Genetic Improvement and Approximation:
From Hardware to Software

Lukáš Sekanina

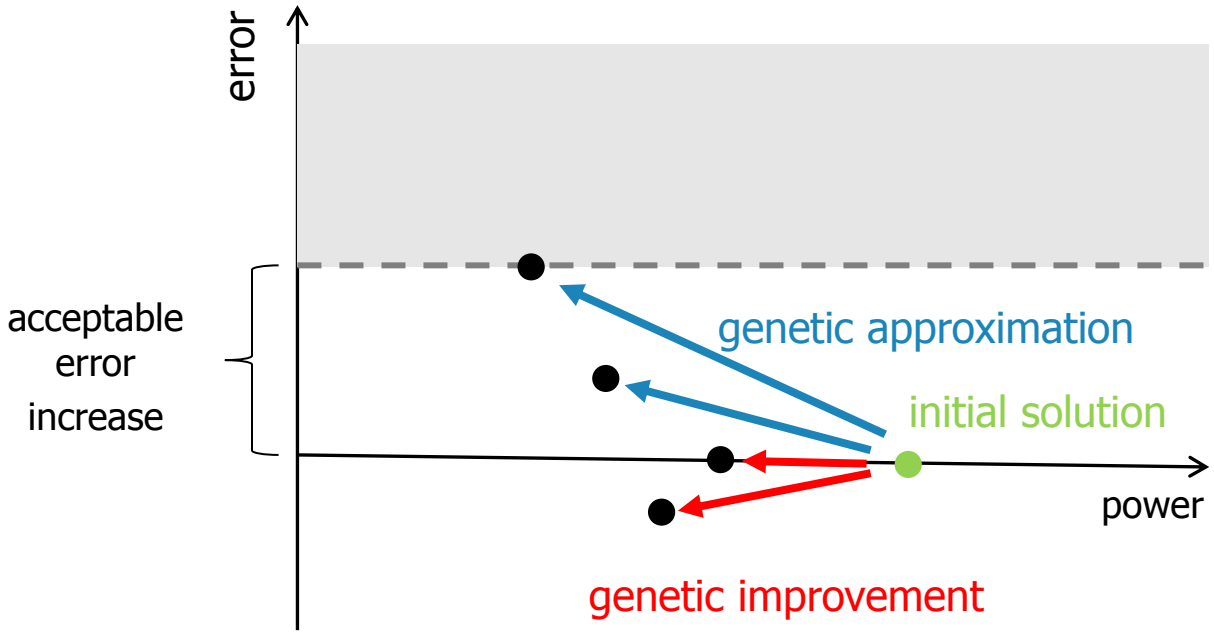
Brno University of Technology, Faculty of Information Technology
Brno, Czech Republic
sekanina@fit.vutbr.cz

IT4Innovations national supercomputing center

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

CREST COW 45, London, January 25 -26, 2016

Genetic improvement and genetic approximation

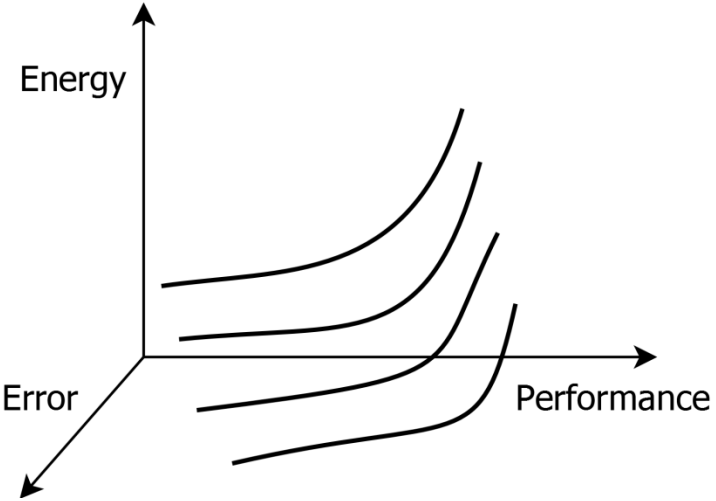


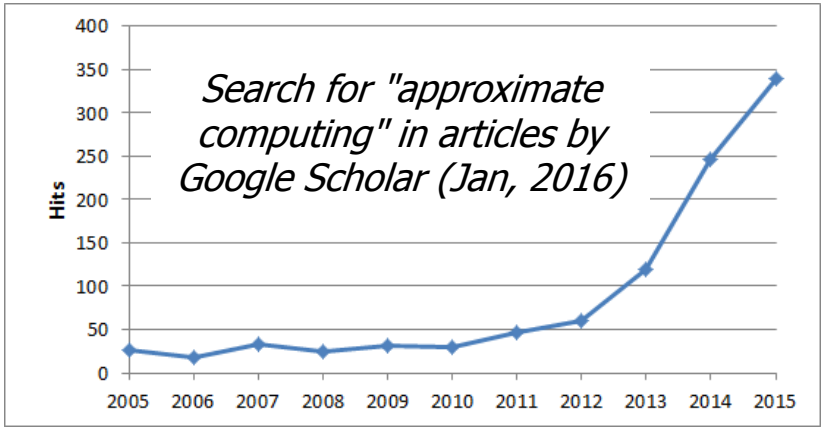
Motivation for Approximate computing

- Variability of circuit parameters for technology nodes < 45 nm is very HIGH
  - Low-power computing, but with unreliable components!
- High performance & low power computing is requested.
  - Many applications are error-resilient - the error can be traded for energy savings or performance.

Functional approximation by means of
Genetic Improvement

Error as a design metric!





Outline

- Genetic improvement of complex digital circuits
- Genetic approximation of complex digital circuits
- Genetic approximation of elementary SW functions for microcontrollers: Median
- Conclusions

HDL – Hardware Description Languages

alu4.pla

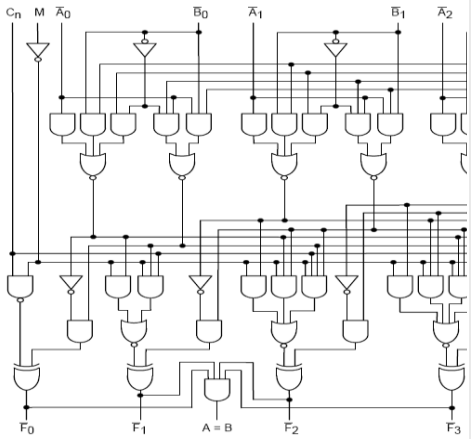
```
.i 14
.o 8
.ilb i_0_ i_1_ i_2_ i_3_ i_4_ i_5_ i_6_ i_7_ i_8_ i_9_ i_10_ i_11_ i_12_
i_13_
.ob o_0_ o_1_ o_2_ o_3_ o_4_ o_5_ o_6_ o_7_
.p 1028
1----1---1---- 10000000
1----0----1--- 10000000
1--------11--- 10000000
-1----1--1---- 10000000
-1----0---1--- 10000000
-1-------11--- 10000000
--1----1-1---- 10000000
etc.
```

