Machine Learning Meets Signal Processing Algorithm Optimization

---

A.K.A.
"Features? What are Features?"
"Cooley-Tukey? Huh?"

Bryan Singer

Joint work with Manuela Veloso and the SPIRAL group

ICML 2000:

- Overview
- Signal Processing Details
- Feature Descriptions
- Feature Evaluations
- Learning
- Conclusions

Newer Stuff:

- No Leaf 1's
- José's Algorithm
- Genetic Algorithm

The SPIRAL Group in ECE

Motto: Faster, Faster, Faster!

Goal: "To automate the implementation and optimization of signal processing algorithms on different computer architectures (uniprocessors, multiprocessors, special purpose hardware)."

- Find the *fastest* implementations of common signal processing algorithms (e.g., FFT, WHT)
- Do this for a wide variety of computer architectures
- Do it automatically

Along the way, they would like to try to **understand** what influences performance

Does Machine Learning have anything to offer toward these goals???

Signal Processing

- A signal processing algorithm
  - ▷ takes as input a signal (vector)
  - ▷ produces a transformation of the signal (vector)
- Most algorithms can be represented by a matrix which when multiplied by the input vector produces the desired output vector
- For large data sets, this matrix multiplication is very slow
- Structure in the transformation matrix allows it to be factored
- These factorization allow for much faster implementations
- We want to find the optimal factorization

This should be simple, right?...

WHT: Scary Formulas

to make you think its hard

The Walsh-Hadamard Transform of a signal x of size 2^n is the product $WHT(2^n) \cdot x$ where

$$WHT(2^n) = \bigotimes_{i=1}^{n} DFT(2),$$

$$DFT(2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

and $\otimes$ is the tensor or Kronecker product.

If A is a $m \times m$ matrix and B a $n \times n$ matrix, then $A \otimes B$ is the block matrix product

$$\begin{bmatrix} a_{1,1}B & \cdots & a_{1,m}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \cdots & a_{m,m}B \end{bmatrix}.$$

For example,

$$\begin{aligned} WHT(2^2) &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \end{aligned}$$

Direct implementation of this would be slow, so ...

WHT Factored: Scarier Formulas

to really amaze you

For positive integers n_i such that $n = n_1 + \cdots + n_t$, $WHT(2^n)$ can be rewritten as

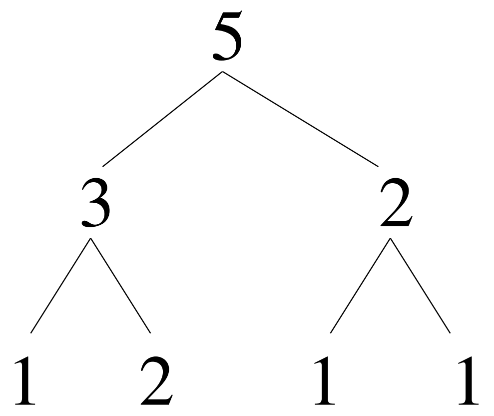
$$\prod_{i=1}^{t} \left(I_{2^{n_1+\cdots+n_{i-1}}} \otimes WHT(2^{n_i}) \otimes I_{2^{n_{i+1}+\cdots+n_t}}\right)$$

And Presto!
We can now write large WHTs in terms of smaller WHTs

For example,

$$\begin{aligned}
&WHT(2^5) \\
&\quad = [WHT(2^3) \otimes I_{2^2}][I_{2^3} \otimes WHT(2^2)] \\
&\quad = [\{(WHT(2^1) \otimes I_{2^2})(I_{2^1} \otimes WHT(2^2))\} \otimes I_{2^2}] \\
&\qquad [I_{2^3} \otimes \{(WHT(2^1) \otimes I_{2^1})(I_{2^1} \otimes WHT(2^1))\}]
\end{aligned}$$

But thankfully, we can visualize this as a split tree:



There is a 1-1 correspondence between trees and formulas

Big Numbers

to make you think this is REALLY hard

- For $WHT(2^n)$, there are on the order of $\theta(5^n/n^{3/2})$ different binary split trees
- $WHT(2^{10})$ has 51,819 different binary split trees
- And that is a pretty small transform!

- These different formulas can have very different running times, sometimes more than a factor of 10
- Small changes in the formula (split tree) can lead to significantly different running times

- How do we predict the fastest formula?

Features? What are Features?

