Measuring the "on-lineness" of data streams

Manfred K. Warmuth **Jiazhong Nie**

University of California - Santa Cruz

Dec. 10, 2015 —- Nips workshop on Easy Data

Includes some earlier work with
Corrie Scalisi, Robert Gramacy, Scott Brandt and Ismail Ari

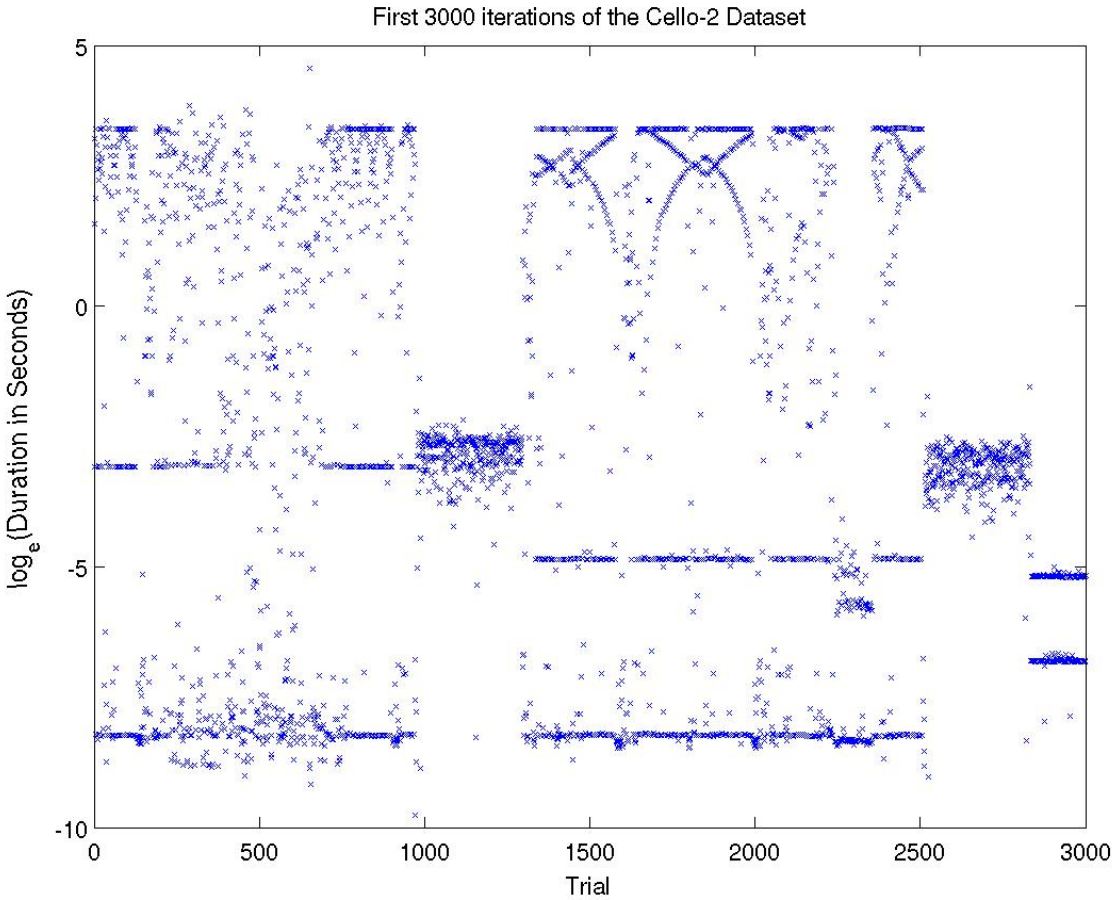
Goals

- Design on-line algorithms in domains that are outside of the reach of theory

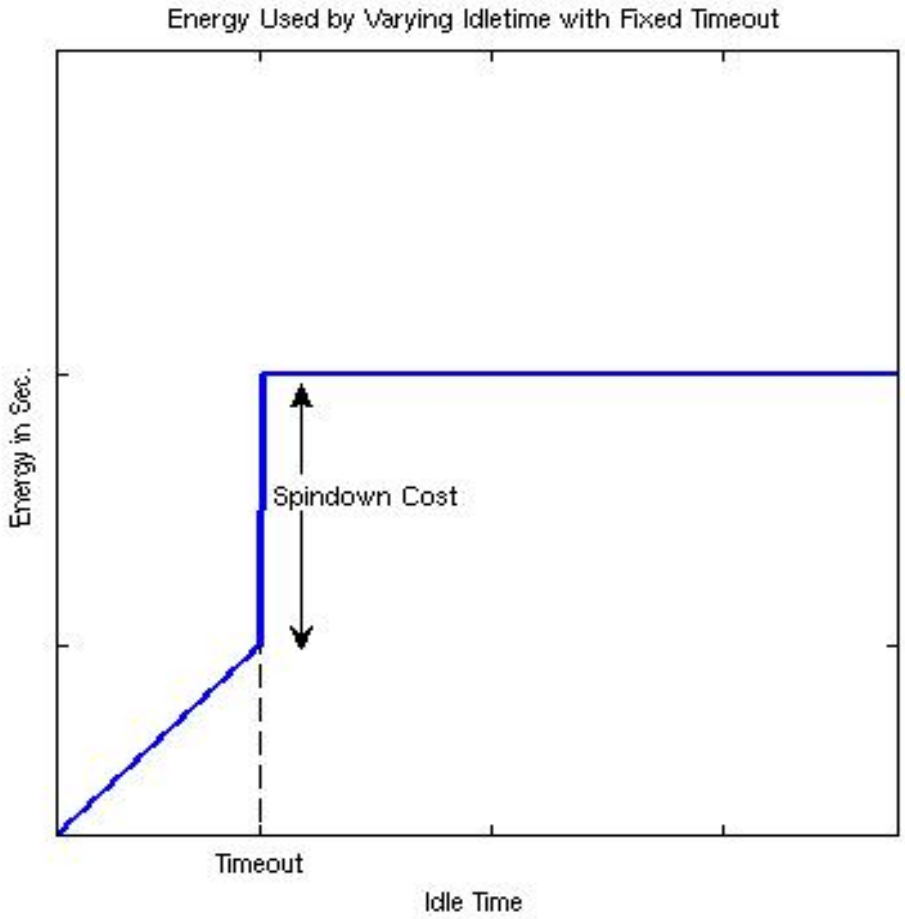
- Design good comparators that exploit the on-lineness of the data

