved filter (kernel 3x3)
PSNR: 37.50 dB
200 FPGA slices; 308 MHz





Comparison of various filters

Mean PSNR for 25 test images



MF – Median Filter; AMF – Adaptive Median Filter

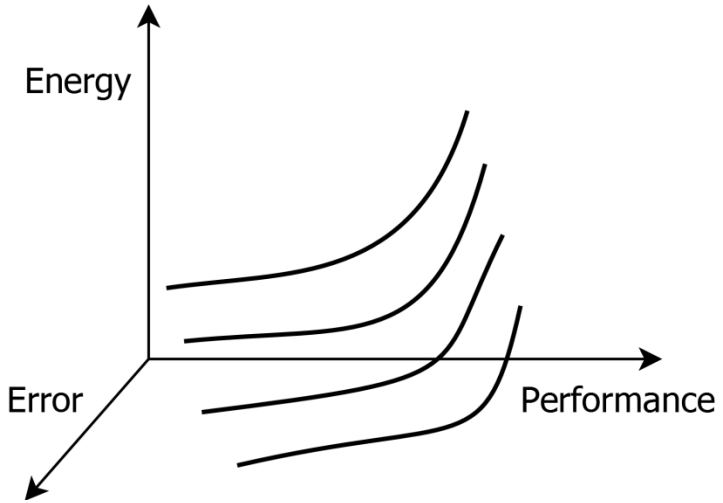
The single filter and 3-bank are evolved filters (Czech patent #304181).

Best SW - Y. Dong, S. Xu: A new directional weighted median filter for removal of random-valued impulse noise. Signal Processing Letters. vol. 14, no. 3, p. 193–196, 2007

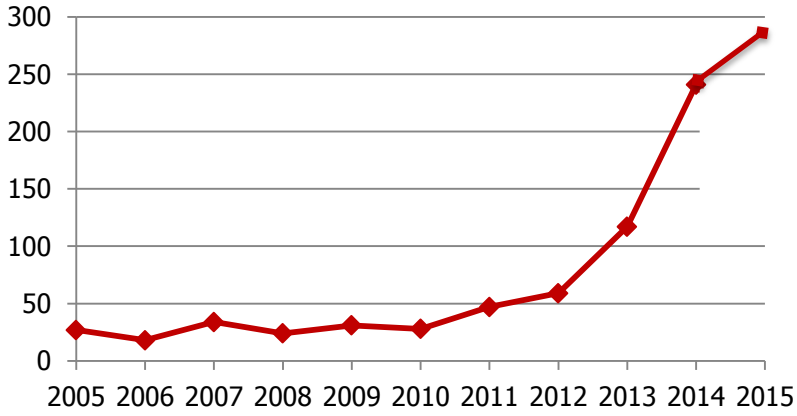
Approximate computing

- High performance & low power computing needed:
  - “Big data” processing
  - Mobile electronics with low power budget
- Many applications are error-resilient
  - the error can be traded for energy savings or performance.
- Approximations are currently introduced for SW as well as HW components.