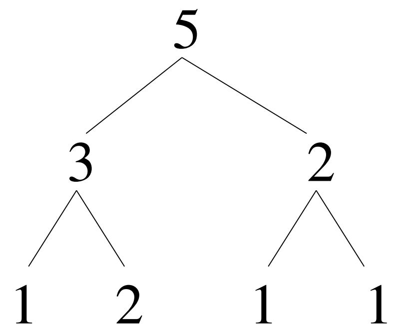
- Simplified Task: Learn to predict running times of formulas
- Can time some formulas to use as training data
- Train a neural network, for example, to predict running time
- **But** what *features* do we use to describe the formulas?
- ECE folks do not usually describe their world with "features"
- But they did have some ideas about what are important aspects of a formula
- They have found looking at different feature sets to be very useful
- Two broad categories of features:
  - ▷ Counting Nodes
  - ▷ Split Tree Shape

Counting Leaf Nodes

- Count the number of different sized leaves in the split tree
- These are the WHTs that must be computed directly
- For example,

$$[\{(WHT(2^1) \otimes I_{2^2})(I_{2^1} \otimes WHT(2^2))\} \otimes I_{2^2}]$$

$$[I_{2^3} \otimes \{(WHT(2^1) \otimes I_{2^1})(I_{2^1} \otimes WHT(2^1))\}]$$



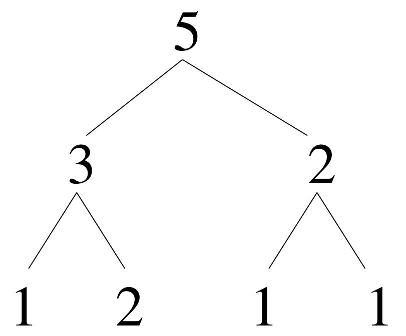
- 3 Leaf 1's
- 1 Leaf 2
- 0 Leaf 3's
- 0 Leaf 4's

Counting All Nodes

- Count the number of different sized nodes in the split tree
- This includes some information about what intermediate sizes are combined from smaller ones
- For example,

$$[\{(WHT(2^1) \otimes I_{2^2})(I_{2^1} \otimes WHT(2^2))\} \otimes I_{2^2}]$$

$$[I_{2^3} \otimes \{(WHT(2^1) \otimes I_{2^1})(I_{2^1} \otimes WHT(2^1))\}]$$



- 3 1-Nodes
- 2 2-Nodes
- 1 3-Node
- 0 4-Nodes

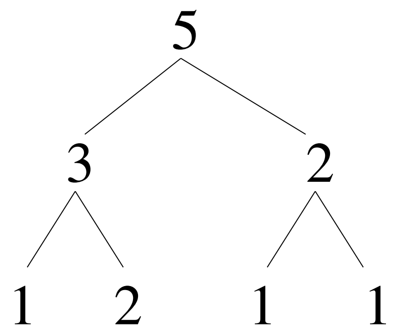
Counting Left and Right Nodes

- Previous features can be modified to include information about whether the leaf or node was a left or right child of its parent

- For example,

$$[\{(WHT(2^1) \otimes I_{2^2})(I_{2^1} \otimes WHT(2^2))\} \otimes I_{2^2}]$$

$$[I_{2^3} \otimes \{(WHT(2^1) \otimes I_{2^1})(I_{2^1} \otimes WHT(2^1))\}]$$



- 2 Left Leaf 1's
- 1 Right Leaf 1
- 0 Left Leaf 2's
- 1 Right Leaf 2
- 0 Left Leaf 3's
- 0 Right Leaf 3's

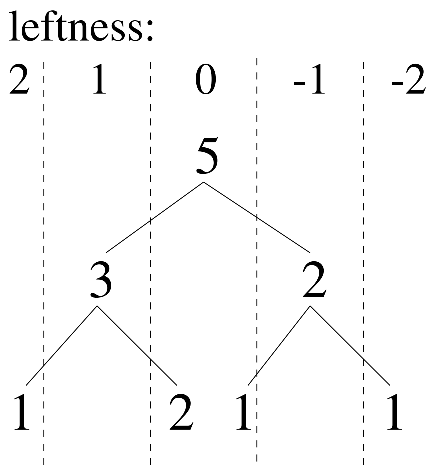
Vertical Profile

finally, something other than node counting

- Let the *leftness of a node* in a tree be the number of left children minus the number of right children along the path from the root to the given node
- Let the *leftness of a tree* be the sum of the leftness of all of the tree's nodes
- For example,

$$[\{(WHT(2^1) \otimes I_{2^2})(I_{2^1} \otimes WHT(2^2))\} \otimes I_{2^2}]$$

$$[I_{2^3} \otimes \{(WHT(2^1) \otimes I_{2^1})(I_{2^1} \otimes WHT(2^1))\}]$$



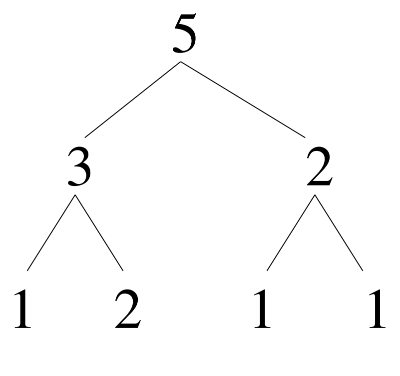
- 3 nodes with leftness 0
- 1 nodes with leftness 1
- 1 nodes with leftness 2
- 1 nodes with leftness -1
- 1 nodes with leftness -2
- Total leftness of 0

