Evolutionary Multi-Level Circuit Synthesis in Given Design Style

Overview

SOURCE	***Life as Darwin theory of random mutation under selective pressure***
SURPRIZE	***Utilization for circuit design***
APPROACH	***Partition of search space + target design style + Information measures***
BENEFIT	***Clever algorithms for circuit design + massive parallelism***

Darwin Theory (1859)

Eyes, hands, brain,.... - all of which share characteristics of species: they are the products of the *random mutations* and *genetic mixing* of evolution

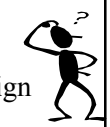
J. Holland (1975)

... idea was to construct a search algorithm modeled on the concepts of natural selection in the biological sciences. The result is a direct random search procedure called *genetic algorithm*

Definition. *Genetic algorithm* is a stochastic search algorithm basing on natural evolution process.

PROBLEM

- How can "creativity" be automated?
- Are engineers necessary to new technology?

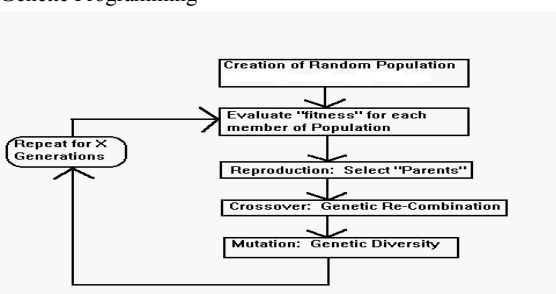


RESEARCH

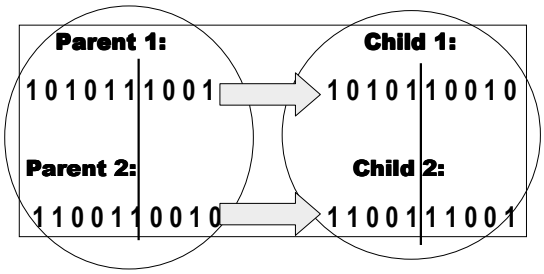
- Biologically inspired evolutionary design process
- Automation of Logic Synthesis and Logic Minimization
- "Computer Designed Computers"

Artificial Genetic Evolution

Basic Process of:
-Genetic Algorithms
-Genetic Programming



Fragment of genetic mixing of evolution in *Holland's* interpretation



Advantages

- *Widely applicable, also in cases where no (good) problem specific techniques are available*
- *Can be run interactively (online parameter adjustment)*
- *No presumptions with respect to the problem space*
- *Low development costs, i.e. costs to adapt to new problem spaces*
- *The solutions have straightforward interpretation*

Disadvantages

- *No solid theoretical basis (yet)*
- *Parameter turning is largely based on trial and error*
- *Often computationally expensive*
- *No guarantee for finding optimal solutions within a finite amount of time (true for all global optimization methods)*

Terminology

Population (set of circuits)

Individual (circuit)

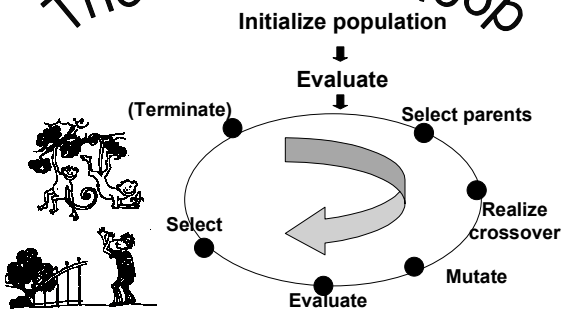
Fitness function (contains all information about the evolving circuit)

Gene (type of gate, inputs and outputs, *etc*)

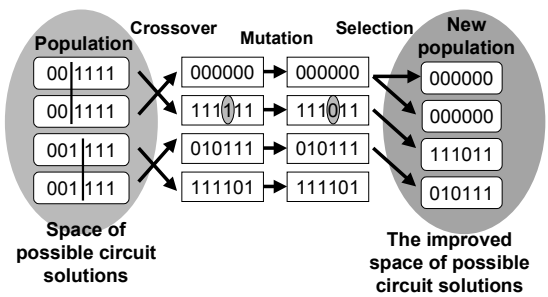
Chromosome (coded circuit)

Probabilistic operators: ***Crossover, Mutation and Selection***

The Evolution loop



Mechanism of genetic algorithm



Definition. *Fitness function* **is a kind of objective, or cost, function which contains *all information** about the problem.**

In biology, fitness is the number of offsprings that survive to reproduction.

