Designing adaptive online algorithms by maintaining a mixture over a set of experts

Manfred K. Warmuth    Corrie Scalisi
Includes some earlier work with
Robert Gramacy, Scott Brandt and Ismail Ari

University of California - Santa Cruz

April 8, 2013 - 290C, lecture 3
Outline

1. Two example problems
2. Measuring the on-lineness of the data
3. The expert framework
4. Shifting experts
5. Experimental results
6. Wrap up
Outline

1. Two example problems
2. Measuring the on-lineness of the data
3. The expert framework
4. Shifting experts
5. Experimental results
6. Wrap up
Two example problems

1. Disk spindown problem

- When to spin down the disk on your laptop?
- Best time-out time/user/usage dependent

![First 3000 iterations of the Cello-2 Dataset](image)
Two example problems

Non-convex loss

If idles times expected to be
- short, then long timeout better
- long, then short timeout better
2. Caching

Whenever small, fast memory and larger, slower secondary memory.

Keep objects in faster memory which likely to be needed again soon.
- **Hit** if requested object resides in cache
- **Miss** otherwise
Caching Policies

- Decides which objects to discard to make room for new requests
- 7 common policies: LRU, RAND, FIFO, LIFO, LFU and MFU
- 5 fancy recent policies: SIZE, GDS, GD*, GDSF, LFUDA
- Criteria:
  - Recency and frequency of access
  - Size of objects
  - Cost of fetching object from secondary memory
- De facto standard: LRU
Two example problems

Which Policy to Choose?

For which situation?
- Disk access on PC
- Web proxy access via browser
- File server on local network
- Middle of the night - during backup
- Application as well as time dependent

Choosing one is suboptimal

All policies claimed to be on-line/adaptive
Characteristics Vary with Time
Two example problems

Best Policy Varies with time
Outline

1. Two example problems
2. Measuring the on-lineness of the data
3. The expert framework
4. Shifting experts
5. Experimental results
6. Wrap up
First trick: Permute the data

- Data not on-line if permuting does not change things
- Algorithm not adaptive, if same performance on permuted data
  - adaptive, if better performance on unpermuted data
Measuring the on-lineness of the data

Permuting trick for disk spindown data

![First 3000 iterations of the Cello-2 Dataset](image1)

![First 3000 durations in the Intel dataset](image2)

on-line :-)  

![First 3000 iterations of Random Permutation of the Cello-2 Dataset](image3)

![First 3000 durations in a randomized permutation of the Intel dataset](image4)

not on-line :-(

M. Warmuth (UCSC)  
Designing adaptive online algorithms
Permuting caching data

highly on-line data

some caching policies already on-line
Using a **comparators** to measure on-lineness of data

**Properties**

- Should exploit on-lineness of data
- Might be too expensive to compute in practice, but can serve as a goal to compare against
- Might rely on information not available to the on-line algorithm
Good comparators?

BestFixed chosen in hind-sight
- Does not capture on-lineness of data since same performance on original and permuted data

Optimal algorithm
- Spin down iff next idle time > spindown cost
- Captures on-lineness, but may be unrealistic
Reasonable comparator for the disk spindown problem

BestShift(\(K\)) for spindown problem
- Partition of the timeline into segments
- BestFixed in each segment

| 2 | 4 | 7 |

![Diagram showing average energy vs total # of shifts with a point representing partition]
Measuring the on-lineness of the data

**BestFixed**($K$)

Dynamic programming: $O(KN^2T)$

where $K$ # of partitions, $N$ # of discrete idle times, $T$ # of trials

[H]
Measuring the on-lineness of the data

BestShift curves

BestShift(K) on Cello-2 Data
50 experts exponentially spaced between 0 and 10

BestShift(K) on Intel dataset
50 experts exponentially spaced between 0 and 10

on-line

not on-line
Comparitors for caching

- **BestFixed**: a posteriori best of 12 policies on entire request stream
- **BestRefetching($R$)**: minimum number of misses with at most $R$ refetches in any sequence of switching policies
Comparator: All sequences of the form

We plot miss rate v.s. refetches:
Measuring the on-lineness of the data

**BestRefetching**($R$)

Dynamic programming: $O(RN^2 T)$
Goal for on-line algorithms

- Beat BestFixed (easy)
- Get close to BestShift / BestRefetching
- In caching reduce I/O’s and end-user latency
- Fast algorithms
Measuring the on-lineness of the data

Score card for caching algorithms

- Miss Rate (%)
- Refetches as % of Total Requests

A = Total I/Os less than BestFixed
B = Total I/Os less than LRU
C = Total I/O more than LRU

A+ = Better than BestRefetching
B+ = Worse than BestFixed
C+ = BestRefetching

miss + refetch <= LRU miss

BestFixed
BestRefetching
AllVC
Outline

1. Two example problems
2. Measuring the on-lineness of the data
3. The expert framework
4. Shifting experts
5. Experimental results
6. Wrap up
What experts?

Caching:
- 12 caching policies

Disk spin down:
- Discretize interval $[0, \text{spindowncost}]$
On-line algorithm for learning as well as best experts

One weight per expert

- Represent confidence of master algorithm in expert
- Master algorithm predicts with convex combination of experts
- Loss update: \( w_i^{t+1} \sim \frac{w_i^t e^{-\eta L_i^t}}{Z_t} \) \[\text{[LW, V]}\]
- Designed to do well against BestFixed
- In some cases log \( N \) regret
Outline

1. Two example problems
2. Measuring the on-lineness of the data
3. The expert framework
4. Shifting experts
5. Experimental results
6. Wrap up
As well as best partition

- Loss Update follows too well
- Follow it by Share Update:
  - Mix in small in $\alpha = 5\%$ times past average weight
  - Updates recover after each shift
  - Faster recovery if expert was used before
  - In some cases regret $= \#$ of bits to encode best partition
1. Two example problems
2. Measuring the on-lineness of the data
3. The expert framework
4. Shifting experts
5. Experimental results
6. Wrap up
Spindown results

Experimental results

on-line :-)  

not on-line :-(

M. Warmuth (UCSC)  
Designing adaptive online algorithms
Experimental results

Caching - we “Tracks” best policy

Miss-rates under FSUP with Master

Requests Over Time t

Miss-rates

- lru
- fifo
- mru
- lifo
- size
- lfu
- mfu
- rand
- gds
- gdsf
- lfuda
- gd
- roll
Experimental results

**WWk Master and Comparator Missrates**

- **8.5% = LRU missrate**
- **2.0% = Obligatory missrate**

Graph showing the miss rates for different refetching strategies as a percentage of total requests. The strategies include:
- BestRefetching(R)
- Rank Ideal
- Rank 60% Ideal
- Rank 40% Ideal
- BestFixed = SIZE
- AllVC

Miss rates range from 4.0% to 5.5%.
UMo Master and Comparator Missrates

16.6% = LRU missrate
1.5% = Obligatory missrate

- BestRefetching(R)
- Rank Ideal
- Rank 60% Ideal
- Rank 40% Ideal
- BestFixed = GDS
- AllVC

Missrates %

Refetches as % of Total Requests
Outline

1. Two example problems
2. Measuring the on-lineness of the data
3. The expert framework
4. Shifting experts
5. Experimental results
6. Wrap up
Pushing the theoretical analysis

Disk spindown:
- Non-convex loss, but in each trial only two loss values
- Experts are sorted
- Analyze with continuously many experts

Caching:
- Prove bounds for ARCing
The upshot

- Measure on-lineness of data
- Design algorithms that provably exploit on-lineness
- Many simple on-line problems amenable to theoretical analysis