

Revision: on-line algorithms

- For some problems, we want an algorithm that makes some decisions before the input is completely known to the algorithm.
- Such algorithm typically cannot perform as good as an algorithm that does have the complete input.
- Instead of optimality, we want the algorithm to perform competitively. That is, if an optimal off-line algorithm need a cost of c , we want our algorithm to guarantee that it needs at most $c \cdot \rho$ for some ρ .
- Such algorithms usually use the ski-principle when deciding whether to make an expensive move.
- We have studied the ski-rental problem, the unbounded-search problem and the cache-replacement problem that need such “competitive analysis”.

ALG/TITCS L10-1

Comparing on-line and off-line status

- In many time, the behaviour of an algorithm depends on the “current configuration” of an algorithm. For example, the behaviour of any paging algorithm depends on what is currently in the cache.
- The adversary is usually much better off if the configuration of your algorithm differs a lot from the configuration of the best off-line algorithm so far.
- To analyze such algorithms, we usually need a way to capture the difference of the two algorithms, and to argue that the difference is not “too much too often”.
- One technique to bound such differences is the potential method.

ALG/TITCS L10-2

The potential method

- We find a positive “potential function” $\Phi(c_1, c_2)$ that, given two configurations c_1 and c_2 , captures how different they are.
- At the beginning, the on-line algorithm and the off-line algorithm have the same configuration, resulting in the lowest possible value of Φ (usually 0).
- If a step of our algorithm causes the difference to increase, we “pre-pay” some cost according to the increase in potential function.
- Later, if a step of our algorithm is expensive *since it differs a lot from the off-line algorithm*, we can then “use the pre-paid amount” to pay for the cost of our algorithm.
- In this way, we basically charge the cost of the expensive step against the previous steps that causes the difference to increase.

ALG/TITCS L10-3

The List Accessing Problem

- Suppose we have a bookshelf containing a fixed number of books.
- Every time somebody searches for a book, he start from the beginning bookshelf and look at each book until he find the right one.
- Then he will pick out the book, do whatever he want with it, and put the book back to the bookshelf.
- Since he start from the beginning of the shelf, it is easier to search for books that are close to the beginning of the shelf.
- Let’s say that searching for a book at the l -th position of the shelf costs l .

ALG/TITCS L10-4

The need of reorganization

- Clearly, if we know *a priori* what books will be accessed the most frequently, we will put those books at the beginning of the shelf.
- But if we don’t, every organization seems to be equally bad: we can always keep on requesting the last book on the shelf, and the algorithm performs poorly.
- Can we do better for this case? Yes... we can move the book earlier.
- We are allowed to move a book after it is accessed, from its original position to any location that is before it.
- Such operations are “free”. The rationale is that we can always leave some trace during the search, so no new search is needed.

ALG/TITCS L10-5

Reorganization algorithms

- Now comes the problem: after each time we access a book, we can reorganize the shelf by putting the book back to a different position.
- But where to put it to? It does not really seem that putting at a particular position is really good, since we don’t know what books will be accessed again.
- But there is something good to actually move the book closer to the beginning of the shelf: an adversary trying to give the algorithm a really bad input would have to pick other books, and by doing that would cost a lot for even the off-line algorithm.
- Remember that competitive analysis is all about bounding the ratio between the on-line algorithm and the off-line algorithm. Making sure that the off-line is bad is just as good as making sure that the on-line is good.

ALG/TITCS L10-6

What we really compare with

There is a question: is the off-line algorithm allowed to make reconfigurations?

When we perform competitive analysis, we give the on-line algorithm and the off-line algorithm the same power—except that the on-line algorithm don't know the input in advance.

That means a competitive algorithm have to guarantee that its cost is as good as any other off-line reorganization algorithms.

Thus the answer is, **yes**, the off-line algorithm can also perform reorganizations. Actually this gives the off-line algorithm some power, as illustrated in the following exercise.

Exercise: Find an input that the off-line algorithm can perform better if it is allowed to reorganize the items, rather than just to have a fixed organization according to the input.

A reorganization algorithm MTF (Move-To-Front)

Here is a very simple reorganization algorithm called Move-To-Front:

Whenever a book is put back to the shelf, put it back to the first position of the shelf.

