6.046 Recitation 7: Competitive Analysis Bill Thies, Fall 2004

Acknowledgements:

Parts of these notes were borrowed from George Savvides.

My contact information:

Bill Thies

thies@mit.edu

Office hours: Sat 1-3pm, 36-153

Recitation website: http://cag.lcs.mit.edu/~thies/6.046/

Announcements:

- PS 4 due Mon
- PS 5 out Mon, due Mon 11/1

Today:

- Competitive analysis
- Self-organizing lists

Definitions

Online Algorithm – An algorithm that must process each query in turn, without detailed knowledge of future queries.

Competitive: - An online algorithm A is α-competitive if there exists a constant k such that for any sequence S of operations:

$$C_A(S) \leq \alpha \cdot C_{OPT}(S) + k$$

Where $C_A(S)$ = cost for algorithm A
$C_{OPT}(S)$ = cost for optimal offline algorithm

<< note that we might not know what OPT looks like, or what it does. But we can use some properties of OPT, like a lower bound, to show that A is competitive. >>

Self-Organized Lists

Problem:

- Support Search(x) in linked list
- Given Sequence S of queries, minimize **total cost**
- Can re-arrange list after each query

Cost-model:

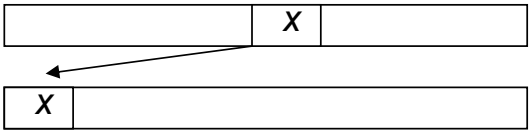
- Search(x) costs Rank(x): the distance of x from the head of list
- Reordering by transpose of adjacent elements.
  Cost of each transpose = 1
- So Cost = Rank(x) + # transposes

<< note that you can have more sophisticated cost model to give a tighter competitive ratio>>

Move-to-Front (MTF)

Move-to-Front: After Search(x), move x to head of list

Can show: MTF is 2-competitive
<< see Sleator/Tarjan paper >>
We will show: MTF is 4-competitive
<< simplifies cost model and analysis >>



Running Example:

		MTF		**OPT**		
i	x	L_i	c_i	L_i^*	c_i^*	
0		ABC		ABC		
1	A	ABC	1	BAC	1+1 = 2	←
2	B	BAC	3	BAC	1	$c_i < c_i^*$
3	C	CBA	5	BAC	3	
4	A	ACB	5	BAC	2	
5	B	BAC	5	BAC	1	← $c_i > 4\ c_i^*$
6	B	BAC	1	BAC	1	
		Total:	20	Total:	10	

The sum comes out within a factor of 2. But how to prove that this is always true? It is non-trivial. Would like to show that c_i is within a factor of 4 of c_i^* for every operation. But this is not true!

Consider i=5. At this point, c_i=5 while c_i^* = 1. Why is the cost higher? The intuition is that we are paying for something "good" besides just returning element B. We are making the MTF list "more similar" to the OPT list. This allows the query for B when i=6 to have a very low cost (=1) in MTF.

Similarly, when i=1, c_i = 1 while c_i^* = 2. Why is the cost lower? The intuition is that MTF is "borrowing from the future" by not maintaining the list to be the same as OPT. MTF will pay for this on a future iteration, since B is further down in the list.

How to formalize this intuition? Use a potential function to capture the amount of work you have invested in the current list.

Potential Function

$\Phi(D)$ = "amount of work invested in data structure D"

$\Phi(D_0) = 0$

$\Phi(D_i) \geq 0$

<< like a bank account >>

Can define the amortized costs in terms of Φ:

$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Where $\hat{c}_i$ = amortized cost of operation i

c_i = true cost of operation i

$\Phi(D_i) - \Phi(D_{i-1}) = \Delta \Phi_i$ = "potential difference"

Intuition:

if $\Delta \Phi_i > 0$, then $\hat{c}_i > c_i$. Storing work in data structure.

if $\Delta \Phi_i < 0$, then $c_i < \hat{c}_i$. Using old work to help pay for c_i.

Potential Function for Move-to-Front

Would like to capture "similarity" of the list for MTF (L_i) and the list for OPT (L_i^*)

$\Phi(L_i)$ = 2 * (minimum # transposes to convert L_i to L_i^*)
= 2 * (# inversions of L_i with respect to L_i^*)
= 2 * | { (x,y): x precedes y in L_i and y precedes x in L_i^* } |

Let's fill in running example to look at the inversions, $\Phi(L_i)$, $\Delta\Phi(L_i)$, and $\hat{c}_i$:

			MTF			**OPT**				
i		x	L_i	c_i	$\hat{c}_i$	L_i^*	c_i^*	**inversions**	**$\Phi(L_i)$**	**$\Delta\Phi(L_i)$**
0			ABC			ABC		**{}**	**0**	**0**
→ 1		A	ABC	1	**3**	BAC	2	**{(A,B)}**	**2**	**2**
2		B	BAC	3	**1**	BAC	1	**{}**	**0**	**-2**
3		C	CBA	5	**9**	BAC	3	**{(C,B), (B,A)}**	**4**	**4**
4		A	ACB	5	**5**	BAC	2	**{(A,B), (C,B)}**	**4**	**0**
→ 5		B	BAC	5	**1**	BAC	1	**{}**	**0**	**-4**
6		B	BAC	1	**1**	BAC	1	**{}**	**0**	**0**
TOTALS				20	**20**		10			

Let's re-examine our problem cases (i=1, i=5) in terms of the amortized cost, $\hat{c}_i$. When i=5, our potential function captured the fact that MTF is making its list more similar to OPT, thereby achieving an amortized cost of 1 even though the true cost was 5. The converse is true for i=1.