1. Disk spindown problem [HLSS]

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



Non-convex loss



If idles times expected to be

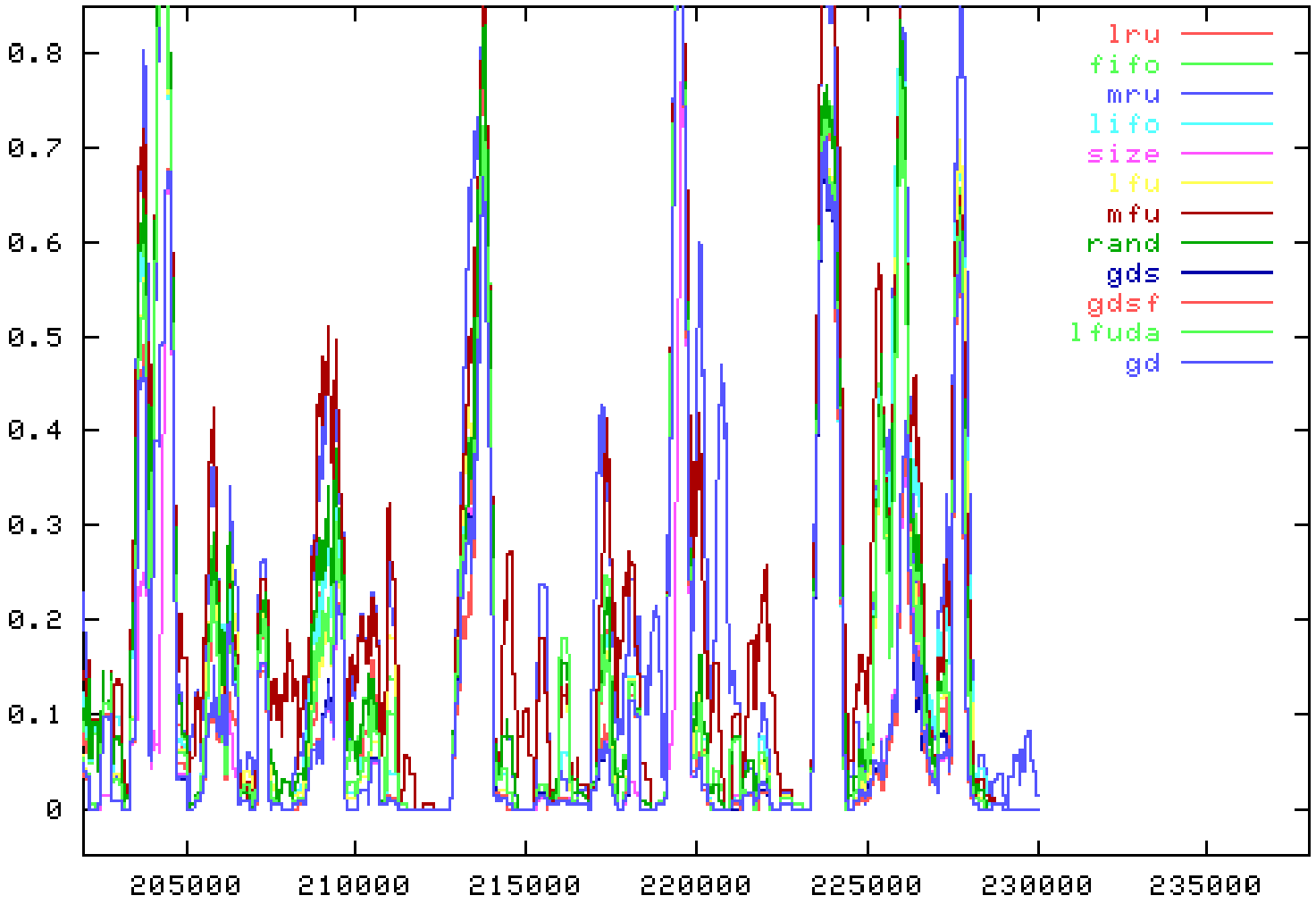
- short, then long timeout better
- long, then short timeout better

2. Caching [BWBA]

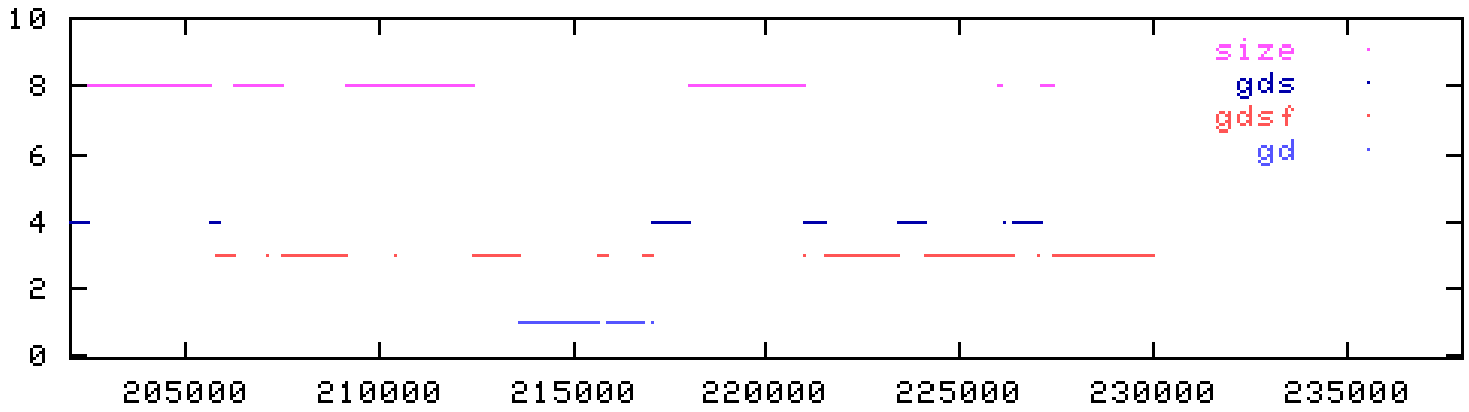
- Want to build combined caching policy from 12 base policies (our experts):