Now, notice that the "bad case" (always search the book originally at the end) suddenly becomes the "best case" for our algorithm.

But is the algorithm too drastic? Once a book is searched, we put it to the beginning, ignoring any previous searches. The answer is **no**: actually the algorithm is 2-competitive.

ALG/TITCS L10-7

ALG/TITCS L10-8

Comparing MTF with an off-line algorithm

- It does not seem to be easy to compare MTF to another algorithm.
- The problem is that the cost of any algorithm depend on the ordering of the books currently on the shelf.
- If at some time, MTF has a book at the end of the shelf and an off-line algorithm has that book at the beginning, the off-line algorithm can perform much better than MTF.
- *But is it really possible that this always happens?*

ALG/TITCS L10-9

A potential function for the list-access problem

We thus want to capture how much two orderings of books are different. We use the following potential function:

$$\Phi(c_1, c_2) = \text{The number of inversions between } c_1 \text{ and } c_2.$$

An inversion is a set of two elements $\{a, b\}$ such that a is before b in one list but after b in the other.

For example, the list (1, 3, 2, 4) and (1, 4, 3, 2) has the following inversions:

ALG/TITCS L10-10

Analyzing MTF

Now we are ready to show that MTF is 2-competitive. By 2-competitive, we mean that for any input, for any other algorithm, we need no more than twice the cost of that other algorithm.

We want to show that summing all the steps, the cost of our searches of MTF is at most twice the cost of searches of that algorithm.

But we cannot show that for each step, the cost of search for MTF is at most twice of the cost of any other algorithm: we have seen that it's simply not true.

Instead, we show that for each step, the cost of search for MTF is at most twice of the cost of any other algorithm minus the change in potential regarding the two algorithms.

That is, when the potential increases, our cost is small enough to "make some savings" in the potential.

ALG/TITCS L10-11

Using potential function

Let us denote the configurations of MTF and an optimal off-line algorithm as c_0, c_1, \dots, c_k and c'_0, c'_1, \dots, c'_k after the 0th, 1st, 2nd, \dots , k -th searches.

- Let the cost of the i -th search for MTF be $C_m(i)$, and the cost of the i -th search of the optimal off-line algorithm is $C_o(i)$.

- We want to show that

$$C_m(i) \leq 2C_o(i) - \Phi(c_i, c'_i) + \Phi(c_{i-1}, c'_{i-1}).$$

- Summing all i , we then have

$$C_m(1) + C_m(2) + \dots + C_m(k) \leq 2(C_o(1) + \dots + C_o(k)) - \Phi(c_k, c'_k) + \Phi(c_0, c'_0).$$

- This proves the ratio bound, since $\Phi(c_0, c'_0) = 0$ and $\Phi(c_k, c'_k) \geq 0$.

ALG/TITCS L10-12

The essence of the potential method

Let's see what is the ingredients of the method.

- Find a good potential function that describe how bad is the current configuration (comparing to a particular algorithm).
- Argue that the cost of each step is a certain multiple of the optimal deducting the change in potential.
- The potential "buffer out" the variance of the badness of the configuration. We make some saving during good times to pay for the bad times.
- The potential method is really meant for researchers: good potential functions are usually **very difficult** to come with, although it results in something **really neat**.
- But you should at least be able to read a proof that uses potential.

ALG/TITCS L10-13

The way we count

Now let's bound the performance. In step i , we count the cost of MTF as follows:

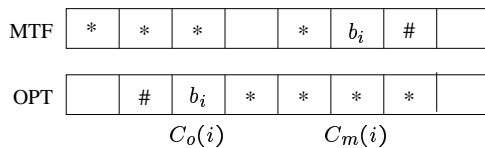
- We receive a request b_i . We first let MTF to deal with it.
- It is found to be at some position of the list. This costs $C_m(i)$ if it is at the position $C_m(i)$.
- MTF then bring the book to the first place. This does not cost us anything. But it changes the potential. We find the change.
- Now we let the optimal algorithm OPT to deal with the request. It requires a cost of $C_o(i)$ if it is found to be at the $C_o(i)$ -th position.
- OPT reorganizes the books. We bound on the change of potential.
- We argue that $C_m(i)$ is at most $2C_o(i)$ subtracting the total potential change.