Examining each of the $\hat{c}_i$, it is now apparent that they are all within a factor of 4 of the true cost of OPT, c_i^*. Note also that the sum of the amortized costs is the same as the sum of the true costs. This is true because the potential was zero at the beginning and end of our operations. What happens if we stopped when we had positive potential? Then the sum of the amortized costs would be greater than the sum of the true costs. Thus, the sum of amortized costs is always $\geq$ than the sum of true costs.

<< this is formalized at end of notes, not covered in recitation >>

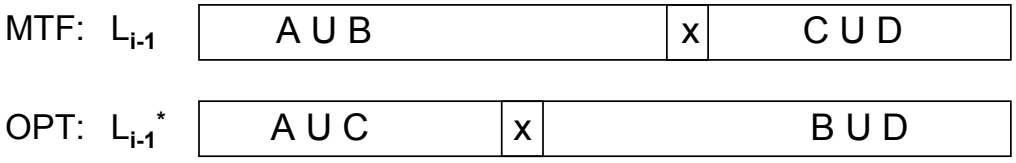
So we showed this for a single example, now let's consider the general case.

General Case

Let's consider what happens during the i'th query. We can divide the elements in the list into 4 non-overlapping sets:

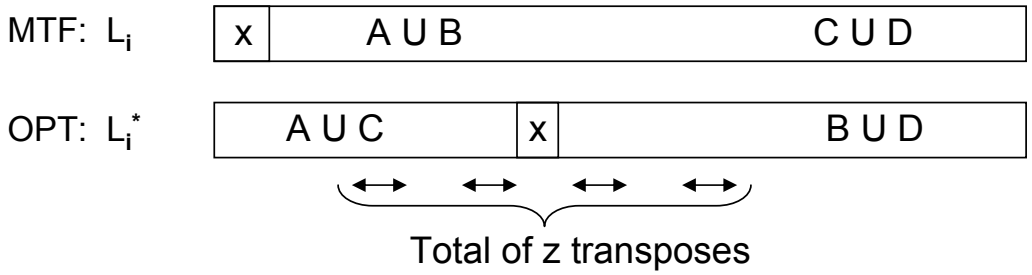
A: elements that are before x in L_{i-1}, before x in L_{i-1}^*
B: elements that are before x in L_{i-1}, after x in L_{i-1}^*
C: elements that are after x in L_{i-1}, before x in L_{i-1}^*
D: elements that are after x in L_{i-1}, after x in L_{i-1}^*

Illustration:



Consider i'th operation:
- MTF moves x to front of L_i
- OPT performs z transposes

The result looks like this:



Now for analysis: $\Delta \Phi_i = \Phi(L_i) - \Phi(L_{i-1})$

= 2 * (# inversions created - # inversions destroyed)

think about picture → = 2 * (|A| - |B| + # inversions created by OPT)

$\leq$ 2 * (|A| - |B| + z)

Bounding the Amortized Cost

$$
\begin{aligned}
\hat{c}_i &= c_i + \Delta\Phi_i && \text{[definition of cost w/ potential func.]} \\
&\leq 2 * (|A| + |B| + 1) + \Delta\,\Phi_i && [c_i = \text{rank}(x) + \#\text{ transposes.} \\
& && \text{rank}(x) = |A| + |B| + 1. \\
& && \#\text{ transposes} < \text{rank}(x)\ ] \\
&\leq 2 * (|A| + |B| + 1) + 2 * (|A| - |B| + z) && \text{[from last page]} \\
&= 4\,|A| + 2 + 2z \\
&\leq 4\,|A| + 4 + 4z \\
&= 4\,(\,|A| + 1 + z) \\
&\leq 4\,(\text{Rank}_{OPT}(x) + z) && [\text{Rank}_{OPT}(x) = |A| + |C| + 1] \\
&= 4\,c_i^* && [\text{cost} = \text{Rank}_{OPT}(x) + \underbrace{\#\text{ transposes}}_{z}\ ]
\end{aligned}
$$

Thus, we have shown $\hat{c}_i \leq 4\,c_i^*$. That is, the amortized cost of each operation in MTF is within a factor of 4 of the true cost for that operation in OPT.

Proving the Competitive Bound

We did not carry out this summation formally in class. We made the argument orally (see page 6).

We want to show that $C_{MTF}(S) \leq 4 * C_{OPT}(S)$ for any sequence of queries S.

$$
\begin{aligned}
C_{MTF}(S) &= \sum_{i=1}^{|S|} c_i \\
&= \sum_{i=1}^{|S|} [\, \hat{c}_i + \Phi(L_{i-1}) - \Phi(L_i) \,] && [\text{since } c_i = \hat{c}_i - \Delta\Phi_i] \\
&\leq \sum_{i=1}^{|S|} 4c_i^* + \Phi(L_0) - \Phi(L_{|S|}) \\
&\leq \sum_{i=1}^{|S|} 4c_i^* && [\text{since } \Phi(L_0) \leq \Phi(L_{|S|})\,] \\
&= 4 \sum_{i=0}^{|S|} c_i^* \\
&= 4\, C_{OPT}(S)
\end{aligned}
$$

This shows that for any sequence of S operations in MTF, the total of the true costs is no more than 4 times the total true costs in OPT.

It is possible to obtain a competitive ratio of 2 using a tighter cost model (in which it is constant cost to move an element to the front of the list.) See the Sleator / Tarjan handout for details.