  LRU, RAND, FIFO, LIFO, LFU, MFU, SIZE, GDS, GD*, GDSF, LFUDA

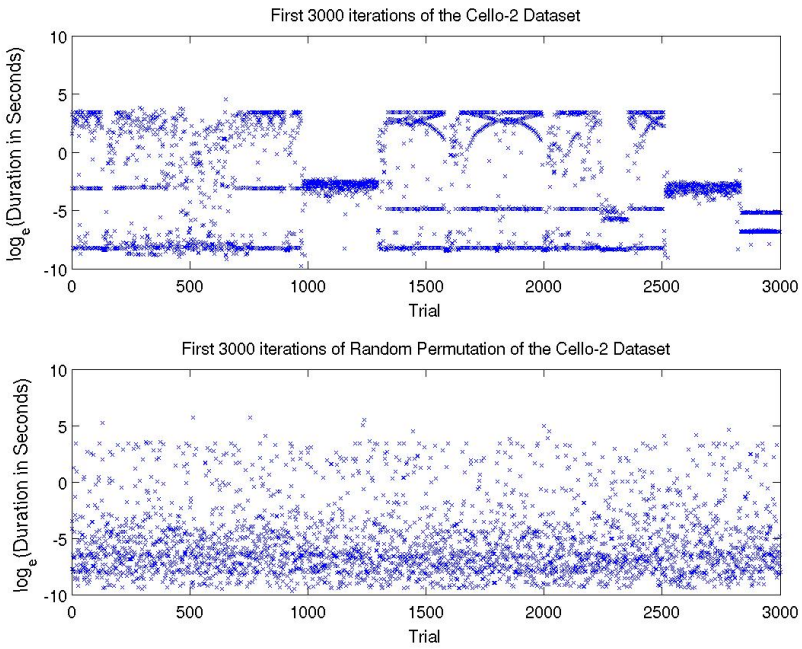
Characteristics Vary with Time



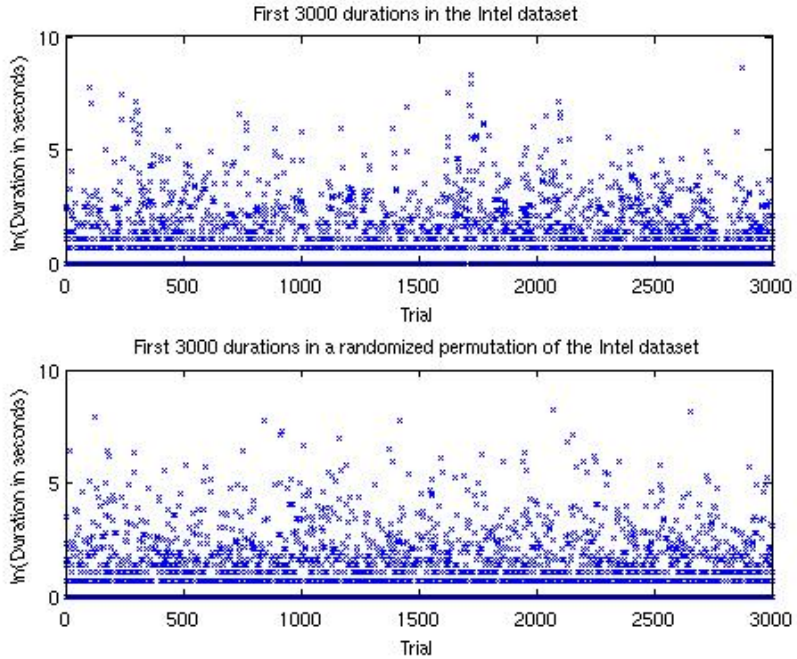
Best Policy Varies with time



Permuting trick for disk spindown data

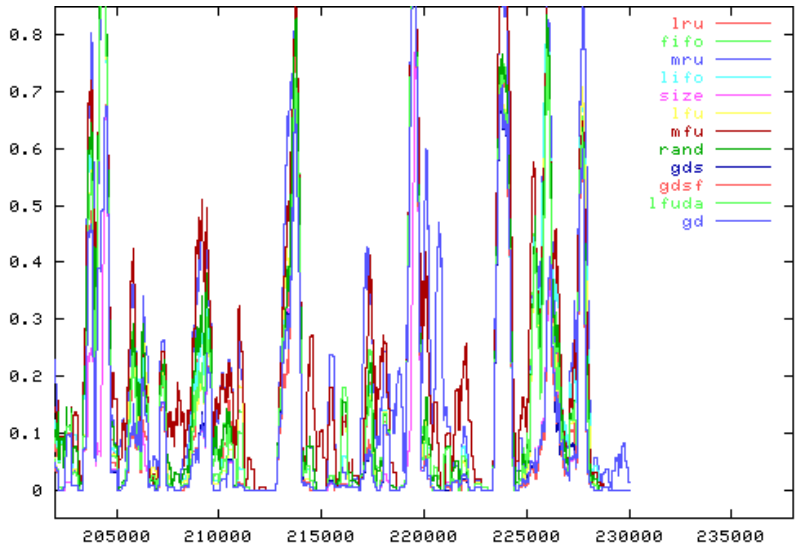


on-line :-)

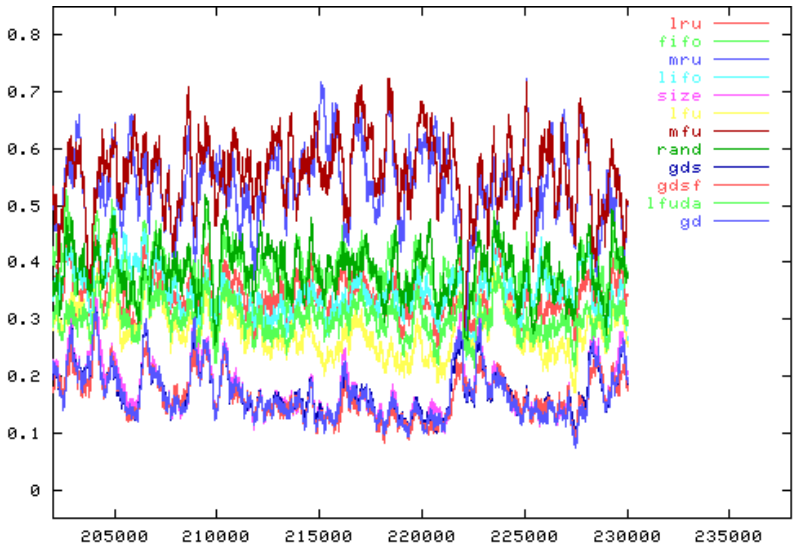


not on-line :-(

Permuting caching data



highly on-line data



some caching policies already on-line

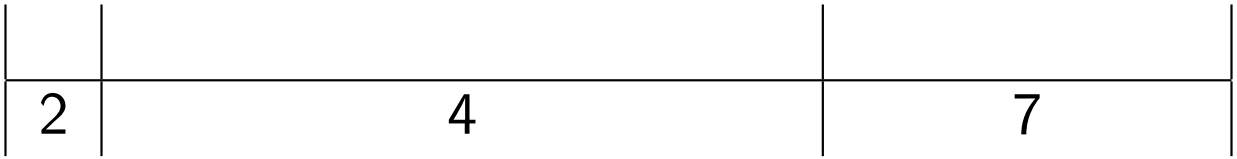
Using a comparators to measure on-lineness of data

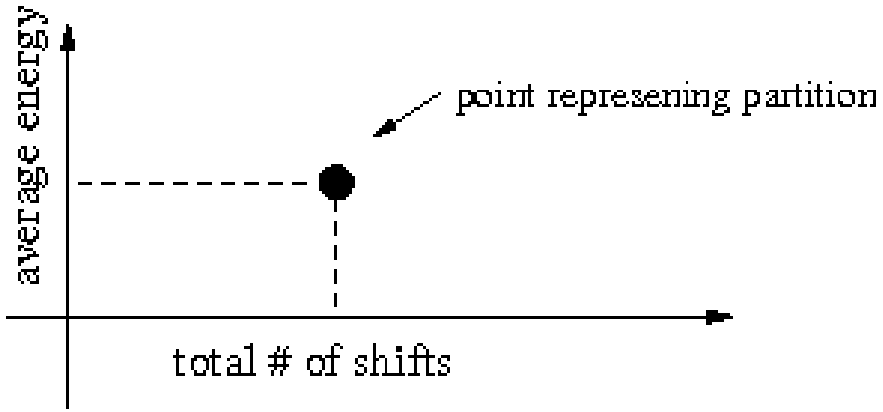
Properties

- 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

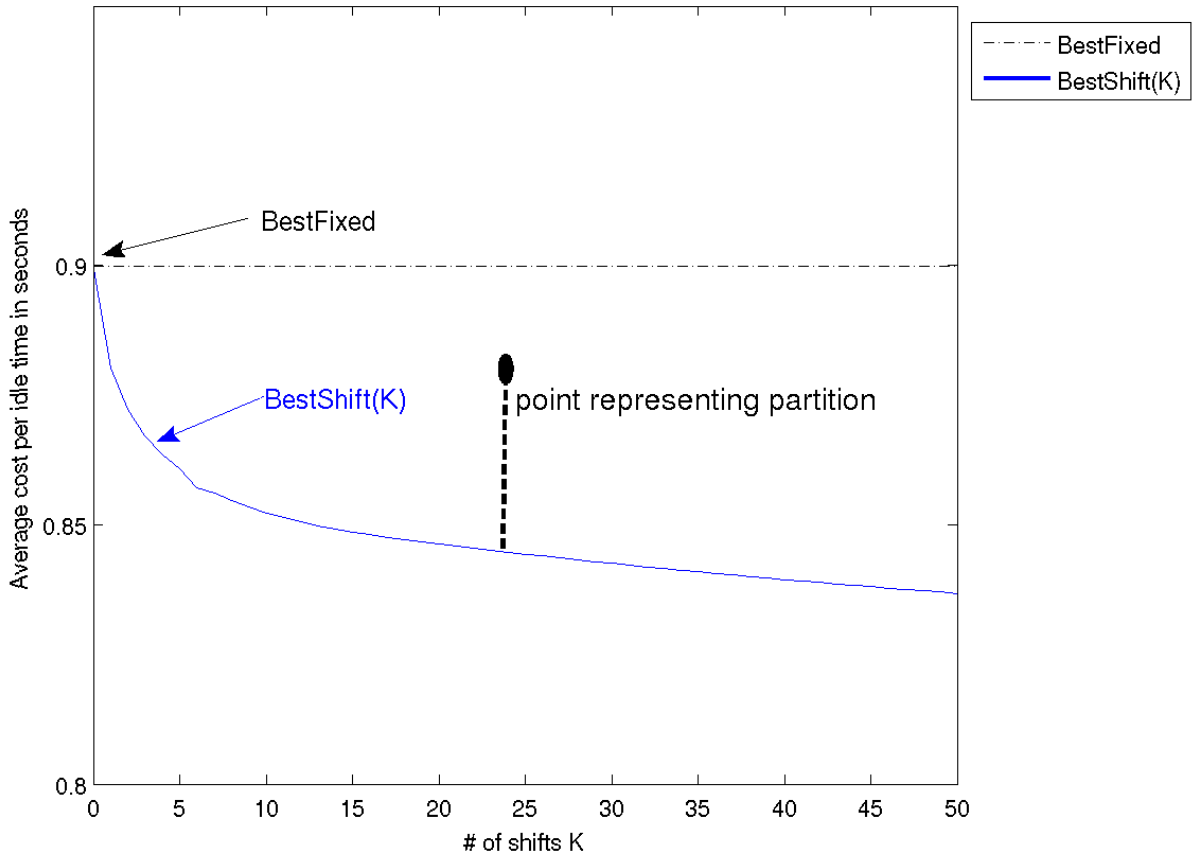
Idea 1: Use dynamic programming to compute BestShift(K) curve

