COS 423 Lecture 2

On-line vs. Off-line Algorithms: Competitive Analysis

Move-to-Front List Rearrangement

The skier's dilemma: to rent or to buy

Number of ski trips is unknown: depends on enjoyment, injuries, etc.

Goal: minimize $ spent

Off-line (knowing the future):

k = #trips, b = price to buy, r = price to rent

If $b \leq kr$ buy, otherwise rent.

But k is unknown!

Rule of thumb: rent until total spent would exceed cost to buy, then buy.

No matter what k turns out to be, $ spent is at most twice the minimum:

If $kr < b$, spending is kr (minimum).

If $kr \geq b$, spending is at most $2b$, minimum is b.

This strategy is *2-competitive*.

In general, an algorithm is *k-competitive* among a family of algorithms if its cost on any input is at most *k* times the cost of the best algorithm in the family on the specific input.

This concept gives us a robust way to measure the efficiency of an on-line algorithm: we compare it to the best off-line algorithm (which knows the entire input sequence) and take the worst-case performance ratio over all input sequences.

Example: self-adjusting lists

Dictionary: contains a set *S* of items, each with associated information.

Operations:

Access(*x*): Determine if *x* is in *S*. If so, return *x*'s information.

Insert(*x*):(x not in S) Insert *x* and its information.

Delete(*x*):(x in S) Delete *x* and its information.

Implementations

- List
- Hash table
- Search Tree
- van Emde Boas Tree

List: simple but costly

Access is via sequential search: start from front, traverse list until finding item or reaching end of list. To insert an item, traverse list (to verify item not present), add to end. To delete an item, find it, remove it, move following items forward one position.

Worst-case cost per access is *n*, average cost is *n*/2 (if items equally likely).

A, S, F, G, Y, T

access *G* costs 4

A, S, F, G, Y, T

insert *B* costs 7

A, S, F, G, Y, T, B

delete *G* costs 7 (if array)

A, S, F, Y, T, B

What if items have differing access frequencies, or access sequence has other kinds of locality, such as a “working set”?

With fixed, independent access probabilities, best list order is non-increasing by probability.

We may well not know the probabilities, or they may not be fixed and/or independent

Self-adjusting list: rearrange after each access or update.

Our model: each swap of two adjacent items costs one.

For simplicity, we ignore insertions and deletions (results extend to cover them)

Update rules

Frequency count(FC): For each item, maintain a count of accesses, keep items in non-increasing order by frequency. After an access of *x*, increment the count of *x* and move it forward past items with a smaller count.

Move-to-front(MF): Move an accessed item all the way to the front.

Transpose(TR): Swap an accessed item with its predecessor

List: *A, N, X, D, K, U* **Access**: *K, X, U, N, X*

MF:	*K, A, N, X, D, U*	9	**TR**: *A, N, X, K, D, U*	6
	X, K, A, N, D, U	7	*A, X, N, K, D, U*	4
	U, X, K, A, N, D	11	*A, X, N, K, U, D*	7
	N, U, X, K, A, D	9	*A, N, X, K, U, D*	4
	X, N, U, K, A, D	5	*A, X, N, K, U, D*	4

Each rule moves an accessed item zero or more positions forward in the list. If item k from the front is accessed, cost of FC is between k and $2k – 1$ (inclusive), cost of MF is $2k – 1$, cost of TR is k if $k = 1$, otherwise $k + 1$.

If the number of accesses is large and the items have fixed, independent access probabilities, FC asymptotically minimizes the total access cost. So does TR, which needs to maintain no frequency counts or other extra information.

In practice: TR is terrible and MF often beats FC.

Typical access sequences display locality of reference not exploited by FC or TR; the fixed, independent probability model is far from accurate.

Needed: a more accurate, more realistic, more robust model.

Competitive Analysis

MF is 4-competitive with the optimum off-line algorithm.

That is, on *any* access sequence, its cost is no more than 4 times the cost of *any* algorithm on the given sequence.

(Assumptions: initial list order is the same for MF and the adversary, each swap costs one, no other rearrangement is possible.)

Proof

Given an arbitrary access sequence, we run MF and OPT, a minimum-cost algorithm, on the sequence concurrently. We define a potential Φ that measures the difference between the current list orders of MF and OPT. Φ is twice the number of *inversions*, pairs of items whose order is different in the two lists.

Φ = 0 initially; $0 \le \Phi \le n(n-1)$ always.

We run the algorithms concurrently as follows:

If *x* is the next item to be accessed, both algorithms (1) access *x* without doing any swaps, then (2) MF does its swaps, then (3) OPT does its swaps.

We compare the amortized cost of MF for (1) and (2) to the actual cost of OPT for (1). We compare the amortized cost of MF for (3) with the actual cost of OPT for (3).

During (2), OPT has no actual cost; during (3) MF has no actual cost. But Φ can change.

Consider access of *x*. Let *x* be *k* from the front in the MF list and *j* from the front in the OPT list.

MF actual cost of (1) and (2) (MF access and swaps) is $2k - 1$. OPT actual cost of (1) and (2) (OPT access but not swaps) is *j*.

Moving *x* to the front of MF's list creates or destroys one inversion for each item *y* previously in front of *x*. To create an inversion, *y* must precede *x* in OPT list.

$1, 2, 3, \ldots\ldots, k - 1, k, \ldots$ MF before

$k, 1, 2, 3, \ldots, k - 2, k - 1, \ldots$ MF after

(items identified by position in MF list)

$x_1, x_2, \ldots, x_{j-1}, k, \ldots$ OPT before

One inversion created or destroyed for each i, $0 < i < k$.

One inversion created if i among $x_1, \ldots, x_{j-1}$:

at most $j - 1$.

One inversion destroyed if i not among $x_1, \ldots, x_{j-1}$:

at least $k - 1 - (j - 1) = k - j$.

MF amortized cost of (1) and (2) ≤

$2k - 1$ (actual cost) $+ 2(j - 1) - 2(k - j) = 4j - 3 < 4j$.

MF amortized cost of (1) and (2) $\leq 4j$.

(3) Swaps by OPT: each swap costs OPT 1. The MF amortized cost of a swap is either 2 or –2, depending on whether it creates or destroys an inversion. If OPT does s swaps, the MF amortized cost of these swaps is $\leq 2s < 4s$.

Thus the MF amortized cost of (1), (2), and (3) $\leq$ $4j + 4s$ = 4 times the OPT actual cost of (1), (2), and (3).

Almost done! Sum over all accesses:

MF total actual cost

$\leq$ MF total amortized cost ($\Phi_0 = 0$, $\Phi_{\text{final}} \geq 0$)

$\leq$ 4(OPT total actual cost).

If initial list orders differ, must add $n(n-1)$ startup cost:

MF total cost $\leq$ 4(OPT total cost) + $n(n - 1)$

(why?)

Different cost models give different results, some better, some worse:

if swapping any pair of items, adjacent or not, costs 1, MF is only *n*-competitive. (Why?)