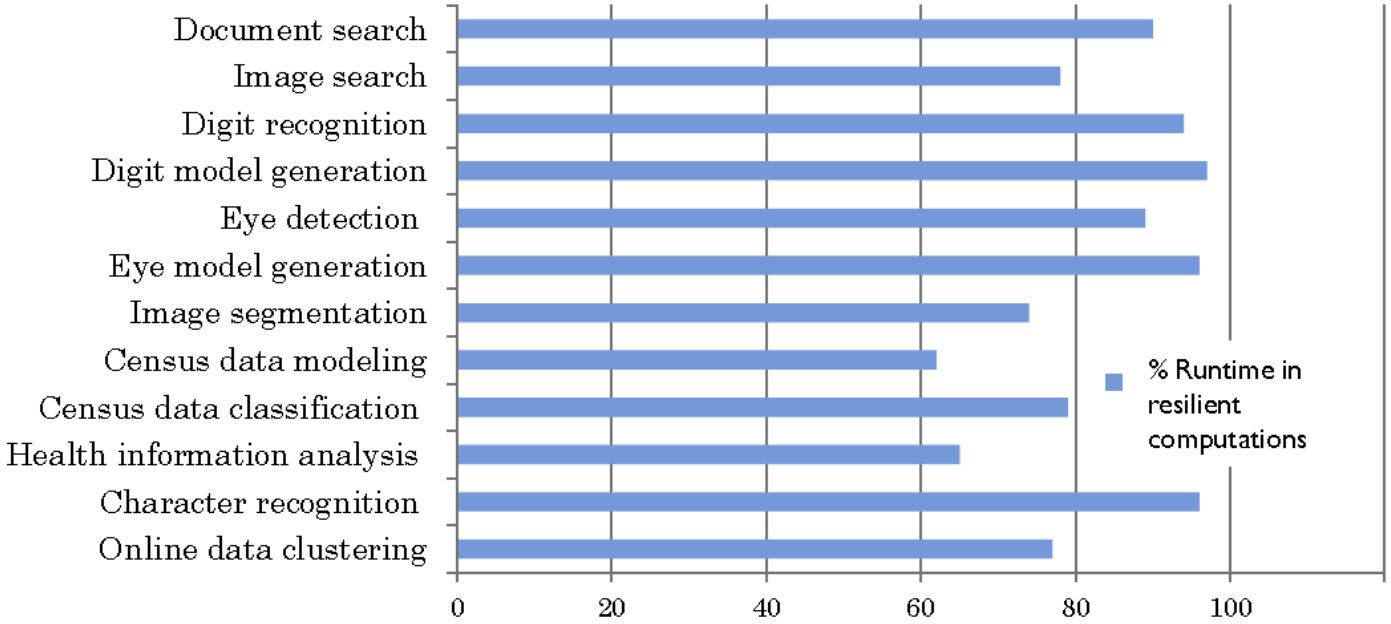
Error as a design metric!



Search for "approximate computing" in articles by Google Scholar (Nov 2015)



