3-1999

Constructive Induction Machines for Data Mining

Marek Perkowski
Portland State University

Stanislaw Grygiel
Portland State University

Qihong Chen
Portland State University

Dave Mattson
Portland State University

CONSTRUCTIVE INDUCTION MACHINES FOR DATA MINING

Marek Perkowski, Stanislaw Grygiel, Qihong Chen, and Dave Mattson

Portland State University,
Department of Electrical and Computer Engineering, Portland, USA

EVOLVABLE HARDWARE OR LEARNING HARDWARE?

- Evolvable Hardware is Genetic Algorithm (GA) plus reconfigurable hardware.
- One may ask: "Why **Genetic** Algorithm"?
- We question the usefulness of GA as a sole learning method to reconfigure binary FPGAs.
- We propose the "Learning Hardware" approach.
- Creating a sequential network based on feedback from the environment (for instance, positive and negative examples from the trainer), and realizing this network in an array of Field Programmable Gate Arrays (FPGAs).

INDUCTION OF STATE MACHINES FROM TEMPORAL LOGIC CONSTRAINTS

1. Training on examples.
2. Constraints solving.
3. Finite State Machine (FSM) minimization.
4. Structural mapping of machine to Regular Automata.
5. Functional decomposition of multi-valued logic functions and relations to Regular Layout.
6. Final mapping of Regular Automata and Layout to FPGA resources.

LEARNING ON A HIGHER LEVEL

- Learning on the level of constraints acquisition and functional decomposition rather than on the low level of programming binary switches.
- **Occam's Razor** learning that allows for generalization and discovery.
- Fast operations on complex logic expressions and solving **NP-complete problems** such as **satisfiability**.
- Algorithms realized **in hardware** to obtain the necessary speed-ups.
- Fast prototyping tool, the **DEC-PERLE-1** board based on an array of Xilinx FPGAs.

SOFT COMPUTING AND MACHINE LEARNING VERSUS HARDWARE DESIGN

- Artificial Neural Nets (ANNs), Cellular Neural Nets (CNN), Fuzzy Logic, Rough Sets, Genetic Algorithms (GA), Genetic and Evolutionary Programming, Artificial Life, solving problems by analogy to Nature, decision making, knowledge acquisition, new approaches to intelligent robotics.
- Learning, adapting, modifying, evolving or emerging.
- Mixed approaches.
- The computer is taught on examples rather than completely programmed (instructed) what to do.
- **Machine Learning** becomes a new and most general system design paradigm unifying these previously disconnected research areas.
- It starts to become a **new hardware construction paradigm** as well.

EVOLVABLE HARDWARE

- DeGaris - Evolvable Hardware is realization of genetic algorithm (GA) in reconfigurable hardware.
- Brain Builder CBM (DeGaris), ROBOKONEKO.
- Neural Nets PLUS Genetic Algorithm.
- The Genetic Algorithm is a simple and practically blind mechanism of Nature.
- It is easily realizable in hardware.
- Although it is relatively easy to do crossover and mutation in hardware, the fitness function evaluation is difficult.

UNIVERSAL LOGIC MACHINE

- Started in Poland, 1977. Logic Design Machine. (TTL logic model): Satisfiability, Petrick Function (ICCAD'85).
- Tsutomu Sasao, 1985: Tautology Engine in EPLDs (ICCD'85).
- Cube Calculus Machine, since 1990. (realization in FPGAs). (Sendai'92).
- Decomposition Machine, since 1997, (DEC-PERLE-1), (Lousanne'98, ICCD'98, Sendai'99).
- Temporal Constraints Machine - new ideas presented here for the first time (reduce to Satisfiability, Tautology, Decision Functions, and Boolean/Multi-Valued Logic Equations.

LOGIC ALGORITHMS IN HARDWARE

- Logic algorithms draw upon human knowledge.
- Logic algorithms are optimal and mathematically sophisticated.
- Logic algorithms lead to high quality learning results: knowledge generalization, discovery, no overfitting, small learning errors (Ross, Abu Mostafa, DFC, COLT).
- Their software realizations use very complex data structures and controls.
- It is difficult to realize them in hardware.