- Partition of the timeline into K segments
- BestFixed in each segment





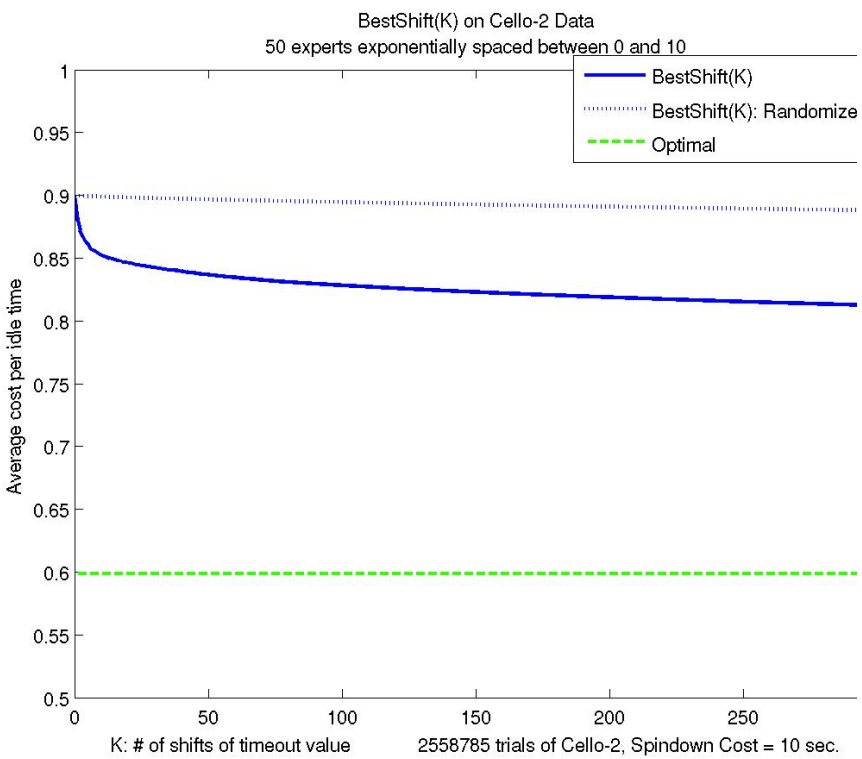
BestFixed(K)



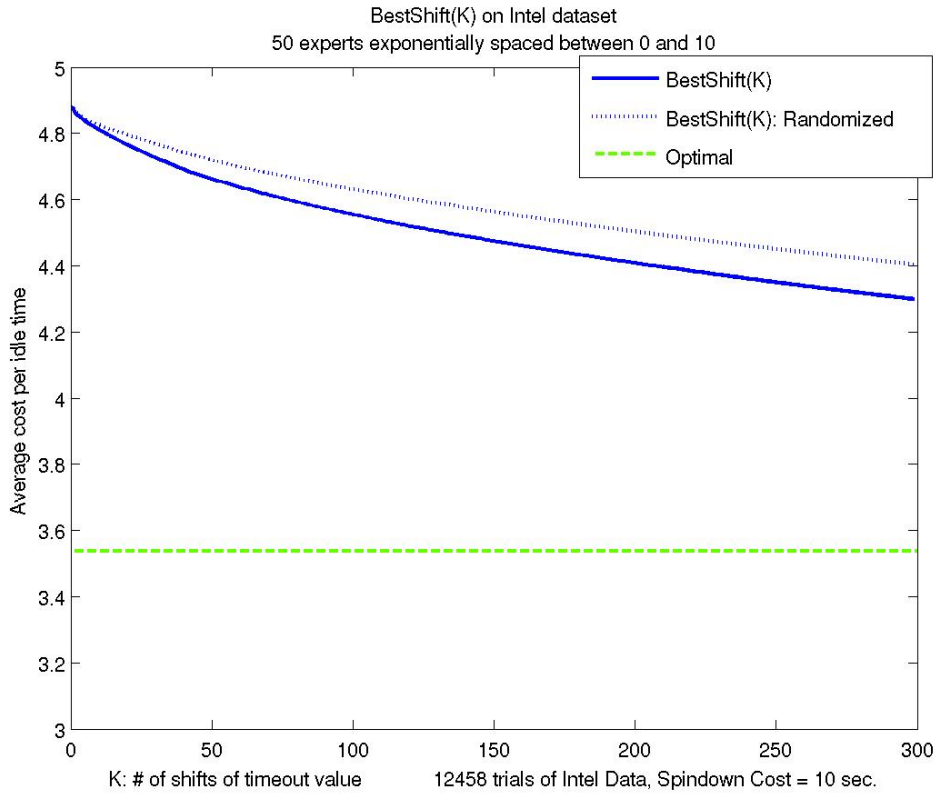
Dynamic programming: $O(KN^2T)$ [H]