Many applications are inherently error resilient



Applications have a mix of resilient and sensitive computations

83% of runtime spent in computations that can be approximated

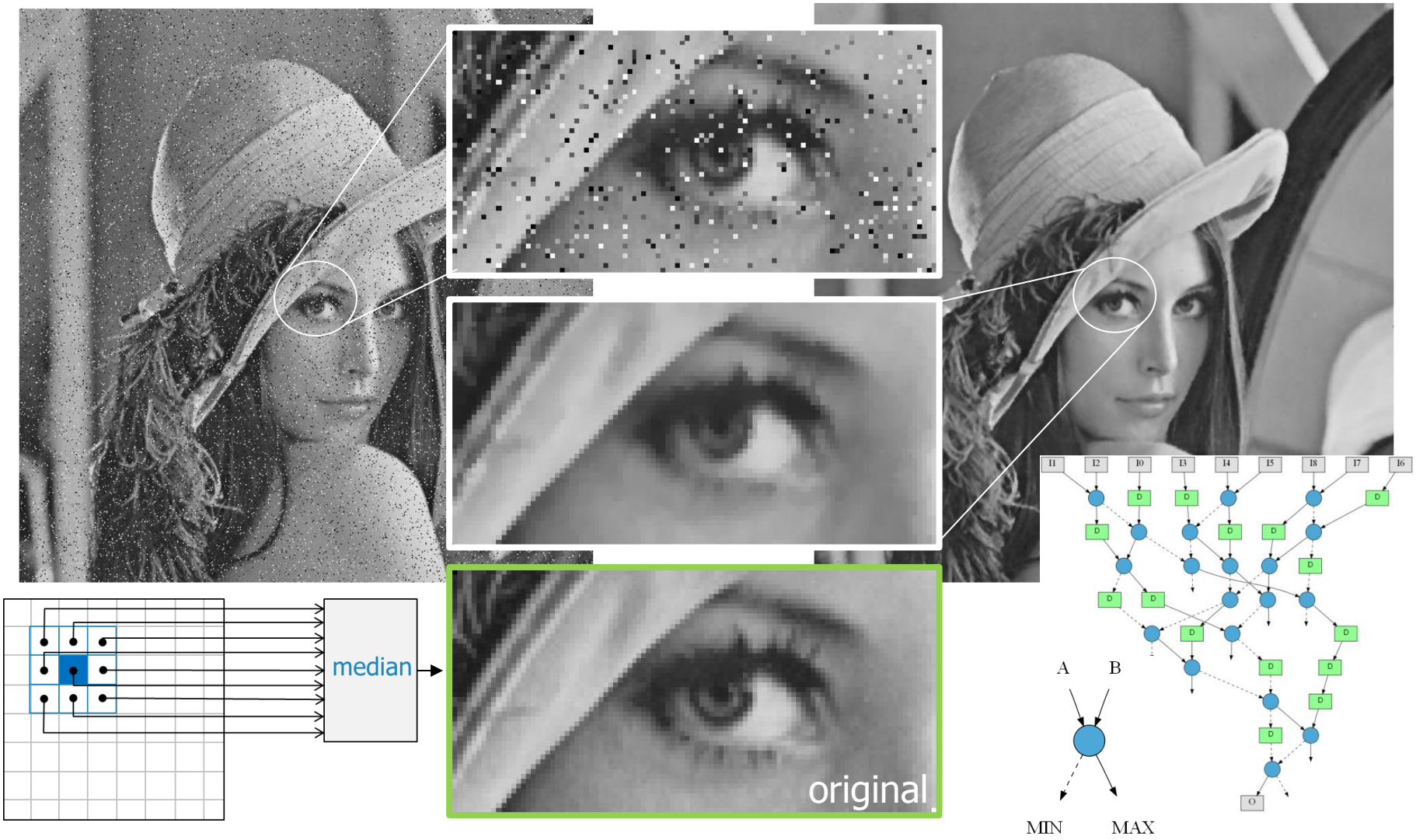
V. K. Chippa, S.T. Chakradhar, K. Roy and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," DAC 2013.

