PLAN OF EVOLVABLE AND LEARNING HARDWARE LECTURES

- **Our hardware**: the DEC-PERLE-1 board.
  - Programming/designing environment for DEC-PERLE/XILINX.
  - Two different concepts of designing Learning Hardware using the DEC-PERLE-1 board.

- **Compare** logic **versus** ANN and GA approaches to learning.

- **Introduce the concept of** Learning Hardware

- **Methods of** knowledge representation **in the** Universal Logic Machine (ULM):
  - variants of Cube Calculus.

- A general-purpose computer with instructions specialized to operate on logic data: Cube Calculus Machine.
  - Variants of cube calculus - arithmetics for combinatorial problems
  - Our approach to Cube Calculus Machine

- **A processor for only one application**: Curtis Decomposition Machine.
Function Decomposition is at least an NP-hard problem.

Most its stages are NP-hard problems.

One approach to find solutions to NP-hard problem is not to attempt at the exact solution, but be satisfied with one which is near exact but obtainable in a reasonable time.

This type of algorithm is based on heuristics, or rules which can be applied which are likely to improve the solution.

Such algorithms, when implemented in hardware, can bring orders of magnitude speed-up.

We have chosen algorithms that are simple, easy, fast and can be relatively easy implemented in hardware.

We showed that decomposition of fuzzy functions and relations can be reduced to decomposition of multi-valued functions and relations.
LEARNING BY FUNCTIONAL DECOMPOSITION MACHINE

- Design philosophy of the FPGA implementation of a point algorithm.
- Phases of the algorithm are executed sequentially, they are then loaded from the host memory, while the intermediate data are stored in DEC-PERLE-1 memories between stages.
- We will show also how generic combinatorial problems are used in logic learning algorithms.
- The ideas of graph coloring will be used for decomposing functions, and thus in Machine Learning.
The decision table represents a data set, with labeled instances, each relating a set of attribute values to a class (the output concept).
Decomposition of the table is to decompose the initial table into a hierarchy of decision tables, each of them no more decomposable.
Thus, each of these new tables, as well as the entire network are less complex and easier to interpret than the original table.
Some regularities not seen in the original table can be found, and the intermediate functions correspond to some features (concepts) of the data set.
Ashenhurst/Curtis
Decomposition has been adopted to multiple-valued logic (ISMVL'97).
It applies iteratively the single decomposition step, whose goal is to decompose a function $y = F(X)$ into $y = G(A,H(B))$, where $X$ is a set of input attributes $x_1,x_2,...,x_n$, and $y$ is the class.
F, G, and H are functions represented as decision tables, i.e. possibly incomplete sets of attribute-value vectors with assigned classes.
A and B are subsets of input attributes, called free and bound set, respectively, such that \( A \cup B = X \).
Functions G and H are developed in the decomposition process and not predefined in any way.
New concept $c_1 = H(B)$ has been found.

The goal is to find the decomposition of the smallest complexity (DFC - Abu-Mostafa).
Example of Decomposition

Three possible non-trivial partitions of attributes that yield three different decompositions

\[ y = G_1(x_1, H_1(x_2, x_3)), \quad y = G_2(x_2, H_2(x_1, x_3)), \]
\[ y = G_3(x_3, H_3(x_1, x_2)). \]

The comparison shows that:

1. decision tables in the decomposition \( y = G_1(x_1, H_1(x_2, x_3)) \) are smaller than those for \( y = G_2(x_2, H_2(x_1, x_3)) \),
2. the new concept \( c_i = H_1(x_2, x_3) \) uses only three values, whereas that for \( H_2(x_1, x_3) \) uses four,
3. we found it hard to interpret decision tables \( G_2 \) and \( H_2 \), whereas by inspecting \( H_1 \) and \( G_1 \) it can be easy to see that \( c_1 = \text{MIN}(x_2, x_3) \) and \( y = \text{MAX}(x_1, c_1) \).
4. This can be even more evident with the assignment of values 0,1, and 2 of a multi-valued variable \( X_i \): \( X_i^0 = \text{lo}, X_i^1 = \text{me}, X_i^2 = \text{hi} \).
An example decision table $y = F(x_1, x_2, x_3)$

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>lo</td>
<td>lo</td>
<td>lo</td>
<td>lo</td>
</tr>
<tr>
<td>lo</td>
<td>lo</td>
<td>hi</td>
<td>lo</td>
</tr>
<tr>
<td>lo</td>
<td>me</td>
<td>lo</td>
<td>lo</td>
</tr>
<tr>
<td>lo</td>
<td>me</td>
<td>hi</td>
<td>me</td>
</tr>
<tr>
<td>lo</td>
<td>hi</td>
<td>lo</td>
<td>lo</td>
</tr>
<tr>
<td>lo</td>
<td>hi</td>
<td>hi</td>
<td>hi</td>
</tr>
<tr>
<td>lo</td>
<td>hi</td>
<td>hi</td>
<td>hi</td>
</tr>
<tr>
<td>me</td>
<td>lo</td>
<td>lo</td>
<td>me</td>
</tr>
<tr>
<td>me</td>
<td>lo</td>
<td>hi</td>
<td>me</td>
</tr>
<tr>
<td>me</td>
<td>me</td>
<td>lo</td>
<td>me</td>
</tr>
<tr>
<td>me</td>
<td>me</td>
<td>hi</td>
<td>me</td>
</tr>
<tr>
<td>me</td>
<td>hi</td>
<td>lo</td>
<td>me</td>
</tr>
<tr>
<td>me</td>
<td>hi</td>
<td>hi</td>
<td>hi</td>
</tr>
<tr>
<td>hi</td>
<td>lo</td>
<td>lo</td>
<td>hi</td>
</tr>
<tr>
<td>hi</td>
<td>lo</td>
<td>hi</td>
<td>hi</td>
</tr>
<tr>
<td>hi</td>
<td>me</td>
<td>lo</td>
<td>hi</td>
</tr>
<tr>
<td>hi</td>
<td>me</td>
<td>hi</td>
<td>hi</td>
</tr>
<tr>
<td>hi</td>
<td>hi</td>
<td>lo</td>
<td>hi</td>
</tr>
<tr>
<td>hi</td>
<td>hi</td>
<td>hi</td>
<td>hi</td>
</tr>
</tbody>
</table>
Two one-step decompositions