Truth table

alu4.blif

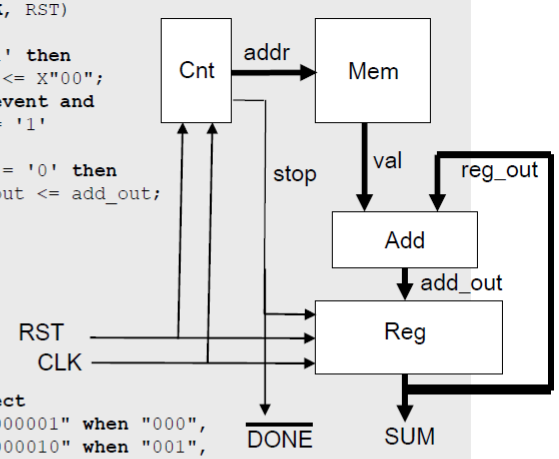
```
.model ./pla/alu4
.inputs i_0_ i_10_ i_11_ i_12_ i_13_ i_1_ i_2_ i_3_ i_4_ i_5_ i_6_ i_7_ i_8_
i_9_
.outputs o_0_ o_1_ o_2_ o_3_ o_4_ o_5_ o_6_ o_7_
.gate NAND A=i_2_ B=_net203568 O=_net196167
.gate NAND A=i_11_ B=_net203428 O=_net196385
.gate OR A=_net204803 B=_net200095 O=o_5_
.gate NOR A=i_0_ B=i_12_ O=_net196891
.gate NAND A=_net203823 B=_net196167 O=_net198561
.gate NAND A=i_1_ B=_net198561 O=_net198562
```

Netlist




```
entity Acc is
port (
   CLK: in std_logic;
   RST: in std_logic;
   SUM: out std_logic_vector(7 downto 0);
   DONE: out std_logic
);
end Acc;

architecture behav of Acc is

signal addr: std_logic_vector(2 downto 0);
signal val: std_logic_vector(7 downto 0);
signal add_out: std_logic_vector(7 downto 0);
signal reg_out: std_logic_vector(7 downto 0);
signal stop: std_logic;

begin

   cnt: process(CLK, RST)
   begin
      if RST = '1' then
         addr <= "000";
         stop <= '0';
      elsif clk'event and clk = '1' then
         if addr = "111" then
            stop <= '1';
         else
            addr <= addr + 1;
            stop <= '0';
         end if;
      end if;
   end process;
```

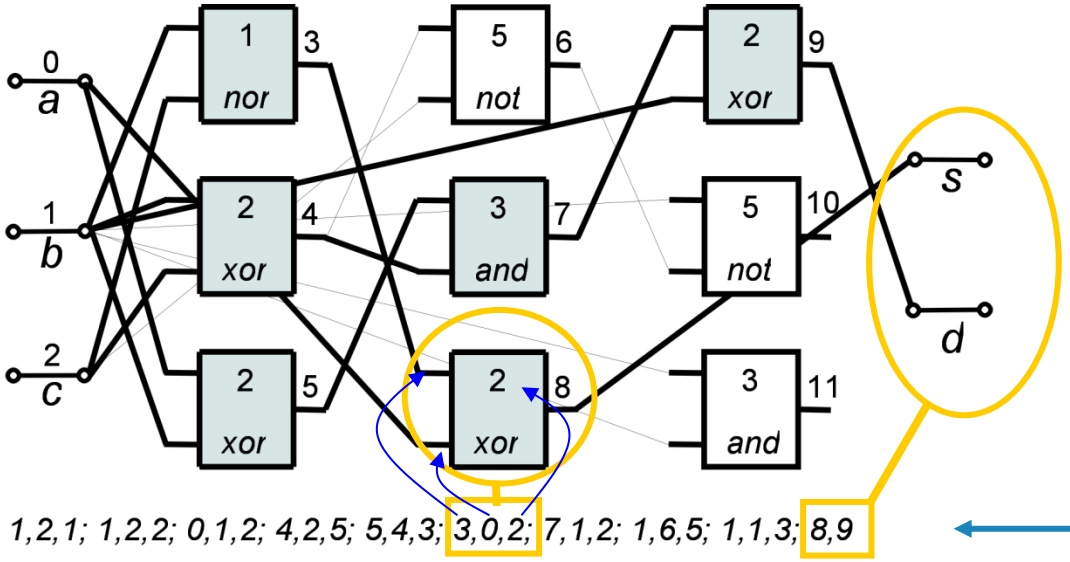
```
reg: process(CLK, RST)
  begin
     if RST = '1' then
        reg_out <= X"00";
     elsif clk'event and
           clk = '1'
     then
        if stop = '0' then
           reg_out <= add_out;
        end if;
     end if;
  end process;
```




```
  with addr select
     val <= "00000001" when "000",
            "00000010" when "001",
            "00000011" when "010",
            "00000111" when "011",
            "00001111" when "100",
            "00011111" when "101",
            "00111111" when "110",
            "01111111" when "111",
            "00000000" when others;

  add_out <= reg_out + val;

  SUM <= reg_out;
  DONE <= not stop;

end behav;
```

VHDL

Digital circuit design with Cartesian GP [Miller 1999]



NETLIST = GENOTYPE

- Example: 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 $^{(5)}$}
- Mutation-based (1+λ) EA

Typical fitness function (circuit functionality):

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

Number of test vectors

Desired response

Circuit response

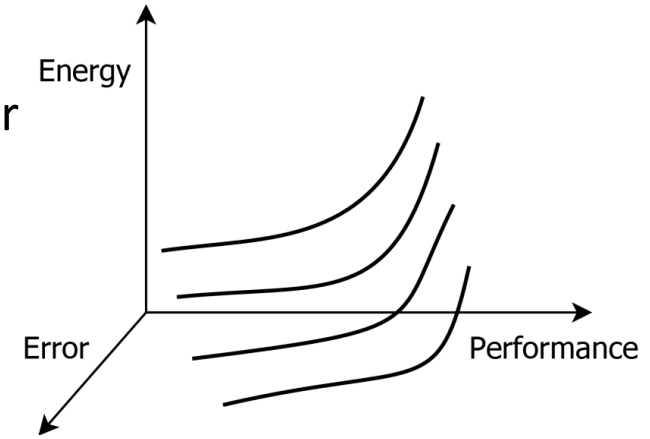
Max: ~20 inputs
Max: ~ tens of gates
No scalable!!!

$K = 2^{inputs}$ for combinational circuits.

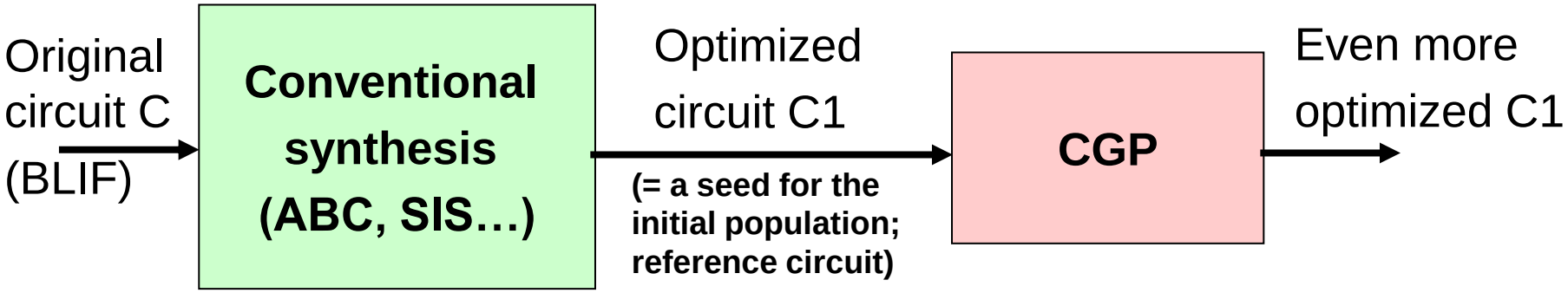
Functionality: Two types of specifications

- Complete specifications
  - A correct output value is requested for every possible input (e.g. for arithmetic circuits)
  - 2^n test cases used to evaluate an n-input circuit
  - Impossible to improve the functionality of a correct solution, only non-functional parameters can be improved.