In genetic algorithm, one must map objective function to a fitness function

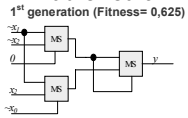
** in our case - all information about the evolving logical network*

Example

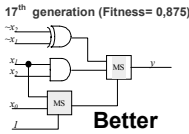
Fitness evaluation

Bad circuit

1st generation (Fitness= 0,625)



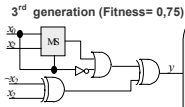
17th generation (Fitness= 0,875)



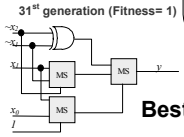
Better

Better circuit

3rd generation (Fitness= 0,75)



31st generation (Fitness= 1)



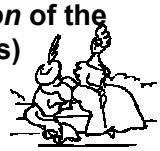
31st generation is the correct circuit

Best circuit

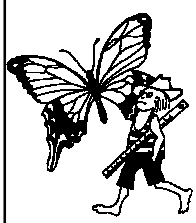
Three probabilistic operators, *crossover,* *mutation* and *selection,* ensure that the best individuals of population will survive, and their *information content* * is preserved and combined to generate even better offspring

Simple crossover

The crossover **operator aims to make a better individual by replacing a part of an individual with a better part of another individual, i.e. combining *valuable information* of the individuals (parents)**



Mutation



The mutation **operator changes certain bit(s) in an individual.**

This operator aims to escape from search space from which individuals cannot escape by means of only crossover operator, i.e. this operator introduces *new information* into the evolutionary process.

Example. ***The string 000110 becomes 001110 if mutation occurs at the third bit***

Selection

The selection **operator chooses good individuals in a population according to their fitness values and the given selection strategy.**

This operator aims to increase better individuals in the population while maintaining certain diversity.

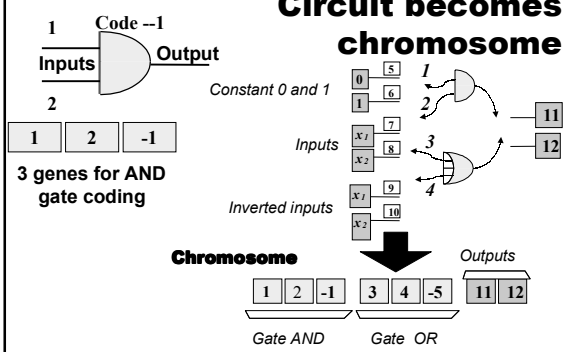
Example. ***The elitism strategy chooses the restricted set of elite individuals***

Crossover+Mutation+Selection = Continuous improvement

The genetic algorithm tries to improve the fitness of the population by combining *information* * contained in high fitness chromosomes

The biggest difficulty of using genetic algorithms is the time which may sometimes be painfully long

Circuit becomes chromosome



Example Genetic Algorithm

REPRODUCTION - Selection of Parent Strings

Bit Strings	Population	Fitness	Tournament Selection	New Generation (Winners)
1	1OO11	3OO	string 4 vs. 2	string 2
2	OOO11	3OO	string 2 vs. 3	string 3
3	111OO	4OO	string 1 vs. 5	string 5
4	1OO1O	2OO	string 3 vs. 4	string 3
5	1111O	6OO	string 4 vs. 1	string 1

Total Fitness = 1800
Average Fitness = 360

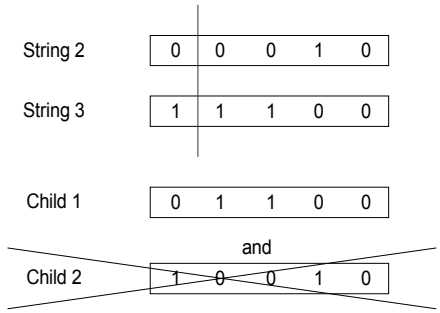
CROSSOVER - Genetic Recombination forming Offspring

New Generation (Offspring)	String Population	Chosen Mate	Mate String	Crossover Bit Site	Resulting String	Fitness of Result
string 2	OOO1O	string 3	111OO	4	O11OO	4OO
string 3	111OO	string 4	1OO1O	1	111OO	4OO
string 5	1111O	string 5	1111O	4	1111O	6OO
string 3	111OO	string 4	1OO1O	3	11O1O	2OO
string 1	1OO11	string 4	1OO1O	2	1OO1O	2OO

Total Fitness (after Crossover) = 1800
Average Fitness (after Crossover) = 360

Example Genetic Algorithm

Crossover Detail:



Example Genetic Algorithm

MUTATION - Genetic Diversity Factor in Offspring

Crossover Population	Result Fitness	Mutation String	Gen=1 Population	Fitness
O11OO	4OO	xxxxx	O11OO	4OO
111OO	4OO	xxxxx	111OO	4OO
1111O	6OO	xxxxx	1111O	6OO
1OO1O	2OO	xx1xx	1O11O	6OO
1OO1O	2OO	xxxxx	1OO1O	2OO

Total Fitness (after Mutation Operation) = 2200
Average Fitness (after Mutation Operation) = 440