ALG/TITCS L10-14

Changing potential

At the time when a request b_i is serviced, only inversions concerning b_i can be changed (since moving b_i does not change the ordering of other books).

Consider the inversions involving b_i (* and # shows inversions of different sides):



Suppose we have v_i '*s in the figure. Then MTF removes v_i inversions and creates $C_m(i) - v_i - 1$ new ones.

After that, OPT can only remove inversions, since b_i is already at the beginning for MTF.

ALG/TITCS L10-15

Bounding potential change

Now we can bound the change of potential:

- MTF changes the potential by exactly $C_m(i) - v_i - 1 - v_i$. However, note that $C_m(i) \leq C_o(i) + v_i$, so the change of potential is at most $2C_o(i) - C_m(i) - 1$.
- OPT changes the potential by at most 0.

Thus $\Phi(c_i, c'_i) - \Phi(c_{i-1}, c'_{i-1}) \leq 2C_o(i) - C_m(i) - 1$. Now we need to add the cost of the search of MTF, which is $C_m(i)$:

$$C_m(i) \leq 2C_o(i) - \Phi(c_i, c'_i) + \Phi(c_{i-1}, c'_{i-1})$$

as desired.

ALG/TITCS L10-16

Competitive analysis: critics

- Whenever we have a new way to judge algorithms, we are always confronted with critics. Competitive analysis is no exception.
- The primary problem about competitive analysis is that it compares the on-line algorithm with an off-line algorithm that has all the information.
- That we lose is no surprise. But for some problems, the performance of the algorithm is very sensitive to a few steps in the input. Then we lose big.
- For these problems, competitive analysis typically gives a very pessimistic idea about how good can algorithms be.
- For example, for the cache replacement problem, all FIFO, LRU and FWF are k -competitive. Two problems: k is bad, and everybody know LRU is in fact better.

ALG/TITCS L10-17

Remedy 1

At some times, the real problem is that our definition does not capture what we want to see.

In the cache replacement example, it does not capture locality of reference at all, which is essential for the algorithm to perform well.

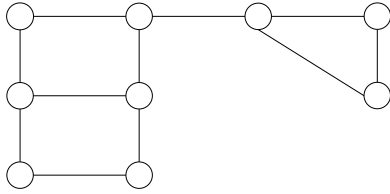
Remedies for this type of problems must address the lacking feature of our model. For example, the access graph model tries to address this problem.

Basically, this restrict the adversary by saying that "we only consider requests that are consistent to a graph G ."

ALG/TITCS L10-18

Access graph

An access graph is a graph that specifies exactly what is a possible request sequence and what is not.



Of course, the graph must not be really highly connected if it is to be useful. In fact, our previous discussion showed that an access graph that is a cycle of length k already causes LRU to be k -competitive.

But it for graphs that are not as highly connected, LRU does perform quite good. E.g. for an access graph that is a list, LRU is actually optimal.

ALG/TITCS L10-19

Remedy 2

- For some problems, the sensitivity of the input makes the competitive ratio really bad.
- Sometimes we can improve the competitive ratio a lot if we compare the algorithm against off-line algorithms that have less resources.
- For example, if we compare FWF with an off-line algorithm that only possess half the cache, then instead of an unbounded ratio bound of k , we have a ratio bound of 2. (Note the big effect of a factor-of-2 advantage of the amount of cache!)
- In another angle, we can say that such studies consider *those requests sequence that leads to drastically different performance when given slightly different amount of cache are not within consideration.*

ALG/TITCS L10-20

Epilog

All the materials in the lecture today can be found in the following book

Online computation and competitive analysis, Allan Borodin, Ran El-Yaniv.
Cambridge University Press, 1998

Another book is also of interest, and covers many ideas in our lectures:

Online algorithms : the state of the art, Amos Fiat, Gerhard J. Woeginger
Springer, 1998.

They are now 1-day reserved material in the library. If you're interested, borrow one of the books and see what has been put to competitive analysis.

ALG/TITCS L10-21