LEARNING HARDWARE

- Learning understood very broadly, as any mechanism that leads to the improvement of operation.
- Evolution-based learning is thus included in it.
- Combinational or sequential network is constructed that stores the knowledge acquired in the learning phase.
- The learned network is next run on old or new data.
- The responses may be correct or erroneous. The network's behavior is then evaluated by some fitness (cost) functions and the learning and running phases are alternating.

TWO PHASES OF LEARNING IN HARDWARE

- **The phase of learning**, which is, constructing and tuning the network.
- **The phase of acting**. Using knowledge, running the network for data sets.
- Comparing to the process of developing and using a computer, the first stage could be compared to the entire process of conceptualizing, designing, and optimizing a computer, and the second stage to using this computer to perform calculations.
- You cannot redesign standard computer hardware, however, when it cannot solve the problem correctly.
- The Learning Hardware will redesign itself automatically using new learning examples given to it.

Logic rather than evolutionary methods for learning

- Michie makes distinction between black-box and knowledge-oriented concept learning systems by introducing concepts of **weak** and **strong** criteria.
- The system satisfies a **weak criterium** if it uses sample data to generate an updated basis for improved performance on subsequent data.
- A **strong criterion** is satisfied if the system communicates concepts learned in symbolic form.
- ANNs satisfy only the weak criterium while our approach satisfies the strong criterium. Our approach operates on higher and more natural symbolic representation levels.

Logic rather than evolutionary methods for learning. II

- The built-in mathematical optimization techniques (such as graph coloring or satisfiability) support the Occam's Razor Principle.
- Solutions are provably good in the sense of Computational Learning Theory (COLT).

Importance of Functional Decomposition

- **Functional Decomposition** is used in many applications: FPGA mapping, custom VLSI design, regular arrays, Machine Learning, Data Mining and Knowledge Discovery in Data Bases (KDD).
- **Exact** decomposition programs are slow.
- **Approximate** programs may give inferior quality solutions.
- How to create a decomposer that will be **both effective** and **efficient**?.
- ANSWER: Software/Hardware Co-Design.

We do not like Genetic Algorithms. Any Discussions?

- In our experience, especially poor results on logic approaches are obtained using the genetic algorithms.
- The same was true based on literature.
- In our approach we want to make use of this accumulated human experience, rather than to "reinvent" algorithms using GA.

The ULM approach to Learning Hardware

1. Based on sets of examples specified in our input language **L**, we create a Reactive State Machine (RSM), in particular, a (combinational) function or a relation with no temporal variables.
2. The description consists in input-output specification, initial state specifications and global environment constraints.
3. This machine is usually non-deterministic, but is state-minimal from construction with respect to all its variables as state variables.
4. The machine can be determinized (converted from non-deterministic to deterministic form).
5. Next the machine is state-minimized (with respect to the new set of state variables, which are a subset of initial input/output variables).
6. The machine is mapped to constrained structural resources which we call Regular Automata (RA).

7. The time-based MV logic expressions of Regular Automata are decomposed. (Ashenhurst-Curtis decomposition). The timed variables, and the multi-valued variables, are converted to new binary variables.
8. The (quasi)optimally constructed network is logically mapped to standard FPGA CLBs and realized using standard partitioning, placement, and routing with the help of EDA tools from Xilinx or other companies. Thus, each RSM is converted to a binary pattern of programming switches in FPGA.
9. The knowledge of the machine is stored in binary memory patterns representing final FPGA reconfigurable information. Under supervision of the software program, the hardware switches between a number of evolved circuits, depending on rules that can also be acquired automatically.
10. As the network solves new problems, the new data sets and training decisions are accumulated and the network is repeatedly redesigned.

Induction of Reactive State Machines from Temporal Logic Constraints

- Our state machine design integrates methods developed in USA and former USSR.
- The use of temporal logic as the input specification.
- The use of Regular Automata for structural design.

The Input Language to Represent the Learning Data

- High-level constraint temporal language.
- temporal logic (Chebotarev), regular expressions (Glushkov,Kleene), Petri nets, State Machine tables, tabular representation of data (Codd), binary and Multi-Valued Cube Calculus (Dietmeyer), Decision Tables, Rough Sets (Pawlak), and Labeled Rough Partitions (Grygiel).

Small Example

	x_1	x_2	y_1	y_2
a	0,2	1	-	2
b	0,1	0	0,2	1
c	2	0	1,2	0
d	1	1	1,2	2

Table 1: Multi-Valued multi-output (combinational) relation in tabular form.