Comparison:
Original Total Fitness = 1800
Original Average Fitness = 360

SCHEMA THEOREM: Success Theory of GA

Schemata Propagation in Reproduction

Schemata	Pattern	Bit Strings	Population	Schemata Membership	Tournament Selection	Winners	Schemata Membership
Schema A	1O***	1	1OO11	Schema A,D	string 4 vs. 2	string 2	Schema B,D
Schema B	OOO**	2	OOO11	Schema B,D	string 2 vs. 3	string 3	Schema C,E
Schema C	*11*O	3	111OO	Schema C,E	string 1 vs. 5	string 5	Schema C,E
Schema D	*OO1*	4	1OO1O	Schema A,D	string 3 vs. 4	string 3	Schema C,E
Schema E	11***	5	1111O	Schema C,E	string 4 vs. 1	string 1	Schema A,D

SCHEMA THEOREM: Success Theory of GA

Schemata Propagation in Crossover

Case	Reproduction Population	Schemata Membership	Mate String	Cross Site	Result String	Schemata Membership	Result Fitness
1	OOO11	Schema A, D	11100	4	01100	Schema C	400
2	11100	Schema C, E	10010	1	11100	Schema C, E	400
3	1111O	Schema C, E	1111O	4	1111O	Schema C, E	600
4	11100	Schema C, E	10010	3	11010	Schema E	200
5	10011	Schema A, D	10010	2	10010	Schema A, D	200

Schemata Propagation in Mutation

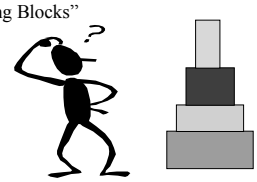
Case	Crossover Population	Schemata Membership	Result Fitness	Mutation String	Gen=1 Population	Schemata Membership	Fitness
1	01100	Schema C	400	xxxxx	01100	Schema C	400
2	11100	Schema C, E	400	xxxxx	11100	Schema C, E	400
3	1111O	Schema C, E	600	xxxxx	1111O	Schema C, E	600

How Genetic Algorithms Work...

Schema (patterns) contain information about solutions!!

Through the genetic operators, the population's schemata collection changes and becomes more refined toward better solutions.

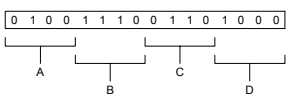
Goldberg: "Short, low-order, and highly fit schemata are sampled, recombined, and resampled to form strings of potentially higher fitness"… "Building Blocks"



Summary of GA Basic Mechanics

Applies an artificial evolutionary process to evolving problem parameters directly

Parameters are represented by a "flat" bit string, which is a direct encode/decode of variable fields



Uses standard Genetic Operators of Reproduction, Crossover, and Mutation

Genetic Programming (GP)

Extension of GA

- **Data Structures (software)**
- **Functions (mathematical & logical operators)**
- **Variables (terminals)**
- **Develops New Algorithms Automatically**

Standard Genetic Operators: Reproduction, Crossover, & Mutation

Bit Strings represent "Trees" (data structures) of different sizes

Most GP research develops new LISP Code

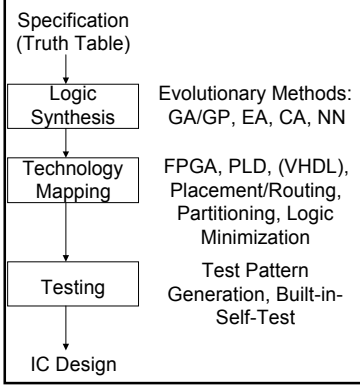
Other Research in Evolutionary Logic Design...

- Evolutionary Algorithms for Computer Aided Design of Integrated Circuits
- "Evolvable Hardware" (EHW) = Evolutionary Computation + Software-Reconfigurable Device (FPGA, etc.)
  - --Online vs. Offline evolution of design
  - --Bottom-up design approach vs. conventional top-down design

Other Research in Evolutionary Logic Design...

- Motivation: Gate-count, Complexity, Time-to-Market, Manpower, $$, …
- CAD Applications: Synthesis, Placement & Routing, Testing
  - --2-level AND-OR logic synthesis with <90 variables, now well solved with conventional CAD Packages/Techniques/Tools
- Performance Evaluation: Quality and Speed

Other Research in Evolutionary Logic Design...



Current Research in Evolutionary Logic Design...

- **JAPAN**
  - --Robotic Control/Navigation: T. Higuchi, et al., ETL
  - --Pattern Recognition Systems; Data Compression: M. Iwata, et al., ETL
  - --Hardware Evolution at Function Level; Adaptive Equalization of Digital Communication Channels; On-line Adaptive Neural Networks: M. Murakawa, et al., U. of Tokyo
  - --ATM Cell Scheduling by Function Level EHW: W. Liu, et al., NEDO
  - --Adaptive Architecture Methodology with Hardware Description Language: H. Hemmi, et al., ATR
  - --CAM (Artificial) BRAIN (evolve NN w/GA): H. de Garis, et al., ATR
- **U.K.**
  - --Robotic Control; Tone Discriminator: A. Thompson, et al., U. of Sussex
  - --Evolving Robot Morphology: H. Lund, U. of Edinburgh

Current Research in Evolutionary Logic Design...