- Incomplete specifications
  - It is difficult to define correct output values for all possible inputs, e.g. filters, classifiers, predictors, …
  - A circuit with an acceptable error is sought using a training set of k test cases, $k << 2^n$
  - GI can improve functional and non-functional parameters.



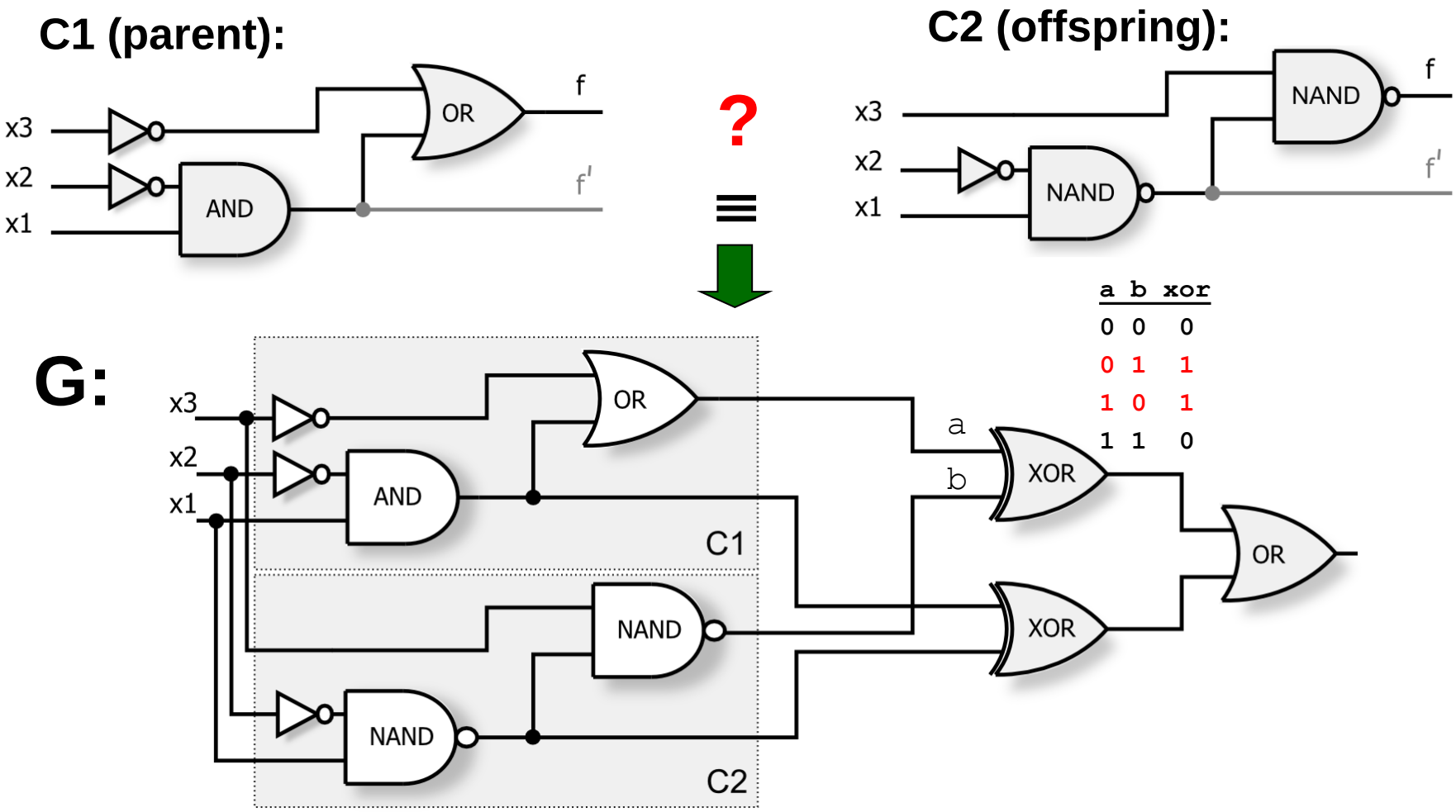
Genetic improvement for complete specifications



- 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.

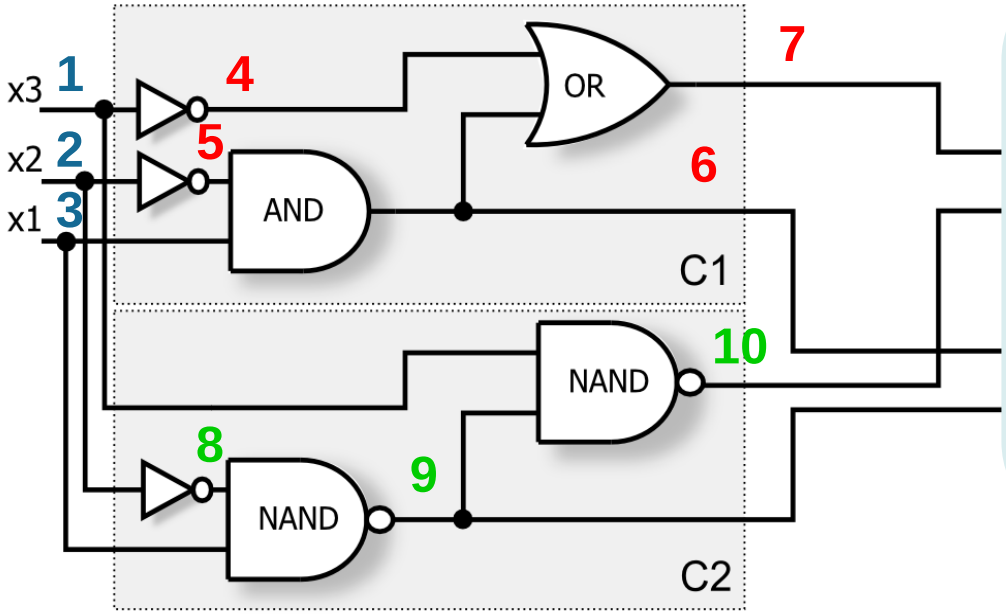
[Vašíček, Sekanina: Genetic Programming and Evolvable Machines 12(3), 2011]

Creating an auxiliary circuit G



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.