Now we add time

Introducing variables that depend on discrete time.

For instance, the example a (row a) from Table 1 can be rewritten to our language as follows:

$$x_1[0,2](t) \ \wedge \ x_2[1](t) \ \Rightarrow y_2[2](t)$$

because both input variables $x_1(t)$, $x_2(t)$ and the output variable $y_2(t)$ are defined in the same moment of time. By allowing previous or next ticks of time, for instance:

$$x_1[0,2](t-2) \ \wedge \ x_2[1](t-3) \ \Rightarrow y_2[2](t+3)$$

we can specify arbitrary regular grammars, regular expressions, sequential netlists, or state machines with multi-valued inputs and outputs.

EXAMPLE: "Man, Wolf, Goat and Cabbage"

- At the beginning, all of them are on the left bank of the river.
- The man should transport them to the right bank.
- The wolf and the goat, as well as the goat and the cabbage, cannot be left alone on the same bank without man.
- The boat is navigated by the man, who can take only one object with him.
- Boolean variables are first-order predicates depending on discrete time.
- $M(t)$ is true when the man in the left bank and false when the man in on the right bank.
- The same applies to variables $W(t), G(t), C(t)$, which denote the wolf, the goat, and the cabbage respectively.

Temporal logic specification of constraints for "Wolf" (1)

1) The wolf and the goat cannot stay on the left bank and on the right bank without the man:

$(W(t)\&G(t)) \Rightarrow M(t)$. $(\sim W(t)\& \sim G(t)) \Rightarrow \sim M(t)$.

The goat and the cabbage cannot stay on the left bank and on the right bank without the man:

$(G(t)\&C(t)) \Rightarrow M(t)$. $(\sim G(t)\& \sim C(t)) \Rightarrow \sim M(t)$.

2) If the wolf is on the left (right) bank, it means that either the wolf stayed there or the man has brought it there from the right (left) bank one unit of time before:

$W(t) \Rightarrow (W(t-1) \mid \sim W(t-1)\& \sim M(t-1)\&M(t))$. $\sim W(t) \Rightarrow (\sim W(t-1) \mid W(t-1)\&M(t-1)\& \sim M(t))$.

Temporal logic specification of constraints for "Wolf" (2)

The same for the goat and the cabbage:

$(G(t) \Rightarrow (G(t-1) \mid \sim G(t-1) \& \sim M(t-1) \& M(t)).$ $\sim G(t) \Rightarrow (\sim G(t-1) \mid G(t-1) \& M(t-1) \& \sim M(t)).$

$(C(t) \Rightarrow (C(t-1) \mid \sim C(t-1) \& \sim M(t-1) \& M(t)).$ $\sim C(t) \Rightarrow (\sim C(t-1) \mid C(t-1) \& M(t-1) \& \sim M(t)\ ).$

3) Any two objects cannot be brought across the river at the same time:

$(W(t) \Leftrightarrow W(t-1)) \mid (G(t) \Leftrightarrow G(t-1)).$

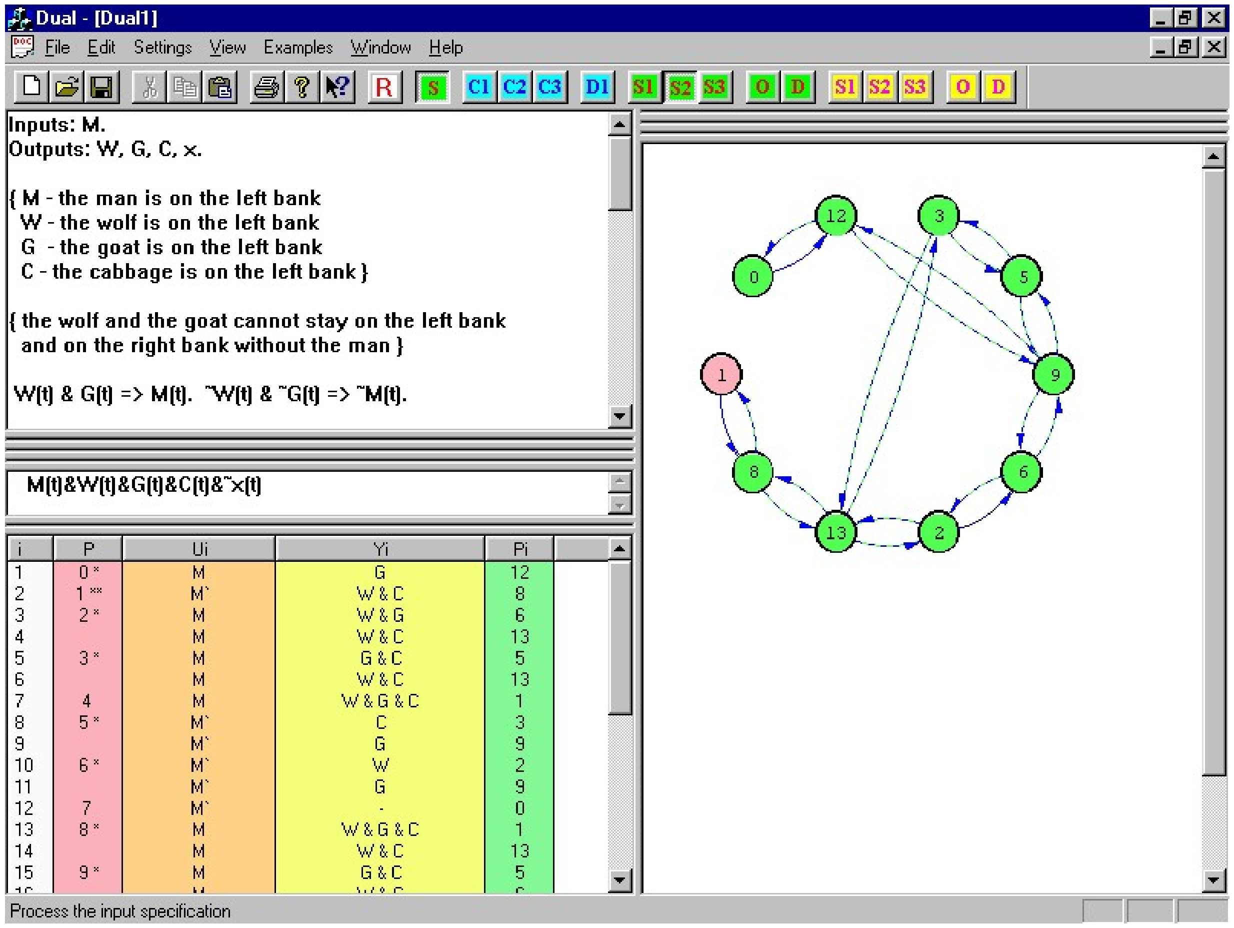
$(W(t) \Leftrightarrow W(t-1)) \mid (C(t) \Leftrightarrow C(t-1)).$

$(G(t) \Leftrightarrow G(t-1)) \mid (C(t) \Leftrightarrow C(t-1)).$

4) The man is not lazy:

$M(t-1) \Leftrightarrow \sim M(t).$

Reactive State Machine for "Wolf"


Dual - [Dual1]
File Edit Settings View Examples Window Help
R S C1 C2 C3 D1 S1 S2 S3 O D S1 S2 S3 O D
0
1
2
3
5
6
8
9
12
13

```
Inputs: M.
Outputs: W, G, C, x.

{ M - the man is on the left bank
  W - the wolf is on the left bank
  G  - the goat is on the left bank
  C - the cabbage is on the left bank }

{ the wolf and the goat cannot stay on the left bank
  and on the right bank without the man }

 W(t) & G(t) => M(t).  ~W(t) & ~G(t) => ~M(t).
```

```
M(t)&W(t)&G(t)&C(t)&~x(t)
```

i	P	Ui	Yi	Pi
1	0 *	M	G	12
2	1 **	M`	W & C	8
3	2 *	M	W & G	6
4		M	W & C	13
5	3 *	M	G & C	5
6		M	W & C	13
7	4	M	W & G & C	1
8	5 *	M`	C	3
9		M`	G	9
10	6 *	M`	W	2
11		M`	G	9
12	7	M`	-	0
13	8 *	M	W & G & C	1
14		M	W & C	13
15	9 *	M	G & C	5

Process the input specification

Figure 1: DUAL interface for RSM acquisition and synthesis ("man, wolf, goat and cabbage" problem).