- **SWITZERLAND**
  - -- Self-Reproduction & Repair of Hardware: D. Mange, et al., Swiss Federal Institute of Technology, Lausanne
  - --Phylogenetic, Ontogenetic and Epigenetic (POE) Model; "Firefly Machine" for on-line CA: M. Sipper, et al., Swiss Federal Institute of Technology, Lausanne
  - -- "Bio-dule" (Artificial Cell) Embryonic Electronics, Self-structuring VLSI, Fault Tolerant Hardware: P. Marchal, et al., Centre Suisse d'Electronique et de Microtechnique
- **GERMANY**
  - --Test Pattern Generation; Learning Heuristics; FPRM Logic Logic Minimization: R. Drechsler, et al., U. of Freiburg
  - --VLSI Routing: N. Gockel, et al., U. of Freiburg
- **U.S.A.**
  - --Analog Circuit Design: J. Koza, et al., Stanford University

Growing Digital Circuits

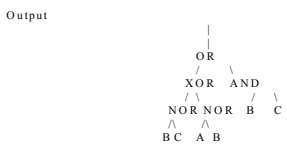
In the Pacific Northwest (Portland, Oregon, USA), we live in the "Silicon Forest" and now we can grow a "forest" in the silicon.

GP Logic Synthesis

This research applies GP to Logic Synthesis

Given: Truth table

Problem: Evolve a logic expression which specifies or "covers" the i/o's of the truth table



(OR (XOR (NOR B C)(NOR A B))(AND B C)) or [!(B + C) ⊕ !(A + B)] + (BC)

Genetic Programming Code

Public Domain

General Evolutionary Workhorse

Reproduction, Crossover, & Mutation

Originally written for "Artificial Ant" and Lawnmower Problems

Extensive Modification/Customization for Logic Synthesis Problem

Allows Other Researchers to Duplicate Results

Available via anonymous ftp to: ftp.cc.utexas.edu in the pub/genetic-programming/code directory

Written by: Adam Fraser, Ph.D. Student, Dept. of Electronic & Electrical Engineering, Cybernetics Research Institute, University of Salford, Salford, U.K.

Comparison Example: Conventional Vs. GP Synthesized Logic

Conventional Logic - SOP Form
f(a,b,c,d) = Σ(0,4,5,7,8,9,13,15)

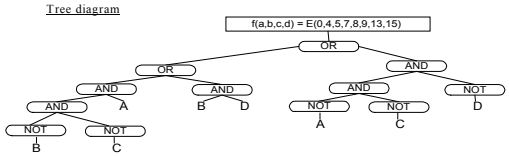
K-map for f(a,b,c,d) = Σ(0,4,5,7,8,9,13,15)

AB \ CD	00	01	11	10
00	1			
01	1	1	1	
11		1	1	
10	1	1		

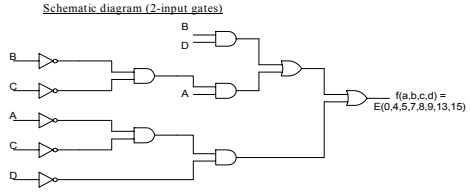
F = a'c'd' + bd + ab'c'

Conventional Logic Design - SOP Form

Tree diagram



Schematic diagram (2-input gates)



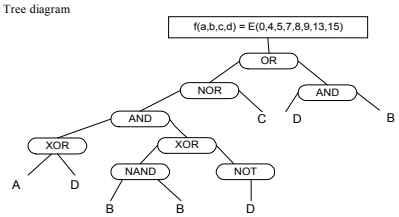
GP Synthesized Logic

f(a,b,c,d) = Σ(0,4,5,7,8,9,13,15)

Synthesized Equation:

((or (and term_D term_B) (nor (and (xor term_A term_D) (xor (nand term_B term_B) (not term_D))) term_C)))
Fitness : 1584
Structural Complexity : 16

Tree diagram



GP Synthesized Logic

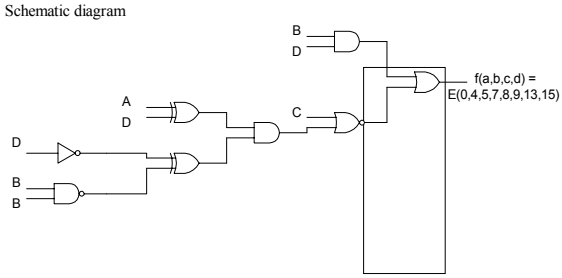
f(a,b,c,d) = Σ(0,4,5,7,8,9,13,15)

Synthesized Equation:

((or (and term_D term_B) (nor (and (xor term_A term_D) (xor (nand term_B term_B) (not term_D))) term_C)))
Fitness : 1584
Structural Complexity : 16