Tseitin transform to create CNF for circuit G



Example: y = not (x)

CNF formula g(x, y) = 1 if the predicate y = OP(x) holds true

x	y	g
0	0	0
0	1	1
1	0	1
1	1	0

g = (~x ∨ ~y)(x ∨ y)

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

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

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

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

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

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

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

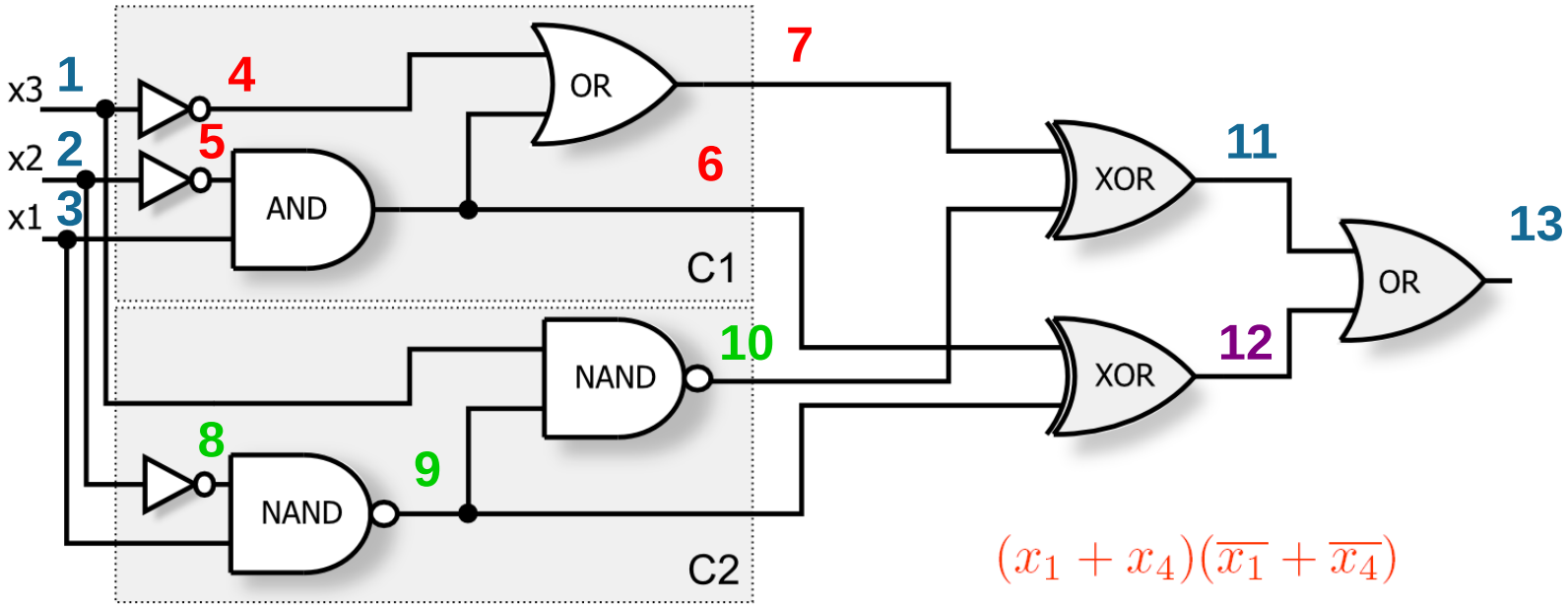
$(\overline{x_7} + x_{10} + x_{11})(x_7 + \overline{x_{10}} + x_{11})(\overline{x_7} + \overline{x_{10}} + \overline{x_{11}})(x_7 + x_{10} + \overline{x_{11}})$

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

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

(x_{13})

SAT solver in action



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

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

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

$x_6)$

$\overline{x_7})$

SAT solver: MiniSAT

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

result: SATISFIABLE / NONEQUIVALENT

model / counter example: 0011111101011

(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 (12 hours each)

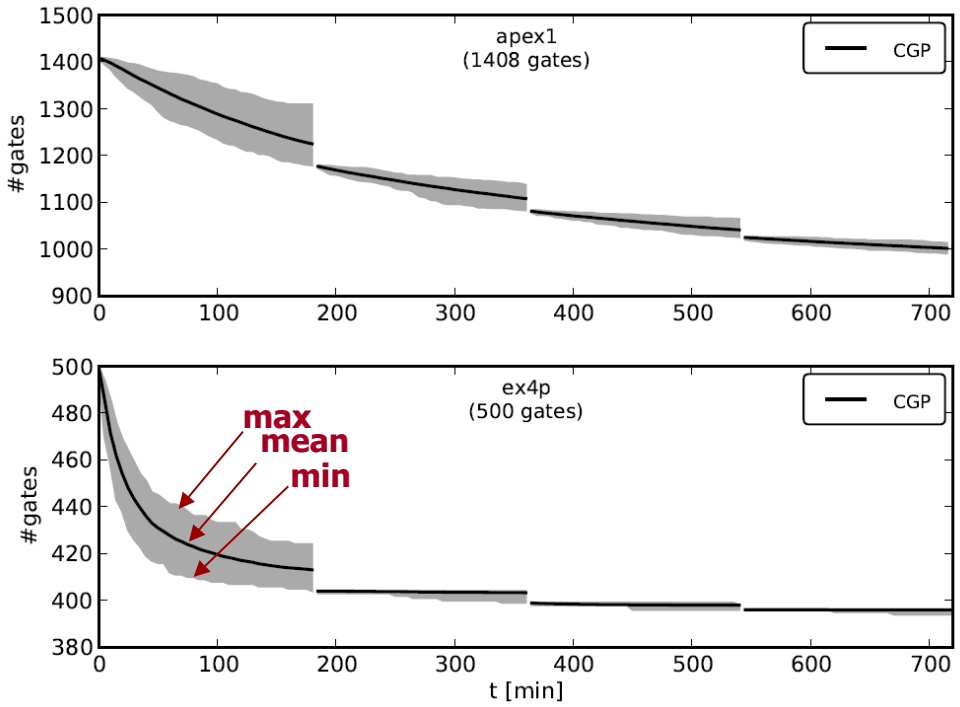
Average area improvement: 25%

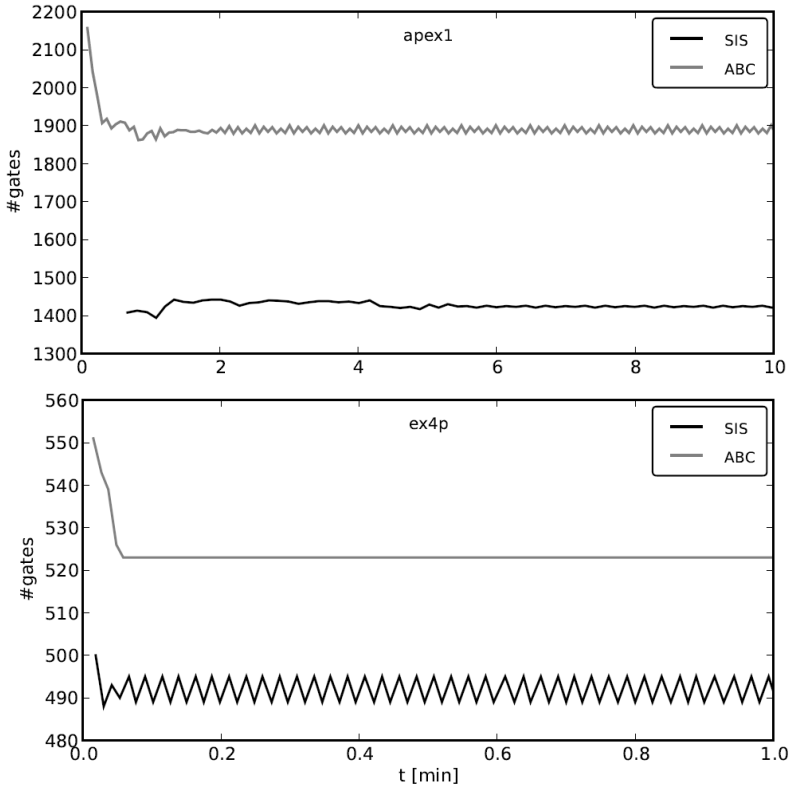
ABC, SIS – conventional open academic synthesis tools – very fast (seconds, minutes)