Reactive State Machine for "Wolf"

- The state transition graph given in dotted lines corresponds to the **non-deterministic algorithm**, which the man should follow to safely transport all the objects to the right bank.
- This algorithm consists of seven steps, with two possible variations, depending on whether he chooses to take the cabbage or the wolf when the goat is already trasported to the right bank.
- Observe that the designer who gives the language **L** expressions to the learning system, does not know the state machine.
- He only specifies **constraints** on the behavior of the system or sample sequences of input and output values.

DEC-PERLE-1 Board for Fast Prototyping

- The DEC-PERLE-1 board (Vuillemin) is organized around a central computational matrix made up of 16 Xilinx XC3090 LCAs surrounded by a four 1MB RAM banks, and 7 other LCAs to implement switching and controlling functions.
- Cube Calculus Machine (CCM) on DEC-PERLE.
- The DEC-PERLE-1 board advocates **regular design styles** without many control signals.
- Good for small SIMD processors, pipelining, systolic processors, Cellular Automata or complex (decomposed) Boolean functions.
- The basic design principle is: **“map two-dimensional tables to two-dimensional logic resource arrays”**.
- Leads to the introduction of the concept of **Regular Automata**.

Demonstration of Learning Hardware in Robotics

- **"An Improved Furby".**
- Humanoid arms/head robot.
- OWI, Robix and LEGO MINDSTORMS robotic kits.
- Sensors (touch, light, sound, temperature).
- Speech recognition.
- Image Recognition.

Demonstration of Learning Hardware in Robotics

- Learning is done with the human as the feedback loop.
- The set of sequences is incomplete, so the machine performs the **generalization** automatically. Adding or removing new rules, by the human supervisor or automatically/randomly, will change the behavior.
- Mimique the human's behaviors seen by the camera
- Like the Furby toy, but with real learning.
- Capable of building its own "world model" and internal model with unlimited behaviors.
- The new states corresponding to changed behavior will be created using our RSM approach.

Brain Builder (De Garis) versus Universal Logic Machine.

\ Approach to Aspect of \ "Learning Hardware" Comparison.	Brain Builder	Universal Logic Machine
Model of Learning	Artificial Neural Net	Reactive State Machine
Training Data	Sample Vectors	Multi-Valued Temporal Logic Constraint Language **L**
How the net is constructed	Genetic Algorithm, ANN Training	Reactive State Machines construction and minimization, Multi-Valued Logic Synthesis
Virtual intermediate representation	Cellular Automata (CA)	Regular Automata (RA)
What is learned:	CA tables	Binary sequential logic nets
Net construction realized in:	Hardware (intrinsic EHW)	Hardware and software (software-hardware codesign)
Mapped to:	Array of binary FPGAs	Array of binary FPGAs
Hardware platform:	Xilinx 6000 series CBM	Xilinx 3090 + on-board memory DEC-PERLE-1 board + DEC workstation

WHAT IS EVOLVABLE HARDWARE

- This talk reviews the in the domain of EHW in years 1989 - 1999 and points out some fundamental open research issues.
- What Is Evolvable Hardware (EHW)
- EHW as an Alternative to Electronic Circuit Design
- EHW as an Adaptive System
- Other EHW-Related Work
- Evolvable Hardware versus Learning Hardware
- Learning Multi-Valued Functions
- Universal Logic Machine - Current PSU approach to Learning Hardware
- Our Proposed Extensions: Learning Finite State Machines.
- Concluding Remarks

WHAT IS EVOLVABLE HARDWARE (cont)

- There are different views on what EHW is, depending on the purpose of EHW.
- EHW can be regarded as "applications of evolutionary techniques to circuit synthesis." (A. Hirst)
- EHW is hardware which is capable of on-line adaptation through reconfiguring its architecture dynamically and autonomously. (T. Higuchi et al.).
- EHW is Genetic Algorithm realized in hardware (DeGaris). (Intrinsic Evolvable Hardware).

LEARNING IS MORE GENERAL THAN EVOLVING

- Learning is more general than evolving.
- Evolving is learning by Nature: blind, random, chaotic.
- Learning is any kind of behavior that improves something.
- Learning Hardware is any kind of hardware system that can change itself and its future behavior dynamically and autonomously by interacting with its environment.
- EHW is a child of the marriage between evolutionary computation techniques and electronic hardware.
- LH is a child of the marriage of Machine Learning and hardware (so far, electronic, but see Hanyu et al for DNA and molecular computing).