(a) $x_1, x_2, x_3, y$

(b) $x_1, x_2, x_3, y$

(c) $x_1, x_2, x_3, y$

(d) $x_1, x_2, x_3, y$

(e) $x_1, x_2, x_3, y$

(f) $x_2, c_2, y$
DECOMPOSITION (cont)

• The following problems must be solved by an efficient decomposition algorithm:

1. how to select sets A and B?

2. how to evaluate the quality of decompositions?

All known methods require nearly exhaustive searches that involve huge repetitions of basic operations.
The assignment of values of \( c \) is trivial in case of a completely specified function, which is, when decision table instances completely cover the attribute space.

Otherwise, when the function is incompletely specified, the relation of compatibility of columns is no longer transitive, and the **graph coloring approach** is used.

Column functions are calculated by a cofactor operation on the original function \( f \).

The **cofactor** \( f\_\{\text{PROD}\} \) of function \( f \) with respect to the literals from \( \text{PROD} \) is this function with all literals from \( \text{PROD} \) substituted to maximum constant value (constant value 1 in case of binary logic).

All functions are represented by arrays of cubes.
For a completely specified binary function, two columns \( n_1 \) and \( n_2 \) are compatible if the Boolean functions corresponding to them are a Boolean Tautology:

\[
\text{n}_1 \text{ compatible } \text{n}_2 \text{ iff } f_\{\text{Prod}_1\} = f_\{\text{Prod}_2\}
\]

which is equivalent to:

\[
\text{n}_1 \text{ compatible } \text{n}_2 \text{ iff } (\text{ON}(n_1) \# \text{ON}(n_2) = \text{emptyset})
\]

and \((\text{ON}(n_2) \# \text{ON}(n_1) = \text{emptyset})\)

where \# denotes the sharp (difference) operation on arrays of cubes, and \text{ON} is the set of true cubes in SOP form.
For an incompletely specified binary function, two nodes of the graph for coloring are incompatible if the corresponding columns are not compatible (cannot be merged into one column):

\[ n_1 \text{ incompatible } n_2 \text{ iff } ( \text{ON}(n_1) \land \text{OFF}(n_2) \neq \emptyset \text{ or } (\text{ON}(n_2) \land \text{OFF}(n_1) \neq \emptyset \text{ or emptyset} ) \]
DECOMPOSITION IN HARDWARE

- Only two basic operations, **cofactor** and **sharp** are used for complete functions.
- Only cofactor and **intersection** are used for incomplete functions.
- In both cases, these operations are repeated many times on cubes from the cube arrays.
- Basic (mv) logic operators used for checking compatibility of columns of multiple-valued functions while creating the graph for coloring.
After creation, the graph is colored in such a way that every two nodes linked by an edge obtain different colors, and the minimum number of colors is used.

Graph coloring can be reduced to sequences of basic logic operators.

Concluding, in addition to cofactoring, the partial combinatorial problems that are solved by our hardware decomposition processor DP are:

- set covering,
- graph coloring,
- maximum clique.

They are all NP-hard, and they all have many other applications in ML.
OTHER VIRTUAL PROCESSORS

- **Rough Set Machine (RSM).**
  - A SIMD processor that realizes the basic operations of Rough Sets theory of Zdzislaw Pawlak.

- **Satisfiability Machine (SM).**
  - A systolic processor to solve satisfiability and related problems that occur in many combinatorial optimization problems.
CONCLUSIONS

- Principles of Learning Hardware as a competing approach to Evolvable Hardware, and also as its generalization.
- Data Mining machines.
- Universal Logic Machine with several virtual processors.
- DEC-PERLE-1 is a good medium to prototype such machines, its XC3090A chip is now obsolete.
- This can be much improved by using XC4085XL FPGA and redesigning the board.
- Massively parallel architectures such as CBM based on new Xilinx series 6000 chips will allow even higher speedups.


PSU POLO Directory with DM/ML Benchmarks, software and papers: http://www.ee.pdx.edu/polo/


P. Sapiecha, M. A. Perkowski, and T. Luba, "Decomposition of Information Systems Based on Graph Coloring Heuristics," Symposium on Modelling, Analysis and Simulation, CESA'96 IMACS Multiconference, Lille, France, July 9-12, 1996.


U.C. Irvine, ``Repository of Machine Learning Databases and Domain Theories,''

J. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. Touati, 