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

BestShift curves



on-line



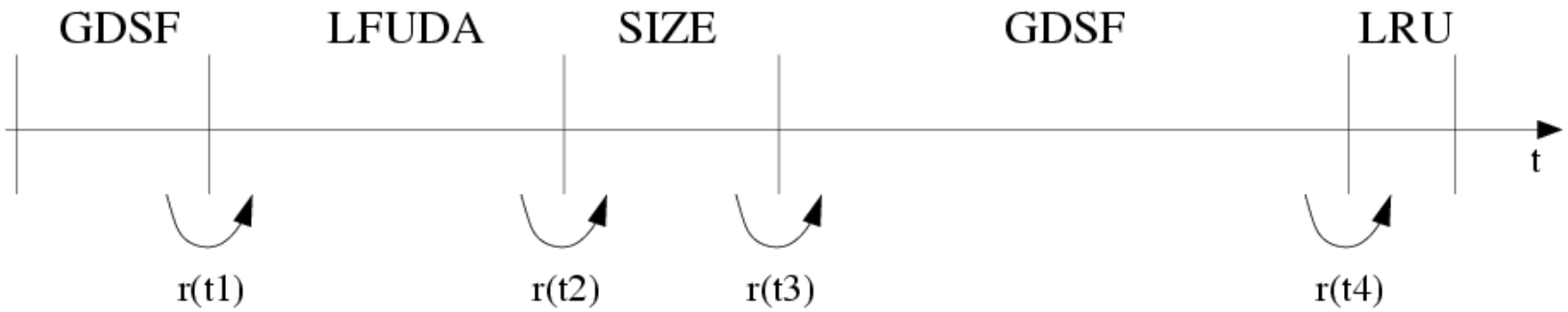
not on-line

Comparators for caching

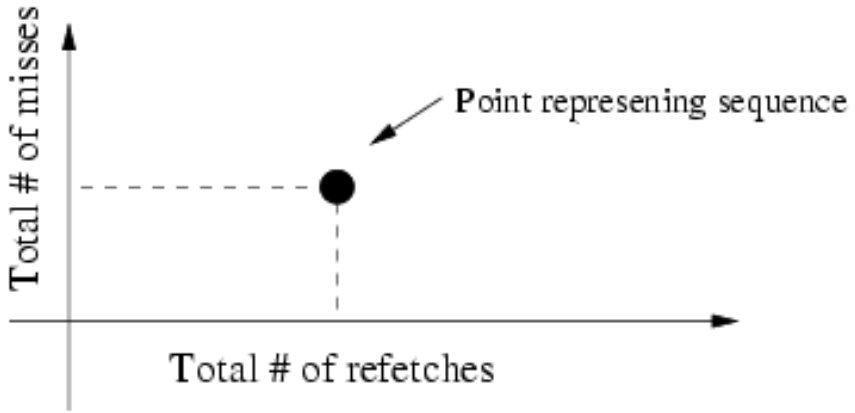
- **BestFixed**: a posteriori best of 12 policies on entire request stream
- **BestRefetching**(R):
  minimum number of misses with at most R refetches
  in any sequence of switching policies

Refetches & Policy Switches

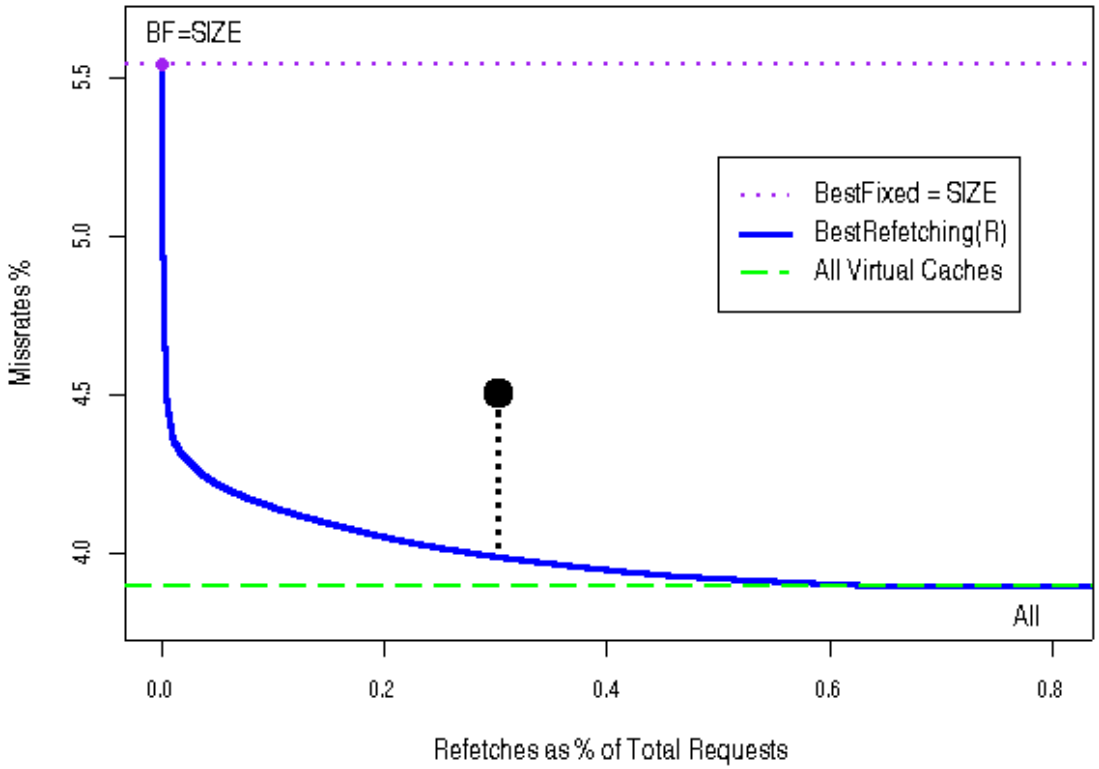
Comparator: All sequences of the form



We plot miss rate v.s. refetches:



BestRefetching(R)



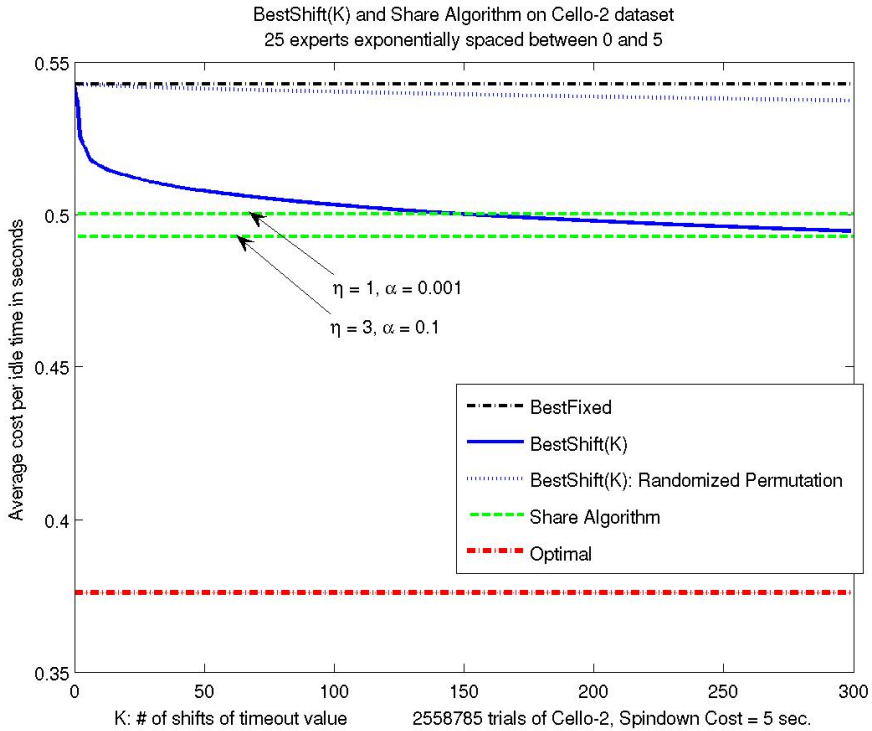
Dynamic programming: $O(RN^2T)$ [H]

Our theoretically sound algorithms become heuristics

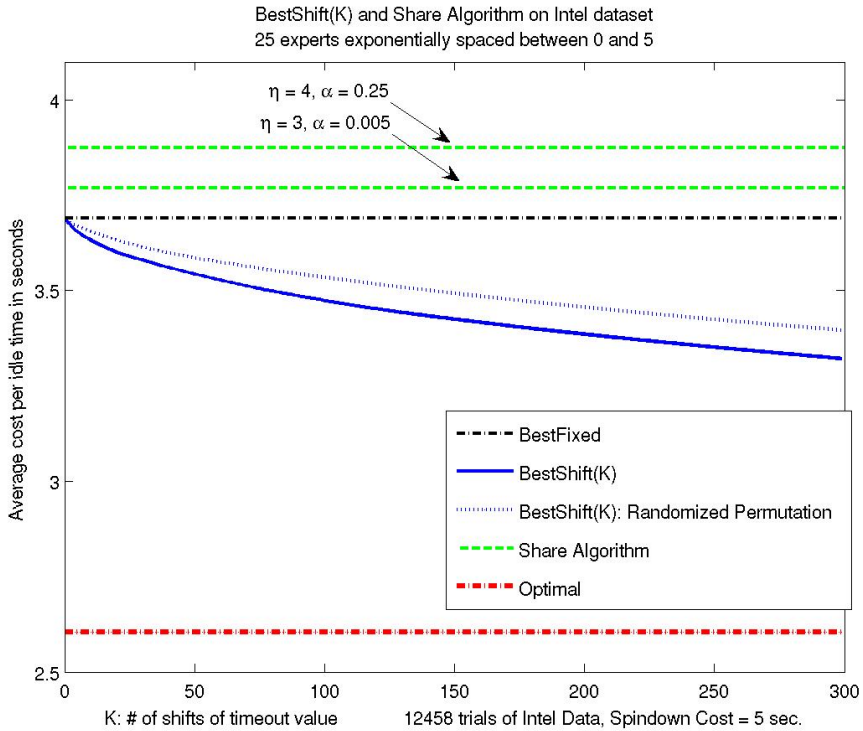
- Use loss and share updates on non-convex losses