EHW AS AN ALTERNATIVE TO ELECTRONIC CIRCUIT DESIGN

- Using EAs to design VLSI chips and boards has a 12 year long history.
- Used in Digital and Analog design; (mixed?).
- Few examples:
  - Evolving Hardware Description Language (HDL) programs.
  - Evolving Electronically Programmable Logic Devices (EPLDs).
  - Evolving analog circuits.
  - Unconstrained evolution of an electronic oscillator (Adrian Thompson).
  - Generalized Reed-Muller Logic using GA (Karen Dill).
  - Arbitrary Tree logic networks using GP (Karen Dill).

TWO MAJOR APPROACHES

- Early and some of the recent work related to EHW only dealt with **optimisation** of VLSI circuits, such as cell placement, logic minimisation and compaction of symbolic layout.
- Circuit **functions** were not designed/evolved by EAs.
- Recent work concentrates on evolving circuit architectures and thus functions. Two major approaches have been used:
  - Indirect Approach,
  - Direct Approach.

INDIRECT APPROACH TO EHW CIRCUIT DESIGN

• The indirect approach does not evolve hardware directly, but evolves an intermediate representation (such as trees) which specifies hardware circuits.

• **Evolving digital circuits.**

For example, SFL (Structured Function Description Language) programs (represented by production trees) can be evolved by a genetic algorithm. A binary adder which considers all 4-bit numbers was evolved successfully.

• **Evolving analog circuits.**

For example, Koza's work on evolving a lowpass "brick wall" filter, an asymmetric bandpass filter, an amplifier, etc. Trees were used to represent circuits. The results were competitive with human designs.
href="http://www-cs-faculty.stanford.edu/ koza/#anchor5384423"

DIRECT APPROACH TO EHW CIRCUIT DESIGN - GATE LEVEL

• The direct approach evolves hardware circuit's architecture bits directly. It works well only with reconfigurable hardware, such as FPGA (field programmable gate array) from **"http://www.xilinx.com/"** (Xilinx).

• The gate level evolution implies that the "atomic" hardware functional units are logical gates like **AND**, **OR**, and **NOT**. The evolution is used to search for different combinations of these gates.

• Typical examples include XOR, counters, FSMs (Finite State Machines), multiplexors, and an electronic oscillator.

• One argument for the direct approach is to exploit hardware resources by **unconstrained** hardware evolution.

DIRECT APPROACH TO EHW CIRCUIT DESIGN - FUNCTIONAL LEVEL

- The gate level evolution runs into the scalability problem quickly.
- The function level evolution uses high-level functions such as addition, multiplication, sine, cosine, etc., and thus is much more powerful.
- Typical examples: two-spiral, Iris, FSMs, image rotation.
- The work is better viewed as an attempt towards **adaptive hardware**, rather than as a design alternative.

ADVANTAGES OF EVOLUTIONARY DESIGN

- Explores a larger design space and thus may be able to discover novel designs.
- Does not assume a priori knowledge and thus can be applied to various domains.
- Does not require exact specification and thus can design complex systems which cannot be handled by conventional specification-based design approach.
- However, constraints and special requirements could be imposed on the evolution if necessary through the fitness function and chromosome representation.
- Some analog circuits might be too difficult (or costly) to design by human experts.

SCALABILITY OF EHW

- Scalability of the algorithm: Time complexity of the EA for EHW?
- Scalability of the representation: Size of chromosomes vs. Size of EHW?
- Time is more crucial since the size of chromosome (space) is usually polynomial in the size of EHW circuits.

WILL ELECTRONIC SPEED SOLVE THE SCALABILITY PROBLEM?

• There have been some expectations that the speed of simulated evolution would not be a problem in a few years as faster VLSI chips come out.

• This statement can be misleading. Electronic speed is **not** a solution to the scalability problem. The scalability problem has to be addressed at the fundamental level.

• The importance of the time complexity issue can be illustrated by an artificial example. If the time complexity of simulated evolution is $O(2^n)$, where n is the size of EHW, then an EHW with 10 components would need $2^{10} = 1024$ nanoseconds ($\approx 10^{-6}$ seconds) to evolve. An EHW with 100 components would need $2^{100} \approx 10^{30}$ nanoseconds (10^{13} years).