C1, C2, C3 – commercial synthesis tools

[Vašíček, Sekanina: DATE 2011]

Experiment 1: Convergence curves

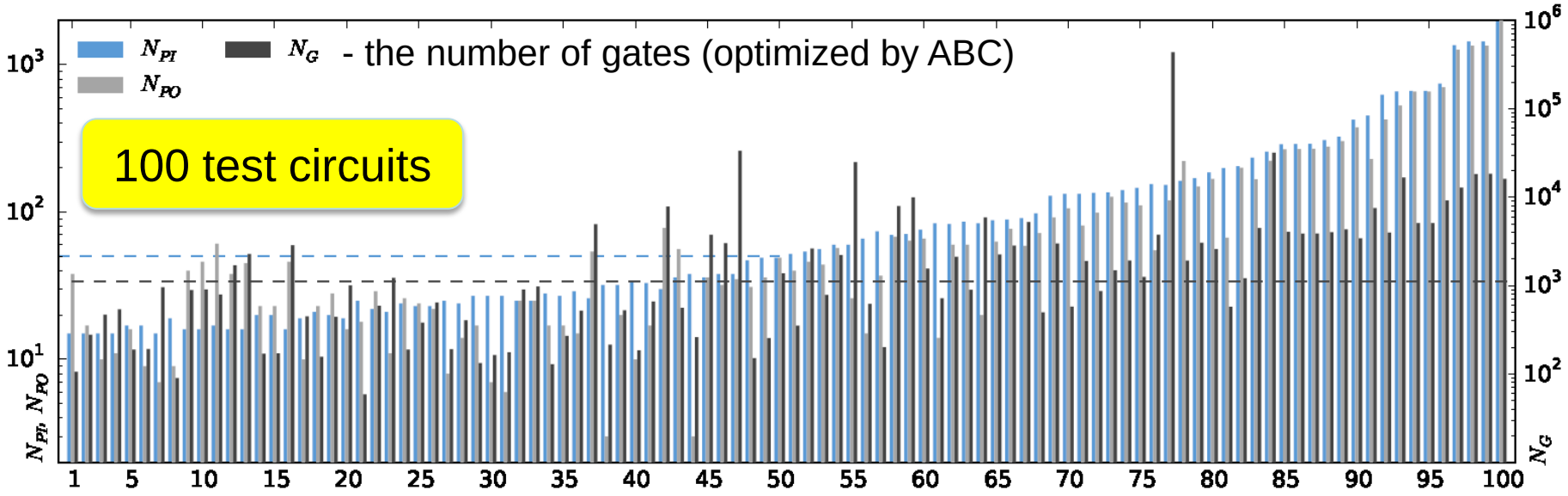




- More time ⇒ better results in the case of CGP
- Current circuit synthesis and optimization tools provide far from optimum circuits!

Experiment 2: SAT solving combined with simulation

SAT solver is called only if the circuit simulation performed for a small subset of vectors has indicated no error in the candidate circuit.



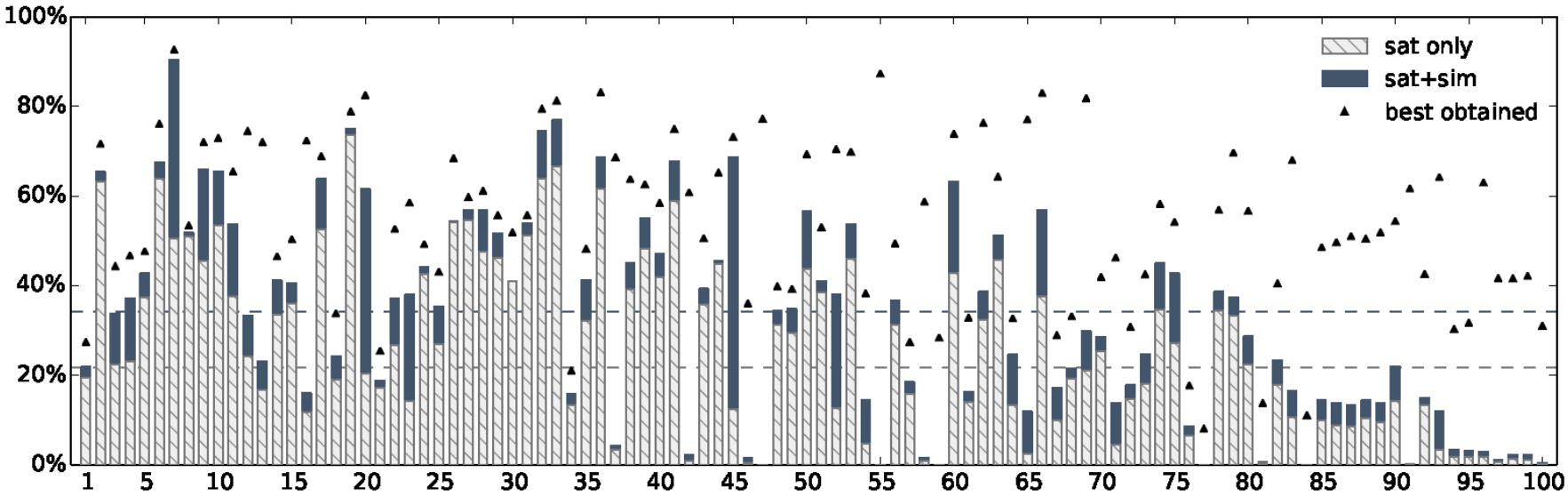
100 combinational circuits (≥15 inputs) - IWLS2005, MCNC, QUIP benchmarks

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)

Experiment 2: SAT solving combined with simulation



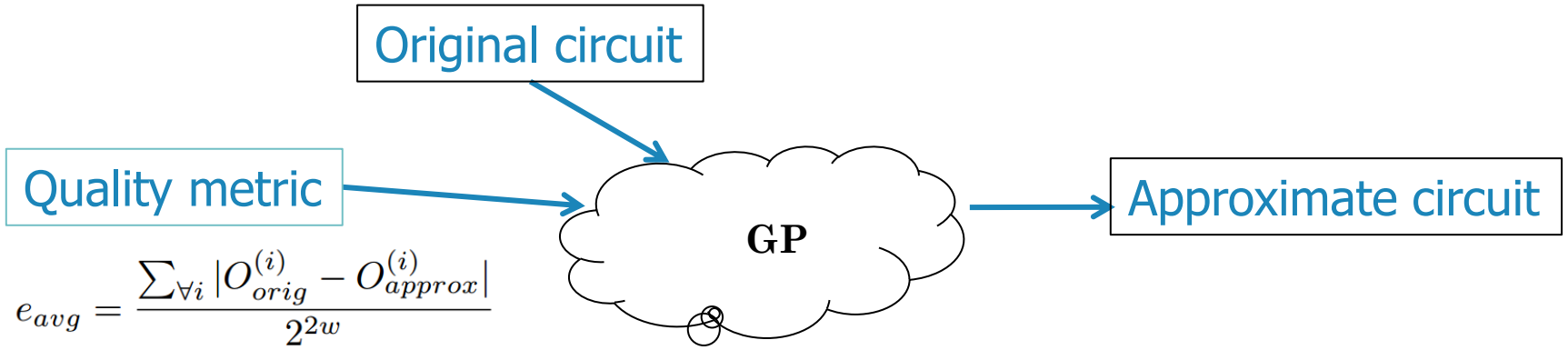
CGP + SAT solver + circuit simulation

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]