- Build a merged cache that does not correspond to the mixture

Spindown results

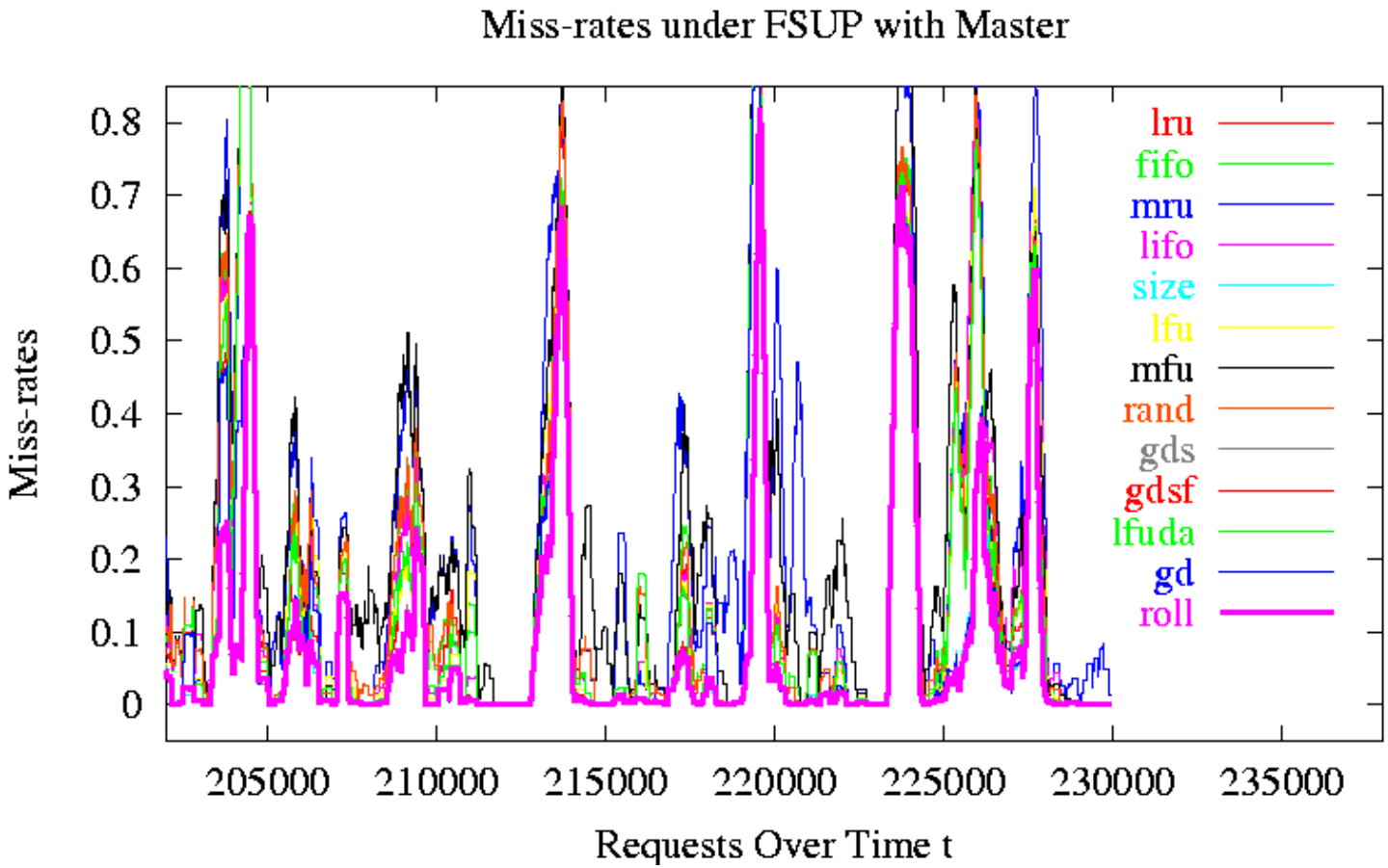


on-line :-)