GP Synthesized Logic

Schematic diagram

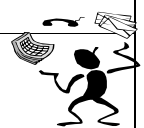


Unconventional Design = unusual choice of gates

Logic Synthesis Experiments

Types of Logic Gates
Population Sizes
Mutation Probability Rates

Objective:	**To determine optimum general parameters for GP-Logic Synthesis problems.**
Terminal Set:	4 Variables: A,B,C,D; 5 Variables: A,B,C,D,E; 6 Variables: A,B,C,D,E,F; 7 Variables: A,B,C,D,E,F,G
Population Size:	**1000-5000**
Mutation Prob. Rate:	**0, 1/10000, 1/1000, 1/100, 1/10, 1**
Function Set: (all are 2-input gates, except the 1-input NOT gate)	**Case 1: and, or, not** **Case 2: and, not** **Case 3: and, or, not, nand** **Case 4: and, or, not, nand, nor** **Case 5: and, nor** **Case 6: and, or, not, xor** **Case 7: and, or, not, xor, nand, nor** **Case 8: nand**
Fitness Measure:	+100 points for each correct truth-table output, -1 point for each logic gate and terminal in solution (for optimization of size)
Criterion:	Goal: to achieve fitness as close as possible to 2^n Perfect fitness is (2^n) - number of gates or terminals, (where n is the number of input variables)
Termination:	50 generations



Empirical Experimental Results

4 Variable Functions
- Test 1: f(a,b,c,d) = Σ(0,4,5,7,8,9,13,15)
- Test 2: f(a,b,c,d) = Σ(4,6,7,15)

5 Variable Functions
- Test 3: f(a,b,c,d,e) = Σ(5,6,9,10)
- Test 4: f(a,b,c,d,e) = Σ(1,2,6,7,9,13,14,15,17,22,23,25,29,30,31)

6 Variable Functions
- Test 5: f(a,b,c,d,e,f) = Σ(1,7,11,21,30)
- Test 6: f(a,b,c,d,e,f) = Σ(10,12,14,20,21,22,25,33,36,45,55)

7 Variable Functions
- Test 7: f(a,b,c,d,e,f,g) = Σ(20,28,52,60)
- Test 8: f(a,b,c,d,e,f,g) = Σ(20,28,38,39,52,60,102,103,127)

Empirical Experimental Results

Types of Logic Gates

Population Size:	1000
Mutation Prob. Rate:	1/1000
Function Set: (all are 2-input gates, except the 1-input NOT)	Case 1: and, or, not Case 2: nand, not Case 3: and, or, not, nand Case 4: and, or, not, nand, nor Case 5: and, nor Case 6: and, or, not, xor Case 7: and, or, not, xor, nand, nor Case 8: nand
Fitness Measure:	+100 for each correct minterm -1 for each logic gate/terminal
Criterion:	Fitness is ((2^n) - gates/terminals)*100 Where n is the number of input variables
Termination:	50 generations

Test	Correct/ Total Minterms	Cover	Gates/Terms in Function Logic Gates Case: 1	2	3	4	5	6	7	8
1	16/16	100.0%	X	X	X	19	19	29	16	X
2	16/16	100.0%	11	18	12	15	13	10	10	19
3	32/32	100.0%	27	X	29	X	X	18	9	X
4	31/32	96.9%	X	17	17	20	X	14	13	X
5	62/64	96.9%	X	X	X	X	X	X	*16	87
6	58/64	90.6%	X	X	23	X	X	26	X	X
7	128/128	100.0%	X	X	10	X	X	X	13	X
8	126/128	98.4%	X	X	X	X	X	X	X	63

*61/64 minterms correct
X = Only best function coverage results listed in table.

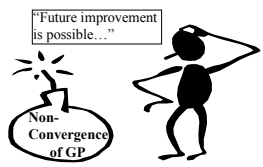
Empirical Experimental Results

Population Size

Population Size:	1000, 2000, 3000, 4000, 5000
Mutation Prob. Rate:	1/1000
Function Set: (all are 2-input gates, except the 1-input NOT)	and, or, not, xor, nand, nor
Fitness Measure:	+100 for each correct minterm -1 for each logic gate/terminal
Criterion:	Fitness is ((2^n) - gates/terminals)*100 Where n is the number of input variables
Termination:	50 generations

Test	Correct/ Total Minterms	Cover	Population Sizes: 1000	2000	3000	4000	5000
4	32/32	100.0%	X	X	31	30	13
5	62/64	96.9%	X	X	18	19	X
6	59/64	92.2%	X	X	X	X	27
8	124/128	96.9%	X	X	X	X	21

X = Only best coverage results listed in table



More Logic Synthesis Experiments...