Genetic approximation



$$e_{avg} = \frac{\sum_{\forall i} |O^{(i)}_{orig} - O^{(i)}_{approx}|}{2^{2w}}$$

$$e_{wst} = \max_{\forall i} |O^{(i)}_{orig} - O^{(i)}_{approx}|$$

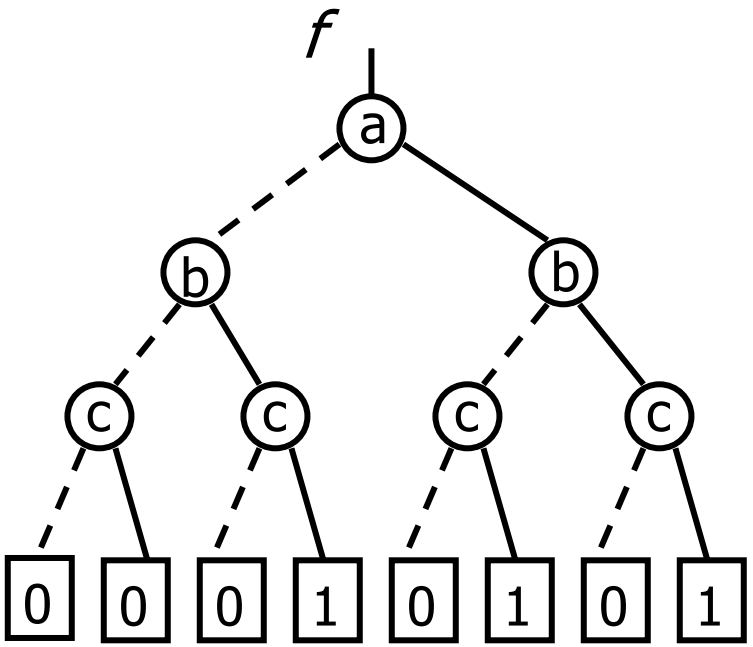
- Relaxed equivalence checking is needed for approximate computing
  - What is the distance between functionality of two circuits?
  - How to calculate this distance for complex circuits when **a simulation using a data set is not accurate**?

The Hamming distance can be obtained using Binary Decision Diagrams for (many useful) complex circuits in a short time!

$f = ac + bc$

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

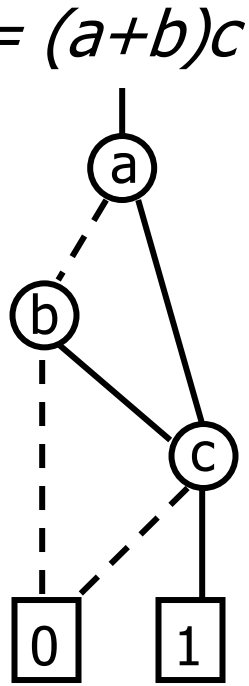
Truth table



Decision tree

1 edge

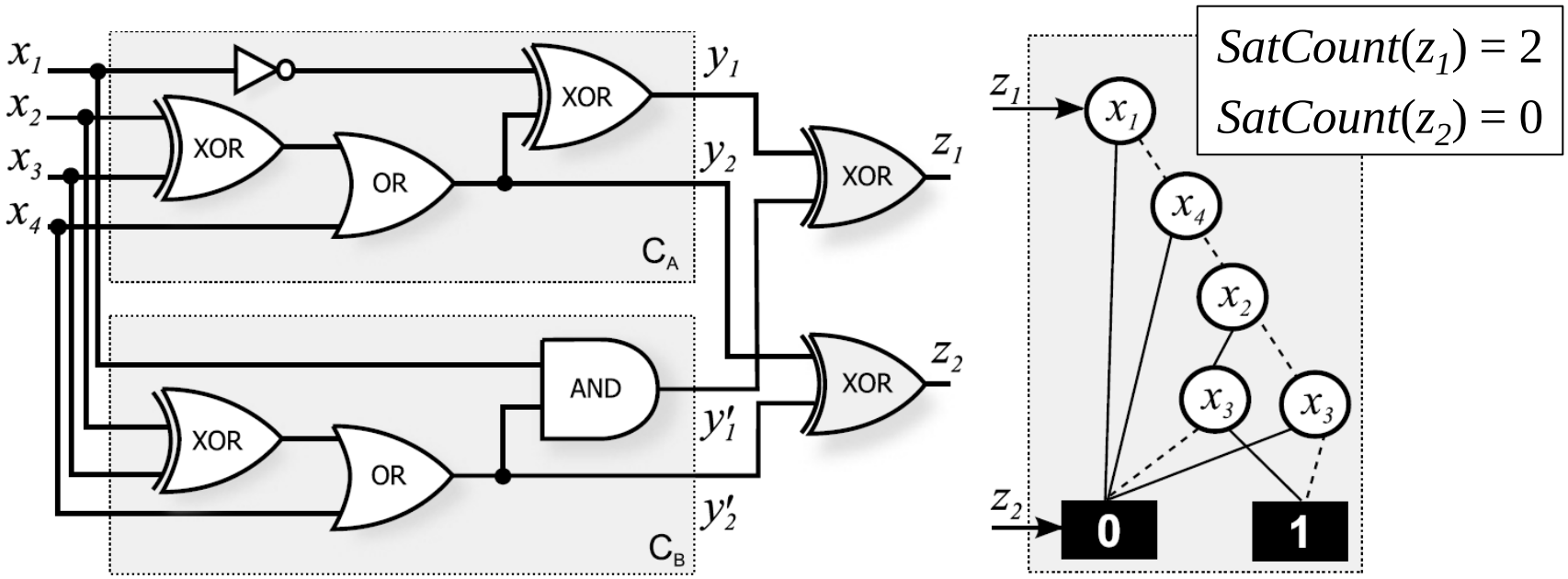
0 edge



Reduced Ordered BDD (ROBDD)

Operations over (RO)BDDs implemented by many libraries, e.g. Buddy.