Courtesy of K. Roy

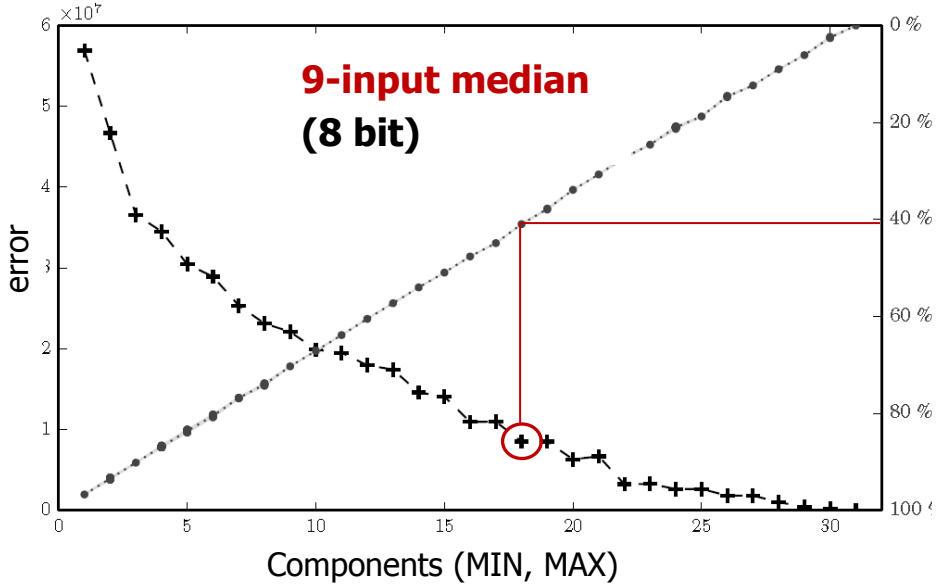
The median function

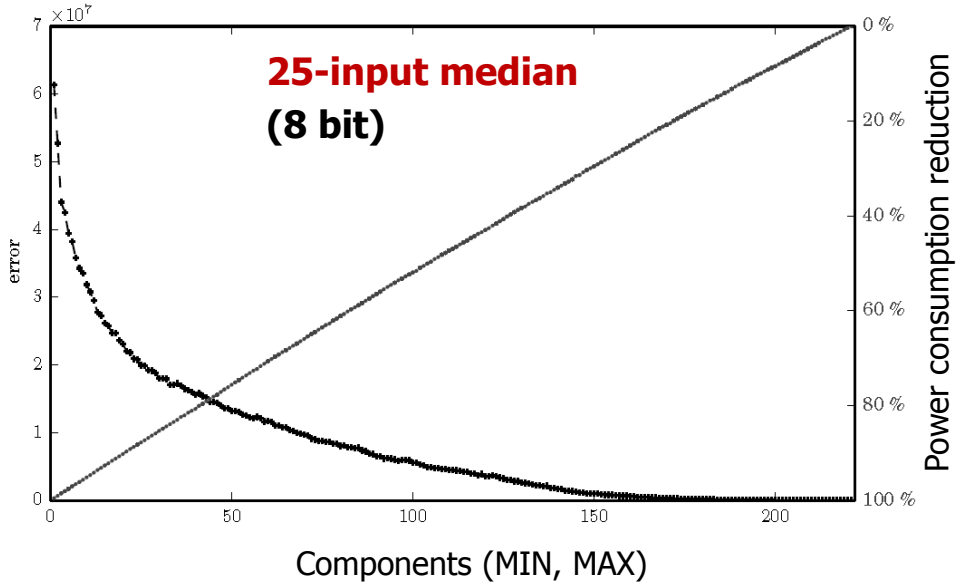
corrupted image
(10% pixels, impulse noise)

filtered image
(9-input median filter)



Functional-level CGP: Approximate median circuits





The error measured using 10^4 test vectors for $w = 9$ and 10^5 test vectors for $w = 25$.

Power consumption and other parameters were estimated during evolution and then validated using an external tool (SIS).

Z. Vasicek and L. Sekanina. *Evolutionary approach to approximate digital circuits design*. IEEE trans. on Evol. Comp. 19(3), 2015

Approximate 9-median as SW for microcontroller

Impl.	Time [μs]			Energy [nWs]		
	STM32	PIC24	PIC16	STM32	PIC24	PIC16
6-ops	2.8	54.5	170.5	86	377	342
10-ops	3.3	70.8	251.5	102	490	504
14-ops	3.9	86.8	336.5	118	600	674
18-ops	4.5	104.5	424.1	138	723	850
22-ops	5.0	116.7	487.8	151	808	978
26-ops	5.9	130.0	558.0	179	900	1118
30-ops	6.0	142.0	627.4	181	983	1257
34-ops	6.4	154.0	819.7	196	1066	1643
38-ops	6.9	165.5	885.0	210	1145	1774
qsort	28.5	1106.2	—	869	7655	—