CIRCUIT VERIFICATION/TEST AND FITNESS FUNCTION

• How to verify the correctness of EHW? How to find a fitness function which guarantee the correctness of EHW?

• For example, if all 4-bit numbers have been correctly added, would all 5-bit, 6-bit, etc., numbers be added correctly by the same circuit?

• Exploiting hardware resources is attractive. Has an EHW exploit something totally irrelevant, such as room temperature or minor Earth movement?

• Is it practical to test all possible situations in which an EHW might be used?

• How robust is EHW to minor environmental changes? Does it degrade gracefully?

• When to stop simulated evolution? How to know whether a correct circuit has been evolved?

EHW AS AN ADAPTIVE SYSTEM

- Current work on **adaptive EHW** can be classified into two major categories:
- EHW controllers.
- EHW recognisers and classifiers.

EHW CONTROLLERS

• A number of control tasks can be performed by EHW, e.g., ATM control and robot control among others.

Some examples:

- Evolving an artificial ant to follow the John Muir Trail in simulation.
- Evolving a wall following robot in a simulated environment, “virtual reality”.
  ”http://www.cogs.susx.ac.uk/users/adrianth/”.
- Evolving an ATM traffic shaper.
- Evolving an adaptive equaliser.

EHW RECOGNIZERS AND CLASSIFIERS

- Evolving FPGA to perform learning tasks, such as letter recognition, the comparator in a V-shape ditch tracer, two-spiral, Iris, FSMs, etc.
- Unlike most other studies, generalisation is explicitly emphasised here.
- A complexity (regularisation) term was included in the fitness evaluation function.

OTHER EHW-RELATED WORK

- Self-reproduction and self-repair hardware at Logic Systems Laboratory (LSL), Computer Science Department, Swiss Federal Institute of Technology – Lausanne. **http://lslsun5.epfl.ch/"**.
- Artificial brains.
  - CAM-BRAIN (CBM) from ATR's Department 6 (Evolutionary Systems) **"http://www.hip.atr.co.jp/ flx/ATRCAM8"**.
  - Artificial Brain Systems at RIKEN. (No hardware implementation.) **"http://www.bip.riken.go.jp/absl/Welcome.html"**.

SOME CHALLENGES TO ADAPTIVE EHW

- Scalability: Efficiency of simulated evolution.
- Generalisation: Dealing with new environments.
- Disaster prevention in fitness evaluation during on-line adaptation.
- On-line adaptation: incremental evolution/learning.

A BEHAVIORAL VIEW TOWARDS EHW

• What is being evolved? A circuit or the circuit's behaviours? In other words, what is actually being evaluated by a fitness function?

• Is it genetic evolution or behavioural evolution?

• **Claim:** It is EHW behaviour, not its circuitry, that is being evolved. Some consequences of taking the behavioural view towards EHW:

1. The environment is crucial. Generalisation should be discussed with respect to environments.

2. The role of crossover needs to be re-evaluated.

CONCLUDING REMARKS

• Population-based learning (simulated evolution) is good at following slow environmental changes, but not at real-time on-line adaptation. Individual learning should be introduced.

• There is some existing work on EANNs and GP which may be useful for function-level EHW, e.g., mutations and other techniques for maintaining behavioural links between parents and their offspring.

• Co-evolution is a very promising approach to deal with the problem of fitness evaluation. That is, co-evolution can be used to generate changing and challenging environments.

FURTHER REMARKS

- Evolutionary design of digital circuits would not be able to compete with the conventional approach.
- Evolutionary design of analog circuits needs to address the issues of circuit verification and robustness.
- Adaptive EHW has most potentials, but would need individual learning to implement on-line learning.
- The most profitable application domains for EHW would be those which are very complex but highly specialised.

WWW RESOURCES

- The following papers are available on-line.

X. Yao and T. Higuchi, "Promises and Challenges of Evolvable Hardware," Submitted to ICES'96. (Available as **"ftp://www.cs.adfa.oz.au/pub/xin/ices96_challenge.ps.gz"**.

DATA MINING BY CONSTRUCTIVE INDUCTION MACHINES

• "Learning Hardware" approach involves creating a computational network based on feedback from the environment and realizing this network in an array of Field Programmable Gate Arrays (FPGAs).