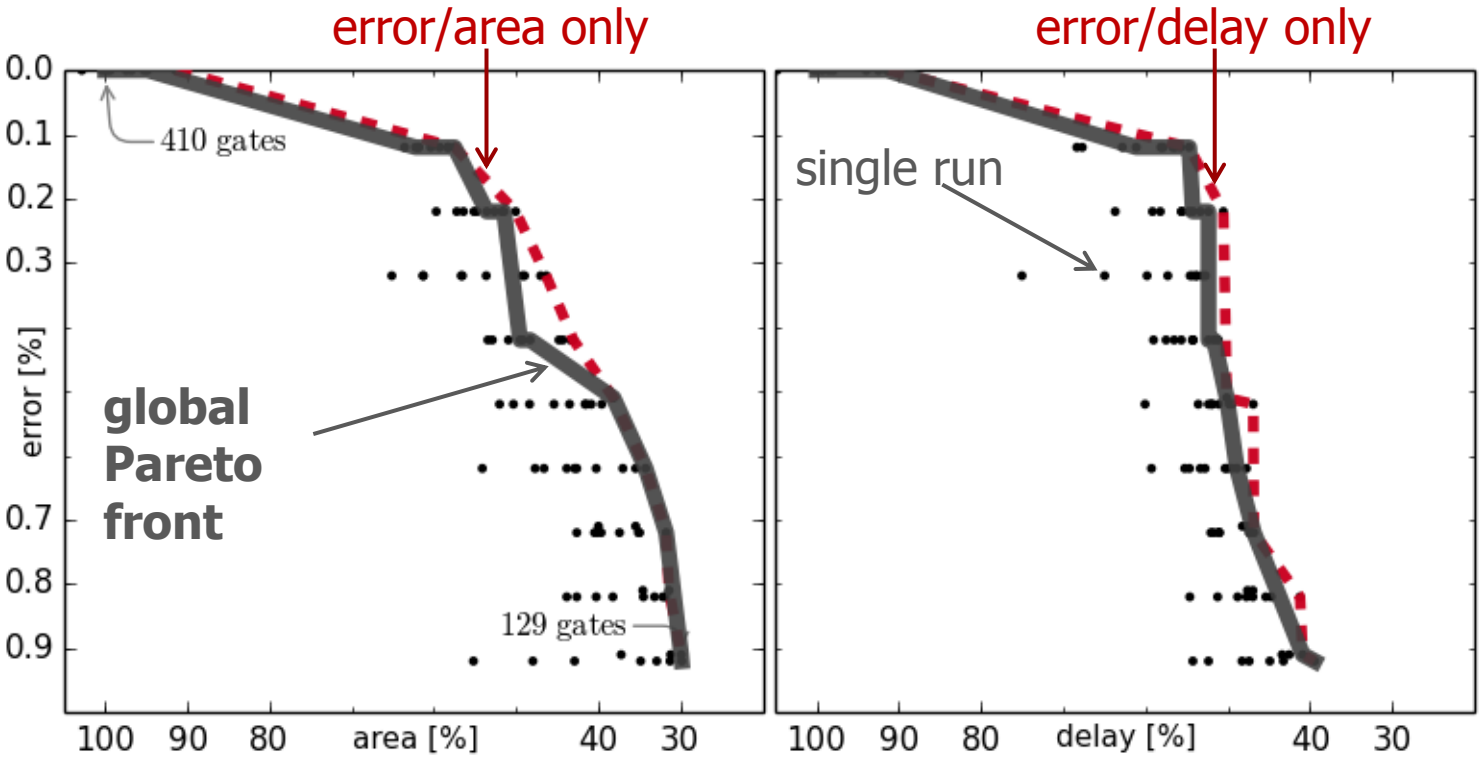
Hamming distance using ROBDD



- Create ROBDD for the parent circuit C_A, the offspring circuit C_B and the XOR gates.
- The error is the average Hamming distance

$$Error(A) = \frac{\sum_{i=1}^{n_o} SatCount(z_i)}{2^{n_i}}$$

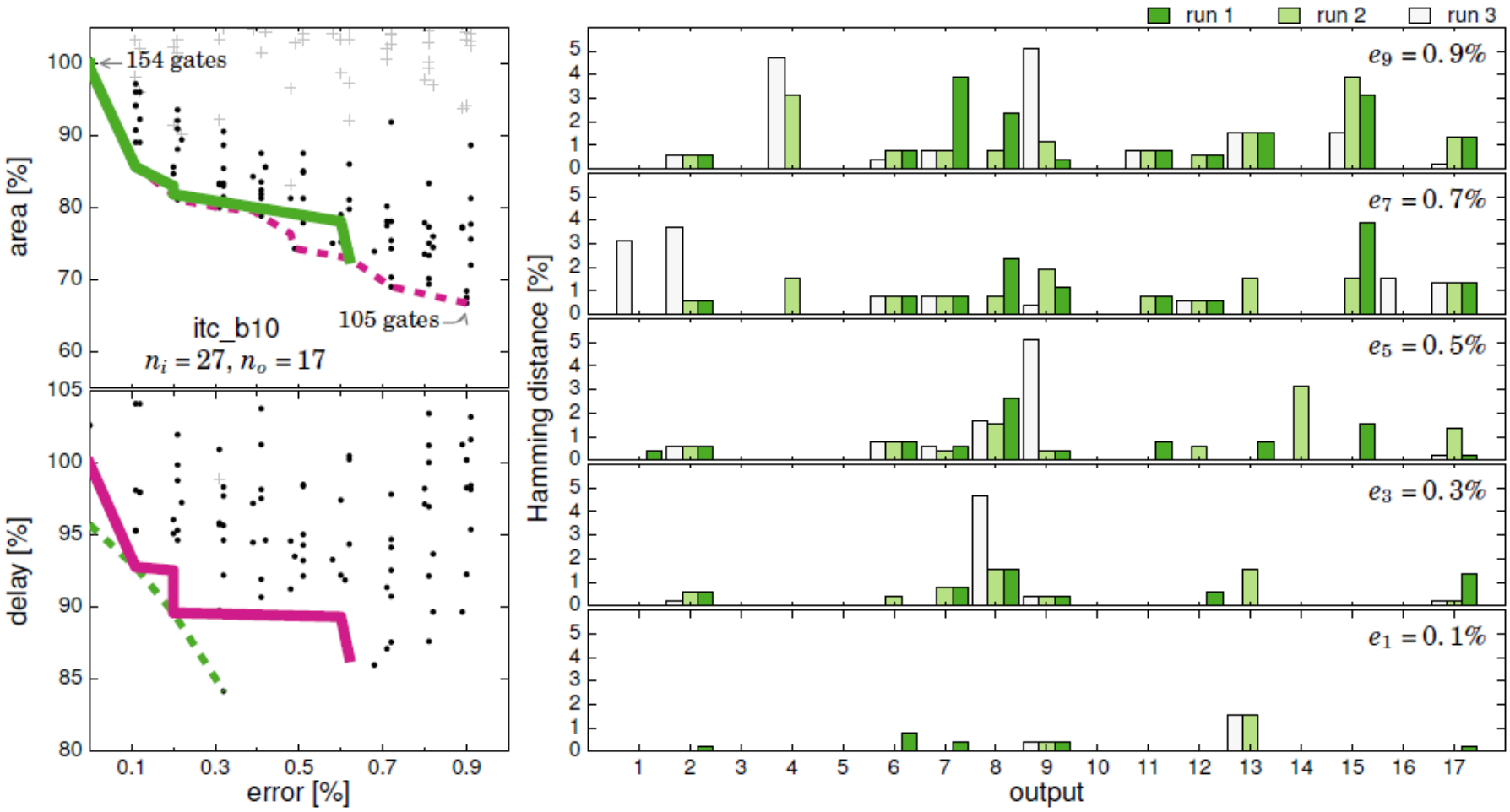
Circuit approximation: Example



- 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

Properly optimize before doing approximations!