Horizontal Profile

- For a horizontal profile, we count the number of nodes at each depth
- As a single number measure, we can add up the path length of each node to the root
- As another single number measure, we can calculate the average path length
- For example,

$$[\{(WHT(2^1) \otimes I_{2^2})(I_{2^1} \otimes WHT(2^2))\} \otimes I_{2^2}]$$

$$[I_{2^3} \otimes \{(WHT(2^1) \otimes I_{2^1})(I_{2^1} \otimes WHT(2^1))\}]$$



- 1 node at depth 0
- 2 nodes at depth 1
- 4 nodes at depth 2
- Total path length of 10
- Average path length of 10/7

Evaluating Features

so are any of these good?

- ECE folks wanted to understand these features better
- Many formulas have the same feature values
- Features divide the formulas into partitions or equivalence classes
- Would like formulas in the same partition to have close running times
- To do the evaluations, we generated every formula for WHT's of sizes 2^{10} and smaller
- These formulas were each timed using a WHT package

Number of Partitions

see all the pretty numbers

Features	2^6	2^8	2^{10}
	WHT size		
Node Count:			
Leaf	11	22	40
All	31	168	947
L/R Leaf	44	142	395
L/R All	149	1832	23548
Shape:			
Leftness	19	41	71
Vert Prof	44	204	888
Tot Path Len	13	26	42
Avg Path Len	12	32	67
Horz Prof	13	38	115
Vert & Horz Prof	54	394	3043
Composite:			
All Nodes & Leftness	117	1222	12394
All Nodes & Vert Prof	122	1463	18966
All Nodes & Tot Path Len	35	220	1533
All Nodes & Horz Prof	35	220	1544
All Nodes, Vert & Horz Prof	123	1535	21140
All Formulas	188	2950	51819

Relative Standard Deviation

a bunch of numbers to make your eyes hurt

Weighted Average followed by Maximum

Features	WHT size 2^6		2^8		2^{10}	
Node Count:						
Leaf	14.0	26.4	16.0	25.7	15.6	23.6
All	3.2	6.0	3.3	14.7	3.0	10.9
L/R Leaf	12.0	25.5	14.8	25.5	14.8	24.6
L/R All	0.4	2.7	0.8	10.8	0.9	23.7
Shape:						
Leftness	35.0	55.7	32.9	40.1	30.4	37.2
Vert Prof	12.9	23.3	14.9	28.1	14.7	39.7
Tot Path Len	12.1	22.2	18.4	30.8	21.1	39.7
Avg Path Len	13.5	22.4	13.1	26.5	11.8	39.7
Horz Prof	12.1	22.2	12.4	26.5	11.4	39.7
Vert & Horz Prof	11.5	23.3	12.0	28.1	11.1	39.7
Composite:						
All Nodes & Leftness	1.3	5.2	1.7	26.0	1.8	26.9
All Nodes & Vert Prof	1.3	5.2	1.4	26.0	1.4	19.0
All Nodes & Tot Path Len	3.2	6.0	3.2	14.7	2.9	13.1
All Nodes & Horz Prof	3.2	6.0	3.2	14.7	2.9	13.1
All Nodes, Vert & Horz Prof	1.3	5.2	1.3	26.0	1.3	19.0
All Formulas	40.0	40.0	34.7	34.7	31.1	31.1

Doing some learning

finally some real machine learning

- Ultimately we would like to be able to predict the fastest running formula
- We trained a neural network to learn to predict running times for formulas
- We trained another neural network to predict which of two formulas will run faster
  - ▷ Don't really care how fast they run, just which one is the fastest
  - ▷ Can present two formulas at the inputs and have the network predict which one runs faster

Neural Network Predictions

Percent error predicting running time or the faster of 2 formulas for $WHT(2^8)$ (2950 different formulas)

Features	Train	Test	Cost	Faster
Leaf	100	100	14.21	28.72
	75	25	14.69	28.32
	50	50	14.61	28.63
	25	75	14.61	28.40
	10	90	14.94	28.33
All	100	100	2.74	5.33
	75	25	3.04	5.31
	50	50	3.15	5.39
	25	75	3.40	5.41
	10	90	3.95	5.71
L/R Leaf	100	100	13.69	25.92
	75	25	13.93	25.03
	50	50	14.18	25.06
	25	75	14.58	25.42
	10	90	15.52	24.98
L/R All	100	100	1.30	2.88
	75	25	1.68	3.00
	50	50	1.98	3.31
	25	75	2.71	3.51
	10	90	4.20	4.18

Conclusions

whatcha trying to say?