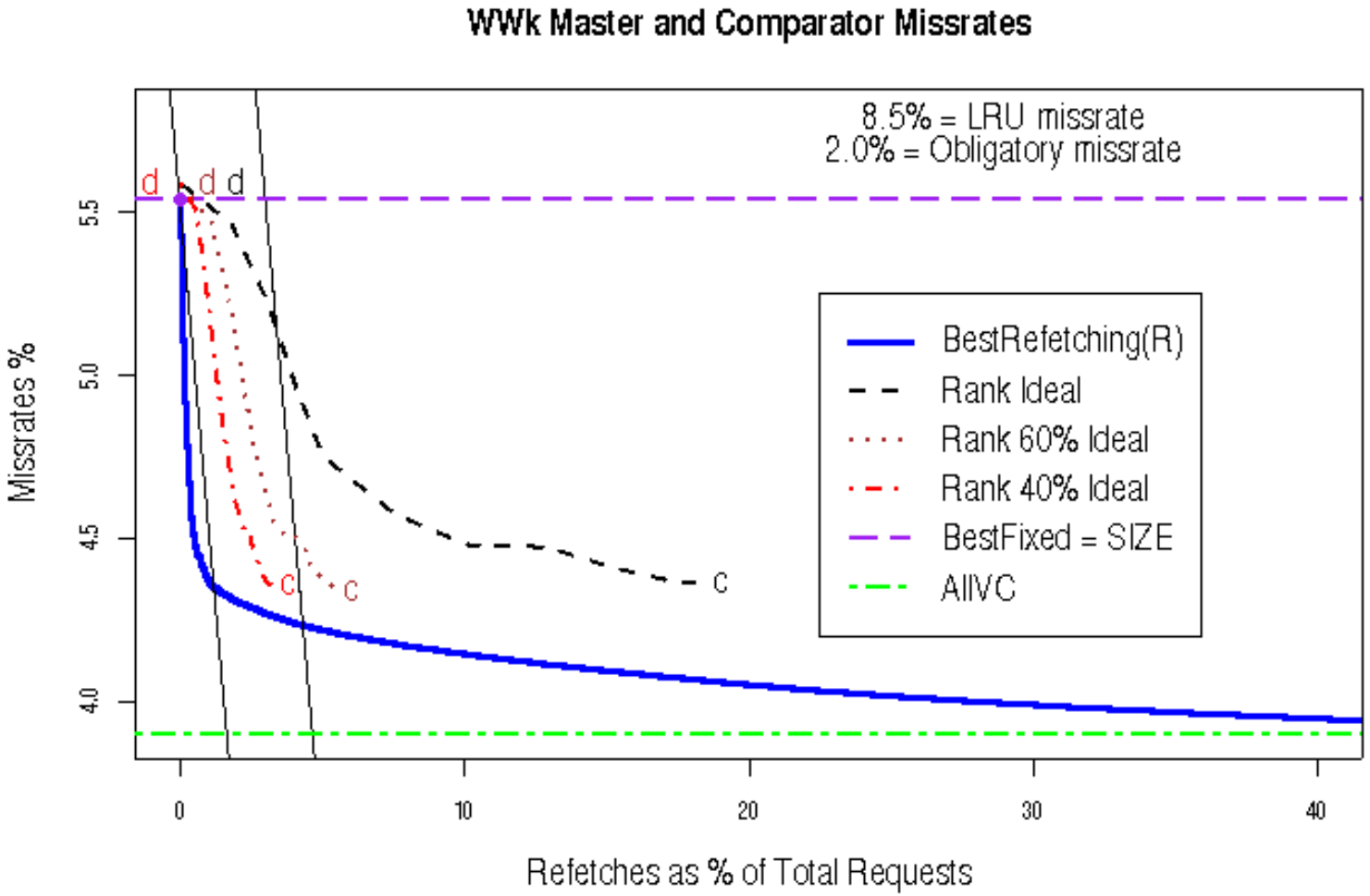


not on-line :-(

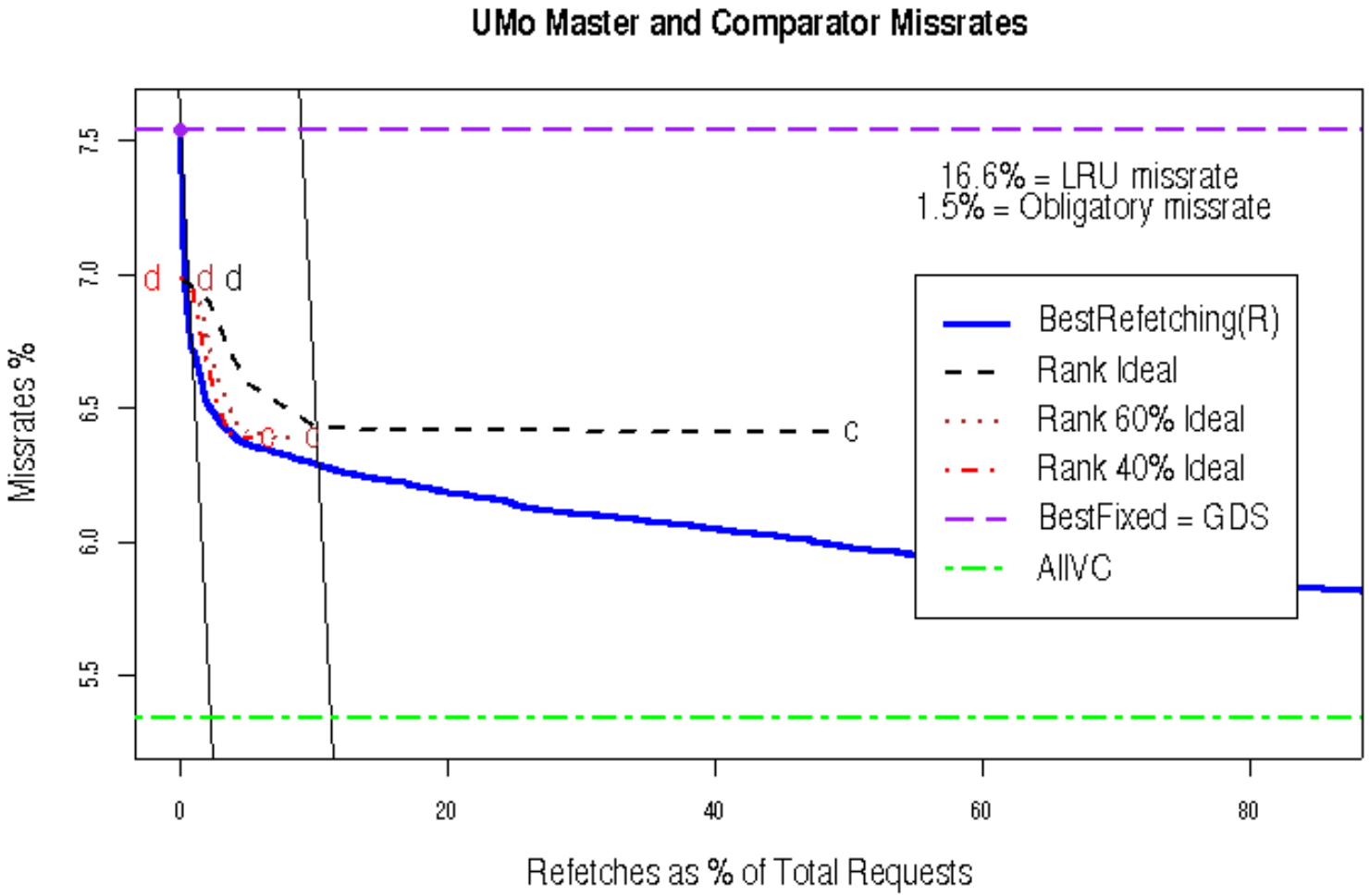
Caching - we "Tracks" best policy



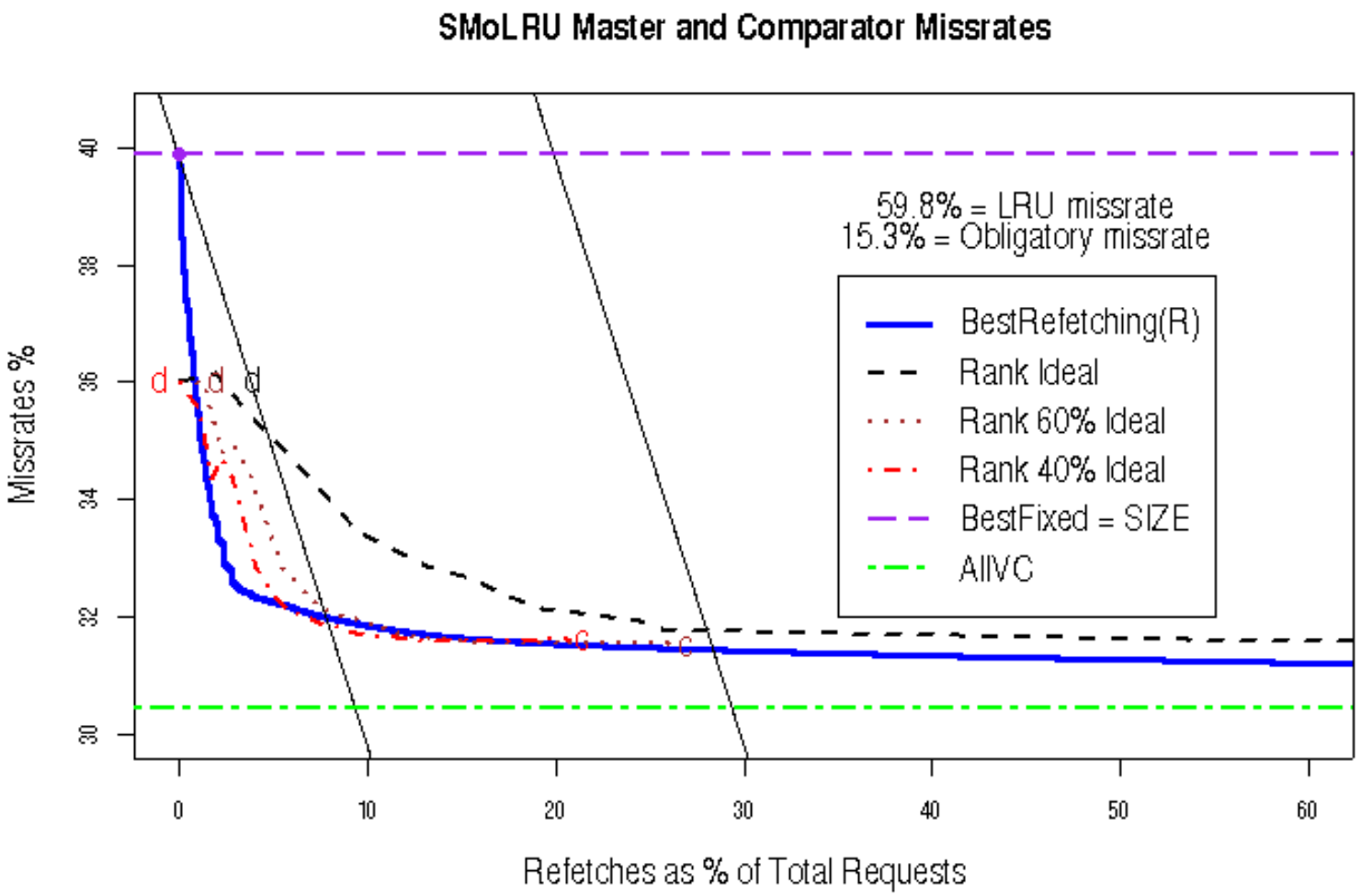
WWk


WWk Master and Comparator Missrates
8.5% = LRU missrate
2.0% = Obligatory missrate
BestRefetching(R)
Rank Ideal
Rank 60% Ideal
Rank 40% Ideal
BestFixed = SIZE
AllVC
Missrates %
Refetches as % of Total Requests
5.5
5.0
4.5
4.0
0
10
20
30
40
d
d
d
c
c
c

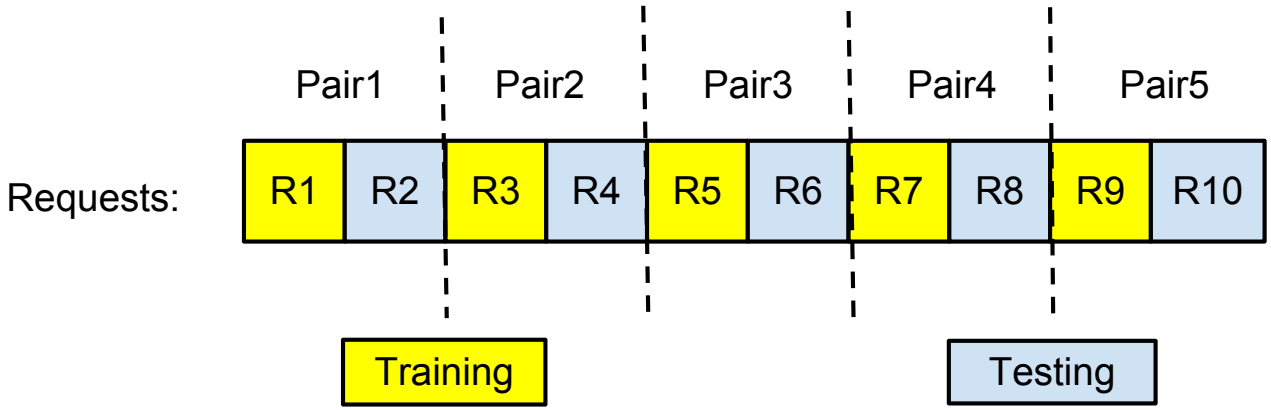
UMo


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
7.5
7.0
6.5
6.0
5.5
0
20
40
60
80
d
c

SMoLRU