Detailed error analysis for itc_b10 circuit

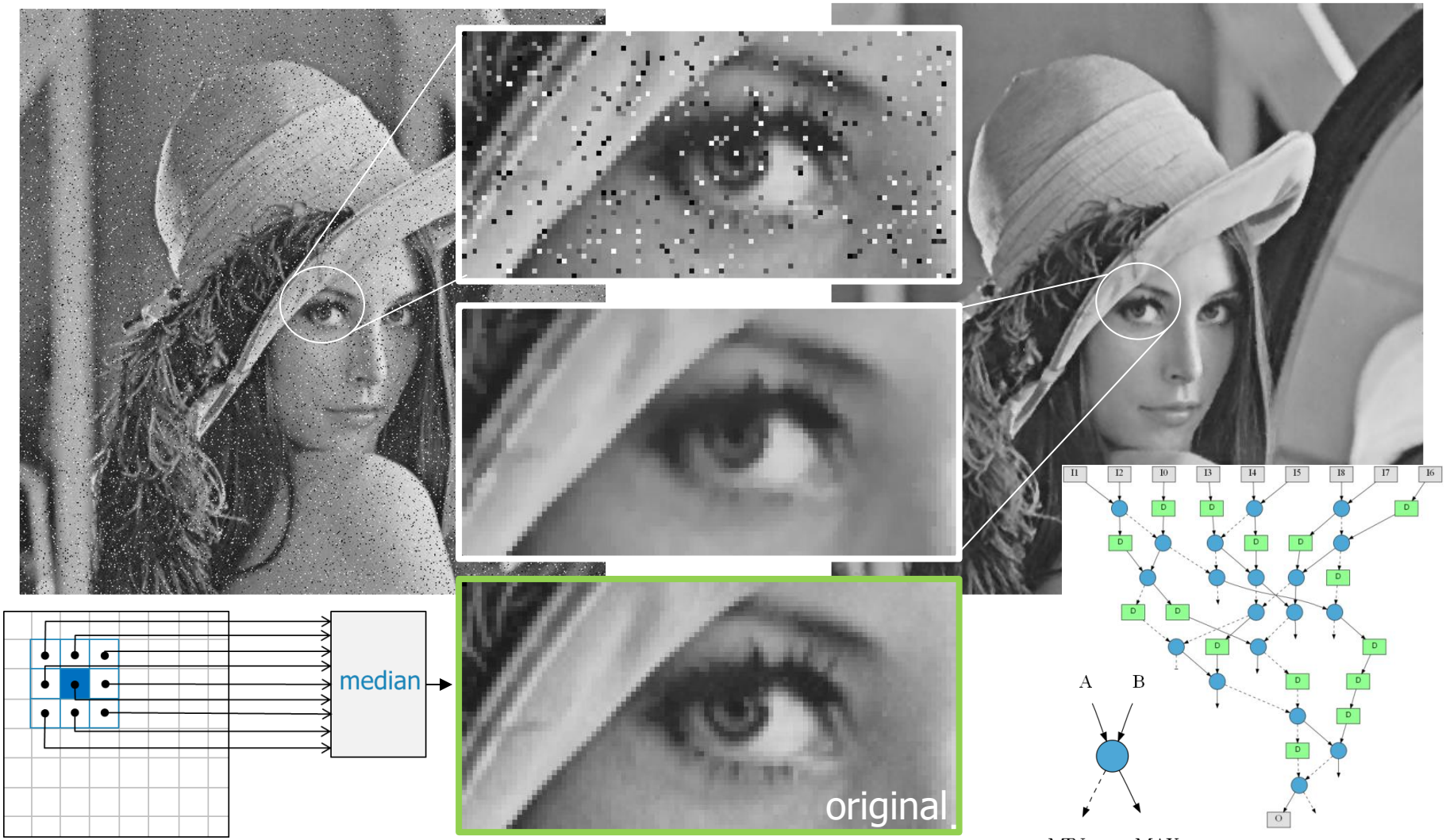


Z. Vašíček and L. Sekanina. Evolutionary Design of Complex Approximate Combinational Circuits. Genetic Programming & Evolvable Machines, 2016, in press.

The median function

corrupted image
(10% pixels, impulse noise)

filtered image
(9-input median filter)

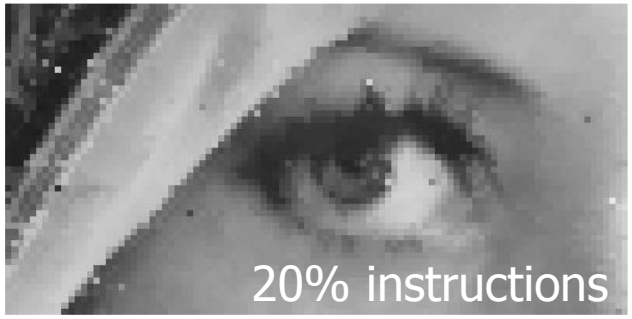


Median as a comparator network

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

```
pixelvalue opt_med9 (pixelvalue * p)
{
    PIX_SORT(p[1], p[2]) ; PIX_SORT(p[4], p[5]) ; PIX_SORT(p[7], p[8]) ;
    PIX_SORT(p[0], p[1]) ; PIX_SORT(p[3], p[4]) ; PIX_SORT(p[6], p[7]) ;
    PIX_SORT(p[1], p[2]) ; PIX_SORT(p[4], p[5]) ; PIX_SORT(p[7], p[8]) ;
    PIX_SORT(p[0], p[3]) ; PIX_SORT(p[5], p[8]) ; PIX_SORT(p[4], p[7]) ;
    PIX_SORT(p[3], p[6]) ; PIX_SORT(p[1], p[4]) ; PIX_SORT(p[2], p[5]) ;
    PIX_SORT(p[4], p[7]) ; PIX_SORT(p[4], p[2]) ; PIX_SORT(p[6], p[4]) ;
    PIX_SORT(p[4], p[2]) ; return(p[4]) ;
}
```

Approximations conducted by means of CGP (and training images):



Approximate 9-median as SW for microcontrollers

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

Conclusions

- Genetic improvement and genetic approximation introduced in the context of circuits described as netlists.
- Complete and incomplete specifications considered.
- The notion of relaxed equivalence checking was introduced.
- Future work
  - Efficient methods of relaxed equivalence checking
    - SAT-based, BDD-based, pseudo-Boolean polynomial representation-based etc.
  - Efficient search methods exploiting properties of a particular relaxed equivalence checking method
  - Real-world case studies

Acknowledgement

- EHW group at Brno University of Technology
  - Zdeněk Vašíček, Michal Bidlo, Roland Dobai
  - Michaela Šikulová, Radek Hrbáček, Vojtěch Mrázek, David Grochol, and other students
- Research projects
  - IT4Innovations Centre of Excellence – National supercomputing center
  - Advanced Methods for Evolutionary Design of Complex Digital Circuits, 2014 – 2016 (Czech Science Foundation)
  - Relaxed equivalence checking for approximate computing, 2016 – 2018 (Czech Science Foundation)

Thank you for your attention!

Lukáš Sekanina

Brno University of Technology, Faculty of Information Technology
Brno, Czech Republic
sekanina@fit.vutbr.cz

IT4Innovations national supercomputing center

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY