Paging Problem

(special case of the k-server problem)

1 Introduction

The k-server problem is the following on-line problem. You are given a metric space and k "servers". At every time step a point of the metric space is "highlighted", which must be covered by one of the servers. The goal is to make as few moves with the servers as possible.

We are not going to investigate the fully general problem, we will restrict ourself to a uniform metric, i.e., $d(i, j) = 1$ if $i \neq j$ and $d(i, i) = 0$ for all i. This problem is also known as the "paging problem".

This is an on-line problem. We are going to look at randomized algorithms versus an oblivious adversary (i.e., the adversary does not know the random choices of the algorithm ahead of time).

2 Upper bound

We present the Marker algorithm (due to Fiat, Karp, Luby, McGeoch, Sleator and Young, 1991), because it has a "proof from the book", and it has a competitive ratio that is only a factor 2 away from the best possible.

2.1 Algorithm

We start of with the servers in any position. Servers can be marked or unmarked; at the start of the algorithm all servers are unmarked. The algorithm is as follows: a request comes in; if there is a server on that point, the server is marked; otherwise, if there is no unmarked server, then all servers are unmarked; now we are sure that there exists an unmarked server, one of them is chosen uniformly at random (and independently) and moved to that point; the server is then marked.

2.2 Analysis

We split the requests into "phases", such that every phase has exactly k distinct points and the first point of the next phase is not equal to any of these. We let R_i denote the set of distinct points in phase i (so $|R_i| = k$ for all i), and we let O_i denote the set of points on which the optimal off line algorithm has its markers at the start of phase i (so also $|O_i| = k$ for all i).

First we will lower bound the number of moves in phase i of the optimal algorithm: obviously the number of moves is at least $|R_i \setminus O_i|$, as the points in this set have to be covered by servers. But the number of moves is also lower bounded by $|R_i \setminus O_{i+1}|$, as the points in this set are vacated in phase i by servers in the optimal algorithm, so the servers have moved elsewhere. Note that $|R_i \setminus O_{i+1}| = |R_i| - |R_i \cap O_{i+1}| = |O_{i+1}| - |R_i \cap O_{i+1}| = |O_{i+1} \setminus R_i|$.

We take the average of these two lower bounds to get

$$\text{OPT}_i \geq \frac{|R_i \setminus O_i| + |O_{i+1} \setminus R_i|}{2}$$

as a bound on the cost for the optimal algorithm incurred in the ith phase.

Summing over all phases yields:

$$\text{OPT} = \sum_i \text{OPT}_i \geq \sum_i \frac{|R_i \setminus O_i| + |O_{i+1} \setminus R_i|}{2}.$$

We want these sums to telescope, so we will massage the summand somewhat.

Note that $|A \setminus B| \geq |A \setminus (B \cup C)| = |(A \setminus C) \setminus (B \setminus C)| \geq |A \setminus C| - |B \setminus C|$. Using $A = R_i$, $B = O_i$ and $C = R_{i-1}$, we get

$$\text{OPT} = \sum_i \text{OPT}_i \geq \sum_i \frac{|R_i \setminus R_{i-1}| - |O_i \setminus R_{i-1}| + |O_{i+1} \setminus R_i|}{2} \geq \frac{1}{2} \sum_i |R_i \setminus R_{i-1}| - k.$$

Now, let's analyze the performance of the algorithm. In every phase i it needs to move servers to every point in $R_i \setminus R_{i-1}$. The points in $R_i \cap R_{i-1}$ may or may not have a server on them when the are requested. Let's upper bound the probability that a point in $R_i \cap R_{i-1}$ does not have a server on it when it is requested. The worst thing that can happen is that all the points in $R_i \setminus R_{i-1}$ are requested before any of the points that have a server on them. The first point that is requested in $R_i \cap R_{i-1}$ does not have a server on it with probability at most the probability that one of the points in $R_i \setminus R_{i-1}$ got its server, i.e. at most $|R_i \setminus R_{i-1}|/k$. After this request, we now know that there is a marked server on this point, and $|R_i \setminus R_{i-1}|$ more points. We will now imagine that this point has the same server as at the start of the phase maybe by swapping two servers. The next distinct point that is requested in $R_i \cap R_{i-1}$ does not have a server on it with probability at most the probability that its server is now on one of the points in $R_i \setminus R_{i-1}$, i.e., probability at most $|R_i \setminus R_{i-1}|/(k-1)$. In general, the $(j+1)$st distinct point that is requested in $R_i \cap R_{i-1}$ does not have a server on it with probability at most the probability that its server is now on one of the points in $R_i \setminus R_{i-1}$ (after swapping), i.e., probability at most $|R_i \setminus R_{i-1}|/(k-j)$.

So we get an upperbound on the number of moves by the algorithm in phase i for the points in $R_i \cap R_{i-1}$ of $\sum_{j=0}^{|R_i \cap R_{i-1}|-1} |R_i \setminus R_{i-1}|/(k-j) = \sum_{j=|R_i \setminus R_{i-1}|+1}^{k} \frac{1}{j} |R_i \setminus R_{i-1}| \leq (H_k - 1)|R_i \setminus R_{i-1}|$, where $H_k = \sum_j 1/k$, because $|R_i \setminus R_{i-1}| \geq 1$ by the definition of phases. Adding one move for each point in $R_i \setminus R_{i-1}$ gives an upper bound of $H_k |R_i \setminus R_{i-1}|$.

3 Lower bound

We will now show that any randomized algorithm can be at best H_k-competitive.

We will show a lower bound where the number of points is $k+1$. We define the following random request sequence: at every time step a point is chosen uniformly at random from all points.

For any algorithm that cannot look ahead, the probability that this point does not have a server on it is $1/(k+1)$, namely the probability that it is not equal to the points that have a server on them, of which there are k. (Note that the point is chosen uniformly at random.)

The offline optimum does the following: it divides the input sequence into phases as described in the previous section. At the start of each phase the server is moved from the node at the start of the next phase to the point at the start of this phase, i.e., the optimal algorithm has a cost of 1 for each phase. (All other points will be requested in this phase.)

The expected cost of any algorithm that cannot look ahead is $1/(k+1)$ times the expected length of a phase (by Wald's equation). The expected length of a phase is $(k+1)H_{k+1} - 1$, from the coupon collector's problem, where we don't count the last "coupon". So the expected cost in a phase is $H_{k+1} - 1/(k+1) = H_k$.