- Some features can partition the space of formulas better than others
- Some simple features though can do a good job
- A neural network can be trained to accurately predict running times of formulas

On to newer stuff!

Onward ho!

- The ECE folks noticed that none of the fastest formulas had leaf 1's
- Unrolled code can be faster than recursive calls

- Removing tress with any leaf 1 GREATLY reduces the total number of trees
- 51,819 binary trees of size 2^{10} but only 101 binary trees with no leaf 1's of size 2^{10}!

- We can then generate and time all binary trees with no leaf 1's for much larger sizes
- Allows us to do so for transform sizes that no longer fit into L1 cache

Number of Partitions Still Similar

See the pretty numbers again

Features	WHT size 2^{12}	2^{14}	2^{16}
Node Count:			
Leaf	18	27	40
All	74	226	722
L/R Leaf	96	204	405
L/R All	426	2070	10327
Shape:			
Leftness	19	29	41
Vert Prof	43	95	203
Tot Path Len	12	18	25
Avg Path Len	11	19	31
Horz Prof	12	21	37
Vert & Horz Prof	53	142	393
Composite:			
All Nodes & Leftness	271	1139	4848
All Nodes & Vert Prof	276	1206	5479
All Nodes & Tot Path Len	79	250	832
All Nodes & Horz Prof	79	250	832
All Nodes, Vert & Horz Prof	277	1219	5591
All Formulas	488	2489	13175

Relative Standard Deviation

Yikes! What happened?

Weighted Average followed by Maximum

Features	WHT size 2^{12}		2^{14}		2^{16}	
Node Count:						
Leaf	15.0	18.5	36.1	73.5	96.3	109.3
All	11.6	18.2	22.0	83.2	52.8	105.2
L/R Leaf	11.1	21.1	29.0	75.4	82.1	123.7
L/R All	2.0	22.4	3.4	70.3	3.8	80.5
Shape:						
Leftness	18.2	23.4	36.6	69.6	76.9	117.8
Vert Prof	15.5	21.5	28.0	80.4	53.6	119.9
Tot Path Len	15.5	21.4	34.0	82.1	75.7	107.1
Avg Path Len	15.5	21.4	34.7	82.1	75.6	107.1
Horz Prof	15.5	21.4	33.1	74.1	68.8	95.6
Vert & Horz Prof	15.2	21.5	23.0	69.7	31.2	94.9
Composite:						
All Nodes & Leftness	4.7	18.4	7.8	73.4	16.2	119.7
All Nodes & Vert Prof	4.6	18.4	6.7	69.7	11.5	115.9
All Nodes & Tot Path Len	11.5	18.2	21.8	84.0	52.3	105.3
All Nodes & Horz Prof	11.5	18.2	21.8	84.0	52.3	105.3
All Nodes, Vert & Horz Prof	4.5	18.4	6.6	69.7	10.9	104.3
All Formulas	19.9	19.9	47.5	47.5	99.9	99.9

Beyond L1 Cache

Bad features! Bad features!

- Beyond the L1 Cache, the only features that partitioned well were those that had almost as many partitions as formulas
- Further, preliminary runs have been unsuccessful on training a neural network for these sizes using even the Left/Right All Nodes features
- Any suggestions for other features to try?

Search Techniques

So, what about the *real* problem?

- Recall that ultimately we would like to find the fastest formula, not just predict running times

- ECE folks have been using dynamic programming:
  - ▷ Try all possible ways to split the root node
  - ▷ For each child, use previously found best way to split it
  - ▷ Assumes the best way to split a node is independent of location in tree
  - ▷ Actually works surprisingly well

- José Moura in ECE suggested the following:
  - ▷ Partition the space of formulas according to a feature set
  - ▷ Generate and time a few random formulas in each partition
  - ▷ Choose a few partitions with the fastest formulas and repeat
  - ▷ Does about as well as dynamic programming but times many more formulas

How about GA's?

How about them?