Don't Cares vs. Function Coverage

Objective:	To investigate the training set size necessary for function learning. This characterizes the learning method's immunity to noise.
Population Size:	1000
Mutation Probability Rate	1/1000
Function Set: (all 2-input gates, except the 1-input NOT gate)	AND, OR, NOT, XOR NAND, NOR
Fitness Measure:	+100 points for each correct truth-table output, -1 point for each logic gate and terminal in synthesized equation (for optimization of structural complexity)
Criterion:	Fitness is ((2^n) - number of gates/terminals) (where n is the number of input variables)
Test Conditions:	Training Sets (portions of original function) from 0-100% of total truth table, in 5% increments
Termination:	100 Generations

Experiments:
1. 9Sym*
2. Majority*
3. 6 Variable Function, Test 6: f(a,b,c,d,e,f) = Σ(10,12,14,20,21,22,25,33, 36,45,55)

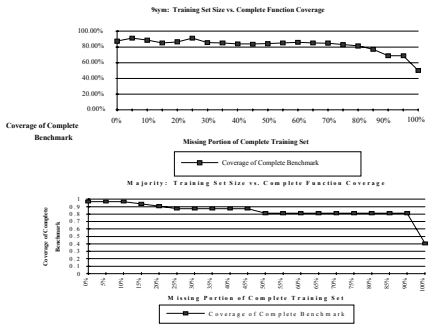
Empirical Experimental Results

Don't Cares vs. Function Coverage

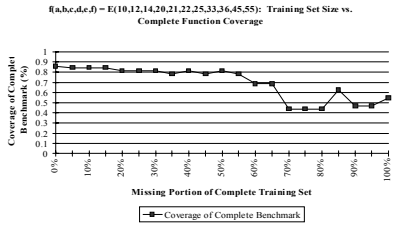
Test	Coverage of Complete Benchmark										
	Percent don't cares										
	0	10	20	30	40	50	60	70	80	90	100
9sym	87%	89%	86%	85%	84%	84%	86%	85%	81%	69%	50%
Maj.	97%	97%	91%	88%	88%	81%	81%	81%	81%	81%	41%
Test 6	86%	84%	81%	81%	81%	81%	69%	44%	44%	47%	55%

Results: Generally, while missing ~80% of the data for a training set, the GP-Logic Synthesis achieved synthesis of ~80% (correct) total function coverage. In other words, given a very small portion of a data file, the GP-Logic Synthesis can synthesize the logic with about an 80% accuracy.

Empirical Experimental Results



Empirical Experimental Results



Result Summary

Types of Logic Gates:

•Large Gate Selection (AND, OR, NOT, XOR, NAND, NOR)

•The universal gate NAND (alone), sometimes showed good results

Population Sizes:

•Improved coverage with larger populations

•Theorized that larger populations increase the total pool of genetic diversity, increasing available traits and characteristics

•But, larger populations slow the rate of evolution, by increasing necessary computations

Mutation Rates:

•Small mutation rates usually introduce an appropriate amount of diversity not already available in the population

•Mutation Rates must be moderate:

-Too small: no diversity available as the evolutionary process converges

-Too big: unbounded diversity creates a chaotic environment

Result Summary

Don't Cares versus Function Coverage:

•Observed that only small training sets are necessary for function recognition

Experimental Results: All tests conducted showed >80% function coverage achieved with training sets missing <80%, 90%, and 55% of their complete truth tables (9Sym, Majority, and "Test 6" 6-variable function).

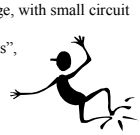
•Results may be biased by the amount of "pattern-ness" present in the test functions, but natural functions usually contain a high degree of pattern.

•Need more experimental data.

Gate-Level Synthesis - Scalability

•Necessary to understand and perfect research in early stage, with small circuit designs, i.e. GP non-convergence problem

•Larger circuit designs will naturally require "gate modules", i.e. (Adder, Multiplexor, etc.)



Future GP-Logic Synthesis Research...

Use Circuit Modules (Adders, Comparators, Multiplexors, etc.) as “functions”, (Automatically Defined Functions)

Create Custom Gate Modules

Apply research to larger functions/designs and Standard Benchmarks

System Design - Computer Architectures

Reduce Synthesis Error!

Goal: 100% Synthesized Function Design Coverage

Future Design Tool

New approach

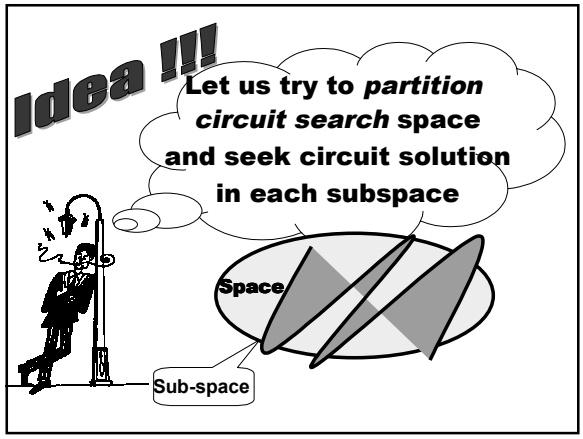
Problem and motivation

The search time depends considerably on the size of the hypothesis space.

A large hypothesis space makes it difficult to find the optimal circuit in a reasonable time (*Sometimes run time for evolving simple network requires dozens of hours)*

Task formulation

Synthesize a logical network in a given design style without *no special software* to implement a design style.