• Feedback, is for instance by positive and negative examples from the trainer.

• Environment can be the trainer.

• Computational networks can be built based on incremental supervised learning (Neural Net training) or global construction (Decision Tree design).

• Here we advocate the approach to Learning Hardware based on Constructive Induction methods of Machine Learning (ML) using multi-valued functions.

• This is contrasted with the Evolvable Hardware (EHW) approach in which learning/evolution is based on the genetic algorithm **only**.

WHY TO USE HARDWARE INSTEAD OF SOFTWARE?

- Supervised inductive learning algorithms require fast operations on complex logic expressions and solving some NP-complete problems.
- Satisfiability, Tautology, Solving Boolean Equations, Graph Coloring, Set Covering, Maximum Cliques.
- These algorithms should be realized in hardware to obtain the necessary speed-ups.
- Fast prototyping tool, DEC-PERLE-1 board is based on an array of Xilinx FPGAs.
- We are developing virtual processors that accelerate the design and optimization of decomposed networks of **arbitrary** logic blocks.

EVOLVING IN HARDWARE VERSUS LEARNING IN HARDWARE

• Soft Computing: Artificial Neural Nets (ANNs), Cellular Neural Nets (CNN), Fuzzy Logic, Rough Sets, Genetic Algorithms (GA), Genetic and Evolutionary Programming, Artificial Life, Solving Problems by Analogy to Nature, decision making, knowledge acquisition, new approaches to intelligent robotics (Brooks).

• Learning, adapting, modifying, evolving or emerging.

• Mixed approaches combine elements of these areas with the goal of solving very complex and poorly defined problems that could not be tackled by previous, analytic models.

EVOLVING IN HARDWARE VERSUS LEARNING IN HARDWARE (cont)

• What is common to all these approaches is that they propose a way of **automatic learning** by the system.

• The computer is taught on examples rather completely programmed (instructed) what to do.

• **Machine Learning** (ML) becomes then now a new and most general system design paradigm unifying many previously disconnected research areas.

• ML starts to become a **new hardware construction paradigm** as well.

EVOLVABLE HARDWARE VERSUS LOGIC METHODS.

• **Evolvable Hardware** (EHW) (De Garis, Higuchi) is a realization of genetic algorithm (GA) in reconfigurable hardware.

• Our approach of Universal Logic Machine (ICCAD '85, Sendai '92, Jozwiak'98), proposes to build a learning machine based on **logic principles**.

• Constructive Induction (Michalski) and Rough Set Theory (Pawlak).

• Genetic Algorithm is a very simple and practically blind mechanism of Nature, it can be easily realizable in hardware.

• We do not believe that this mechanism alone cannot produce good results.

EVOLVABLE HARDWARE VERSUS LOGIC METHODS. TRADE-OFFS

- The logic algorithms that use previous human knowledge are optimal and mathematically sophisticated. They lead to high quality learning results.
- Their software realizations use so complex data structures and controls that it is very difficult to realize them in hardware.
- Software/hardware realizations may suffer from the consequences of the Amdahl's Law.
- Interesting software-hardware design trade-offs must be resolved to realize optimally the learning algorithms based on logic.

LEARNING HARDWARE

• "Learning Hardware" is any mechanism that leads to the improvement of operation, evolution-based learning is thus included.

• The process of learning some kind of network. It stores the knowledge acquired in the learning phase (the network can become equivalent to a state machine or fuzzy automaton by adding some discrete or continuous memory elements).

• The learned network is next run (executed, evaluated, etc.) for old or new data given to it, thus producing its responses - expected behaviors(decisions, controls) in unfamiliar situations (new data sets).

• The responses may be correct or erroneous, the network's behavior is then evaluated by some fitness (cost) functions and the learning and running phases are interspersed.

LEARNING HARDWARE: TWO PHASES.

• The process of solving problems is thus always reduced to **two phases**: **the phase of learning**, which is, constructing and tuning the network, and **the phase of using knowledge**, that is, running the network for data sets.

• The first stage could be compared to the entire process of computer design, and the second stage to using this computer to perform calculations.

• You cannot redesign the standard computer hardware. The Learning Hardware will redesign itself automatically based on new learning examples given to it.