- Instead of using bits, maintain trees
- Many "obvious" possibilities then for mutation and cross-over
- Do not need to restrict ourselves to binary trees

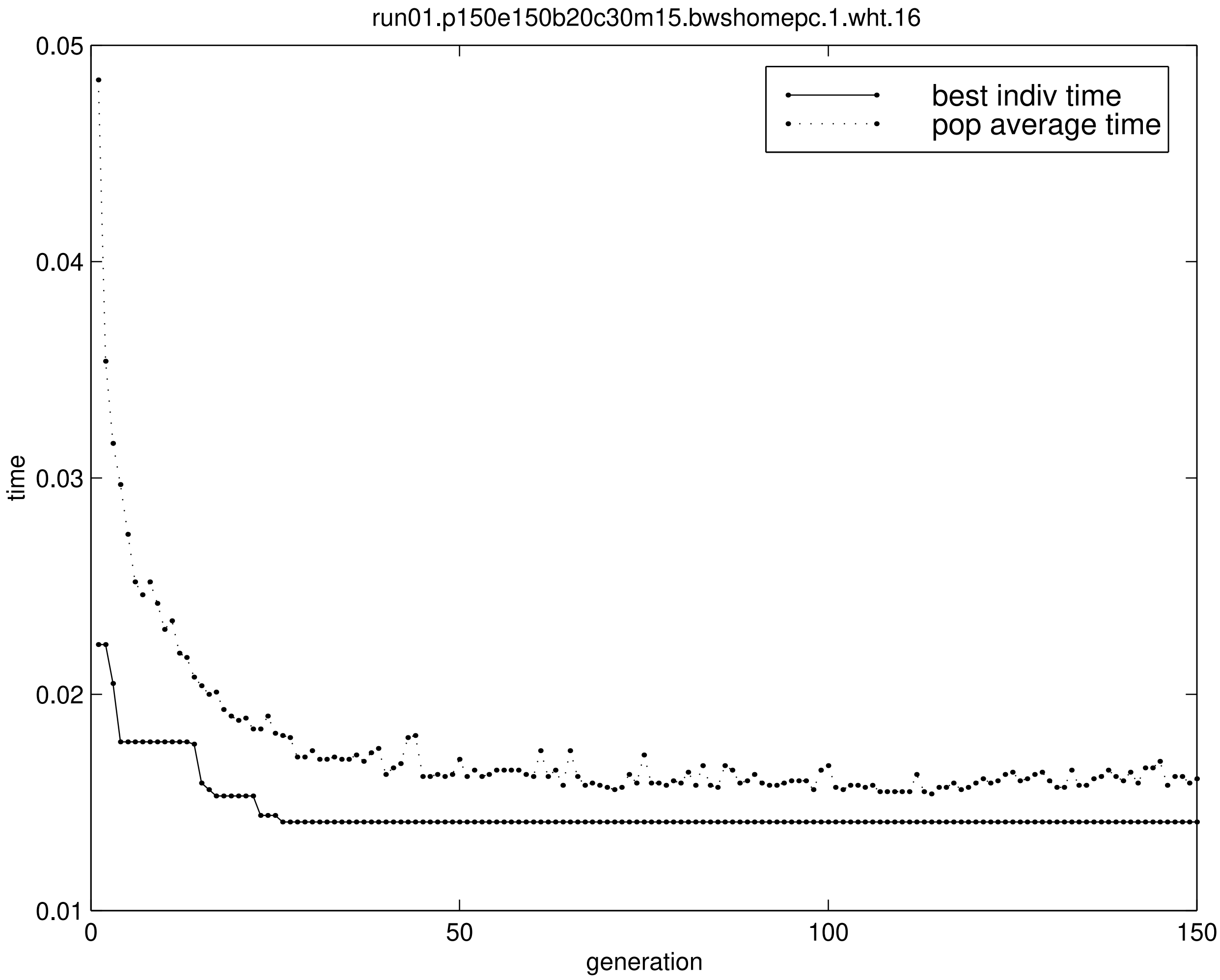
GA details

What you always wanted to know

- Random Population:
  - ▷ Choose a random set of children and recurse on each
  - ▷ Choose a random set of leaves and randomly combine
- Selection:
  - ▷ Tree added to new population with probability proportional to 1 over its running time
- Mutations:
  - ▷ Pick a random internal node and swap 2 of its children
  - ▷ Pick a random node and move it and its subtree anywhere
  - ▷ Pick a random internal node that could be a leaf and make it a leaf
  - ▷ Pick a random leaf that could have children and give it children
- Crossover:
  - ▷ Pick a random node from each tree of the same size and swap the nodes and their subtrees

GA in Action

Don't you wish you could generate pretty graphs like this?



- Does significantly better than DP in some cases
- Optimal formula found is often non-binary