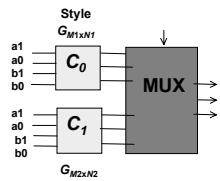
Decomposed 2-bit adder

Good news:

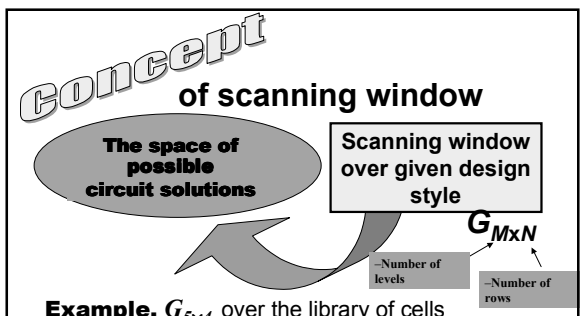
We can partition circuit search space

Bad news:

We need multiplexer



2-bit adder was evolved independently as two sub-circuits C_0 and C_1 .To merge these "pieces" we need a multiplexer - MUX



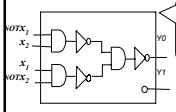
Example. G_{5x4} over the library of cells $L=\{AND,OR\}$ can be considered as "a guide" to design M-level, $M \in \{2,3,4,5\}$, networks under different scenarios

Benefit

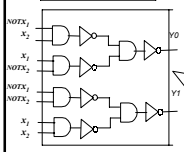
Scenario of evolutionary design

Scenario 1

Style G_{4x2} (invalid circuit)

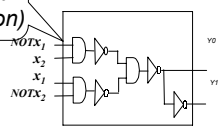


Style G_{4x4} (non optimal circuit)



Scenario 2

Style G_{5x2} (good solution)



x_1	x_2	Y_0	Y_1
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

Over library
□={*AND, NOT*}

Experiment

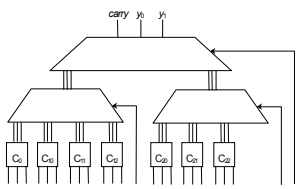
EvoDesign against SIS (*gate/time*)

Test	EvoDesign	SIS [Berkeley,1994]
sao2	165 / 120 *sec*	203 / 0.15 *sec*
rd53	16 / 57*sec*	34 / 0.1 *sec*
life	34 / 41 *sec*	138 / 0.2 *sec*
misex1	61 / 49 *sec*	155 / 0.4 *sec*
Total	276 / 267 *sec*	530 / 0.85 *sec*

Run time is terrible !!!

EFFECT 2 times

Generalization for 3-valued 2-digit multiplier



 # sub-circuits - 7

 # gates - 73

 # MUX - 3

Run time: 249 min

EvoDesign for synthesis ternary and quaternary 2-digit multipliers

Test	Gate	Time
2mult-3	73 gates +3 MUXs	249 *min*
2mult-4	220 gates+5 MUXs	148 *min*

Observation. *When parallel and independent processing of network space, the different target design styles can be used for each subspace*

Experiment

Quaternary circuits (*gate/time*)

Test	Direct	Parallel
monks1tr	6 / 3 *hours*	7 / 0.3 *hours*
monks1te	5 / 4 *hours*	7 / 1,6 *hours*
monks3tr	-	18 / 6,1 *hours*

The maximal time for processing of one subspace

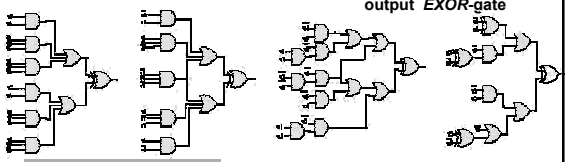
Concept of target design style

- Library of cells $L=\{AND, OR, EXOR, NOT, NAND, NOR\}$
- Number of levels
- Permissible interconnections between cells,
- Types of gate in each level

EvoDesign against Sasao method

Design style:

END-OR-EXOR 3-level network with a single-output *EXOR*-gate



Sasao method

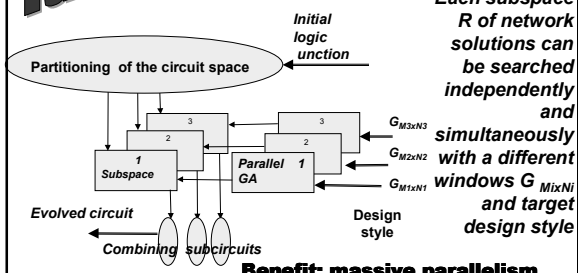
EvoDesign algorithm

(I) Population size - 60 (ii) Maximal number of generations - 10^4

(iii) Crossover - 0.7 (iv) Level back - 3 (v) Tournament selection and - discriminator is 2 and 90%

Idea of massively parallel circuit design

Assumption. *Each subspace R of network solutions can be searched independently and simultaneously with a different windows G_{MixNi} and target design style*

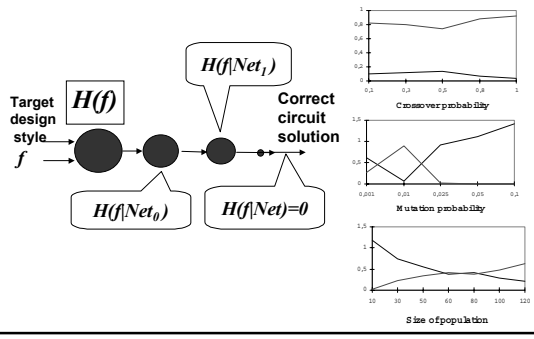


Benefit: massive parallelism

Idea !!!