SMoLRU Master and Comparator Missrates
59.8% = LRU missrate
15.3% = Obligatory missrate
BestRefetching(R)
Rank Ideal
Rank 60% Ideal
Rank 40% Ideal
BestFixed = SIZE
AllVC
Missrates %
Refetches as % of Total Requests
30
32
34
36
38
40
0
10
20
30
40
50
60

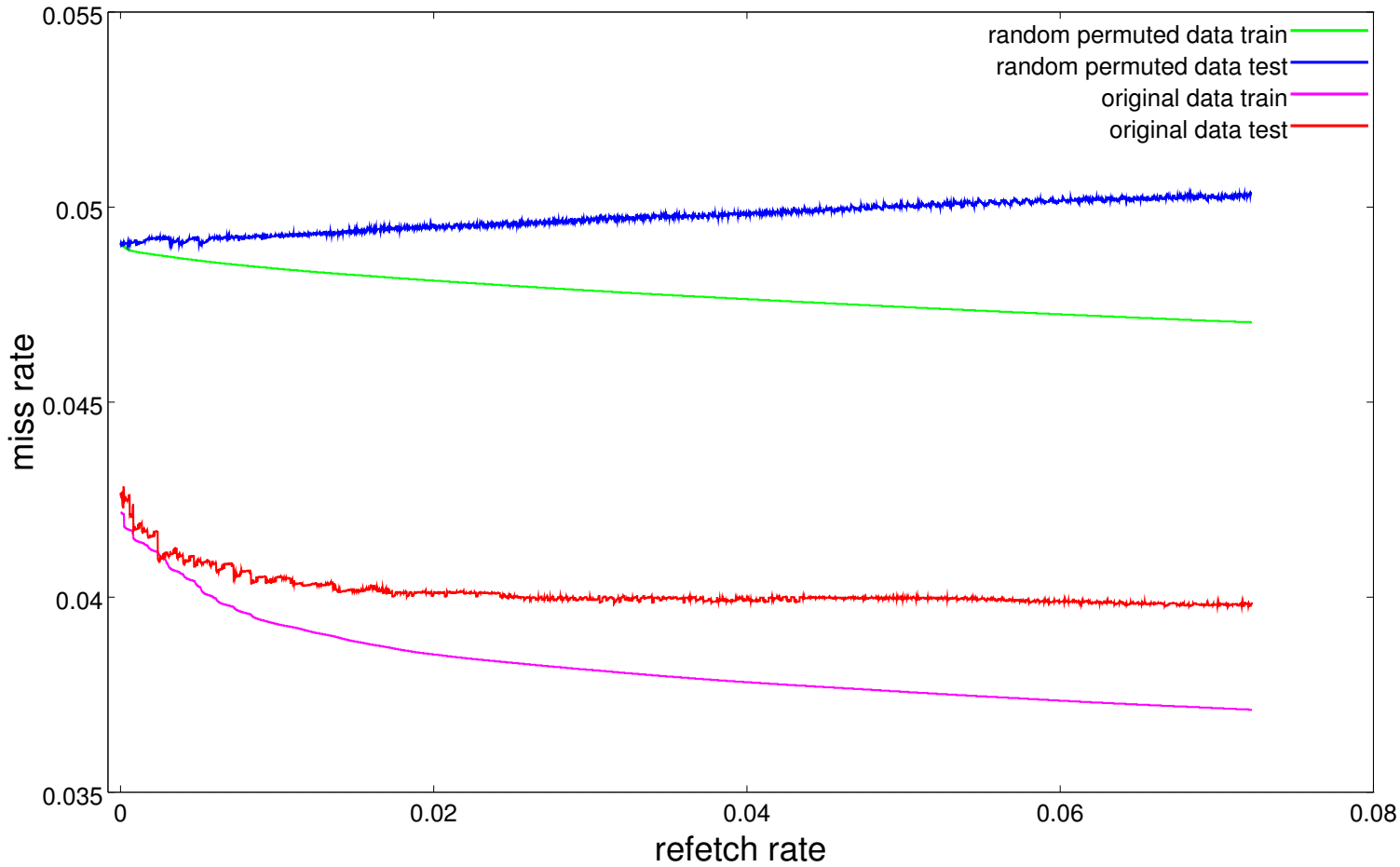
Idea 2: Split into even/odd requests



- Best partition based on training set
- Performance based on test set

Miss Rate of Testing Requests

No overfitting to random data: testing miss rate goes up immediately



Upshot!

Don't be afraid
to use your algorithms
as heuristics in domains
where the theory breaks down