34.9% error prob.,
max. error dist. 2
52% power reduction

4.8% error prob.,
max. error dist. 1
21% power reduction

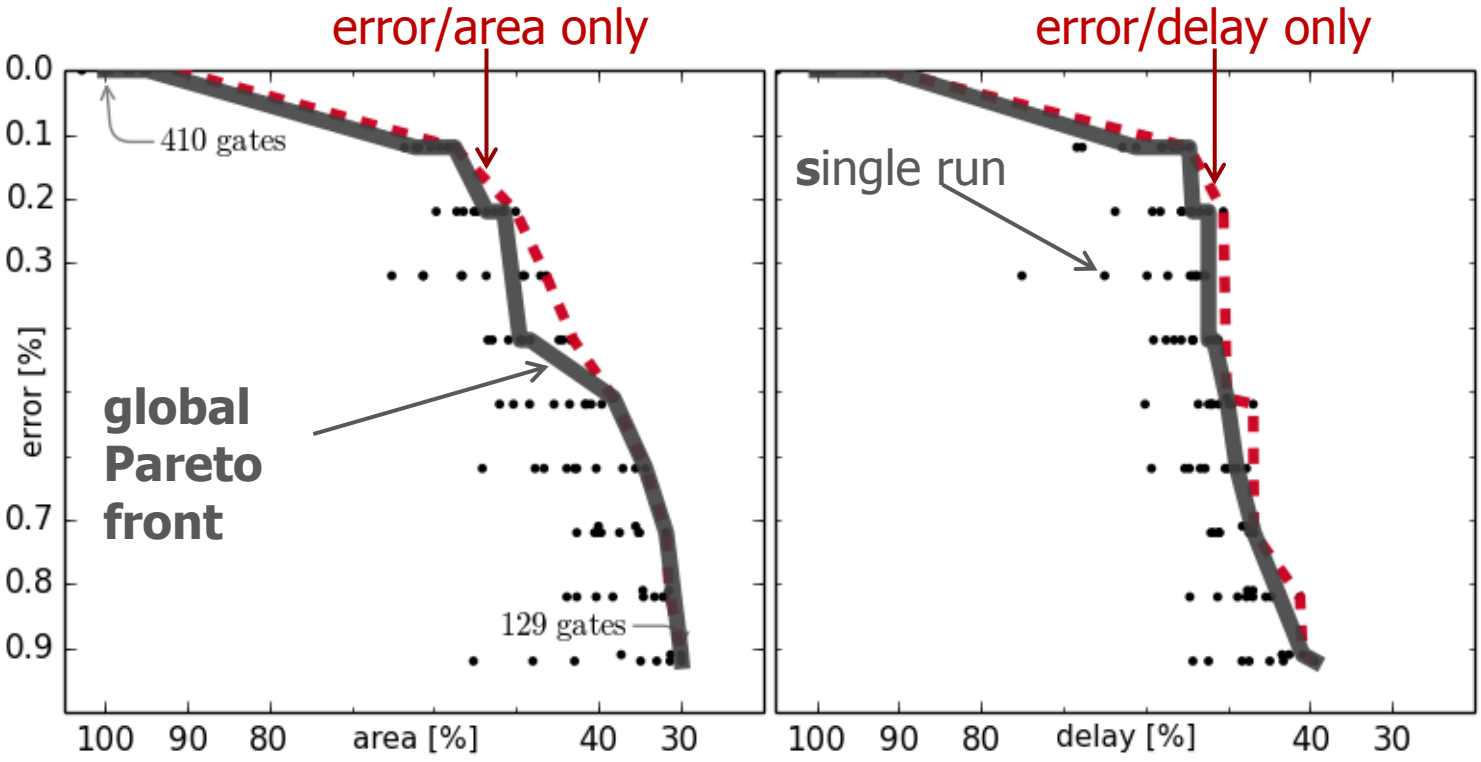
fully-working median

ops = operations in the source code.

```
#define PIX_SORT(a,b) {
 if ((a)>(b))
   PIX_SWAP((a),(b));
}
```

V. Mrazek, Z. Vasicek and L. Sekanina. GECCO GI Workshop, 2015

Circuit approximation: CGP & BDD



- Clmb (bus interface): 46 inputs, 33 outputs
- Original clmb: 641 gates, 19 logic levels, |BDD| = 6966, $|BDD_{opt}|$ = 627 (SIFT in 2.3 s)
- Optimized by CGP (no error allowed):
  - Best: 410 gates, 12 logic levels -- in 29 minutes (2.9×10^6 generations)
  - Median: 442 gates, 13 logic levels

Conclusions

- Evolutionary design has provided innovative designs in many areas of engineering and technology.
  - CGP vs GP vs LGP vs GE vs …
- Recent applications
  - Approximate computing
  - Genetic improvement
- Open problems
  - Scalability
    - Representation
    - Fitness function
  - Non-determinism
  - Trust

Thank you for your attention!

Lukáš Sekanina
Faculty of Information Technology
Brno University of Technology
sekanina@fit.vutbr.cz

IT4Innovations
national
supercomputing
center