If genetic information processing is *to extract valuable information* from *genetic information*, let us use Shannon information theory to measure evolutionary process of circuit design

Evolutionary circuit design and information streams



Entropy of fitness function

Definition. *Entropy based fitness function is the conditional entropy of a target function f given current function Net*

$H(f|Net) = - p_{00} \cdot \log_2 (p_{00} / p_0) - p_{10} \cdot \log_2 (p_{10} / p_0) - p_{01} \cdot \log_2 (p_{01} / p_1) - p_{11} \cdot \log_2 (p_{11} / p_1)$

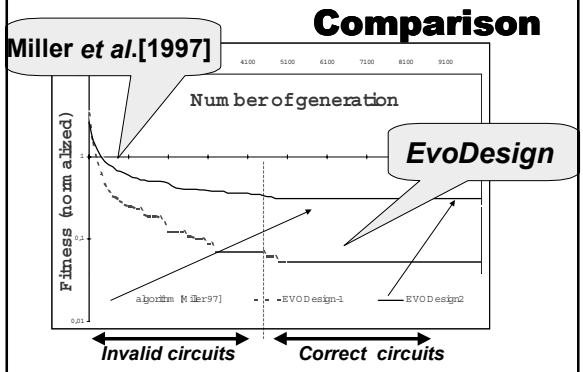
Example *2-digit ternary adder:*

0th generation $H(f|Net_0) = 1.585$;

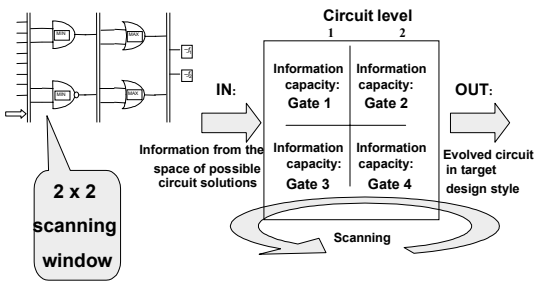
380th generation $H(f|Net_{380}) = 0.853$;

12610th generation $H(f|Net_{12610}) = 0$;

Comparison



Information capacity of scanning window



Concluding Remarks

- An extension of the evolutionary multi-level network synthesis due to parallel (and flexible) window-based scanning of the subspaces of possible network solutions with target design style
- Evolutionary network design becomes more attractive, if the concept of a *given design style* is realized.
  - In this case designer really becomes an expert and needs *no special software*.
- B-decomposition of the network search space is more preferable, because it *does not require any multiplexers* to merge a network from subnetworks.

Non solved problem

– *Can evolutionary computation be of practical interest for CAD Community?*

– *In which applications evolutionary computation can be an efficient support of traditional techniques?*

References

Dill, Karen M. Growing Digital Circuits: Logic Synthesis and Minimization with Genetic Operators. Master of Science Thesis. Department of Electrical and Computer Engineering, Oregon State University, June 1997.

Drechsler, Rolf. "Evolutionary Algorithms for Computer-Aided Design of Integrated Circuits Tutorial". Genetic Programming 1997 Conference, Stanford University, Sunday, July 13, 1997 -- 1:00-3:15 PM.

Goldberg, David. E. Genetic Algorithms in Search, Optimization, & Machine Learning. New York: Addison-Wesley Publishing Company, Inc. 1989.

Higuchi, Tetsuya. "Evolvable Hardware Tutorial". Genetic Programming 1997 Conference, Stanford University, Sunday, July 13, 1997 -- 9:15-11:30 AM.

Koza, John. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, Masachusetts: The MIT Press, 1992.

Koza, John. Genetic Programming II: Automatic Discovery of Reusable Programs. Cambridge, Massachusetts: The MIT Press, 1994.

Sipper, Moshe, Eduardo Sanchez, Daniel Mange, Marco Tomassini, Andres Perez-Uribe, and Andre Stauffer. "A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems". IEEE Transactions on Evolutionary Computation. Vol. 1, Number 1 (April 1997), pp. 83-97.

Sources

Karen M. Dill
James H. Herzog
Marek A. Perkowski

T.Luba*, S.Yanushkevich, M.Opoka, C.Moraga#, V.Shmerko

Warsaw University of Technology, Warsaw, Poland
#Dortmund University, Germany
Technical University, Szczecin, Poland