CSC2421: Online and Other Myopic Algorithms
Fall 2025

Allan Borodin

November 12, 2025

Week 10

Annoucements

- I want to try to schedule some project talks for next week so please volunteer if you think you can be ready to give a high level view of the project next week. The written project does not have to be submitted until the last day of classes (December 2). Projects not presented next Wednesday, the 19th will be presented on the 26th. Feedback from the presentatations should help the written project.

  So far, I have two voluteers for presenting next Wednesday, November 19. Namely, David and Nadim will talk about the k-taxi problem, and Tristan and Raghav will talk about perpetual voting. Is there another volunteer?

  In addition to these two projects, I think Michael will talk about throughput scheduling, Jacob and Reina will discusss various aspects of online graph colouring and Jiaqi and Yifan will discuss online algorithms in the context of reinforcement learning.

  Who has not told me about their project?

Extensions of three basic online problems

Chapter 8 discusses extensions to the ski rental, (one-dimensional) bin packing and k-server problems. The extensions to the ski rental problem arguably provide more meaningful applications than the basic ski rental problem. But the basic buy or rent paradigm in ski rental is apparent in all these applications.

One dimensional bin packing is already a problem with real applications as are all the given extensions.

The k-server problem is of great importance in terms of the concepts that evolved from the problem and its analysis. The given k-server extensions are mainly of theoretical interest and have no immediate applications although like the basic k-server problem, the extensions may provide a basis for new ideas in online analysis. There is also the possibility of a more applicable k-taxi problem than what is discussed in the text.

We will start with extensions to ski-rental that allow for multiple buying options.

The File Migration problem

In multiprocessor and distributed systems, there is a need for different processes to share data. As in standard paging/caching where a page consists say of many words of memory, we have to move data in blocks (e.g., files) while a process may only want a small amount of data within the block. We now consider moving blocks of data between nodes in an undirected edge-weighted network. Such problems are called distributed file or page management problems.

Given an edge-weighted network of processors, we will assume that when a data management system is responding to a request from node v_j (i.e., from a process executing on node v_j) for a piece of data in a file located at node v_ℓ, it will pay a cost $c(v_j, v_\ell)$, the min cost of a path between v_j and v_ℓ. If the data management system also wants to move an entire file from node v_ℓ to node v_j, it will pay a cost of $D \cdot c(v_j, v_\ell)$ where D is some large constant. The objective is to minimize the total cost of file accesses and file transfers.

File management problems

There are different distributed file managment problems. The most basic problem is what we have called the *file migration* problem. In this problem, there is one copy of a file and the management system has to respond to each request by first paying the cost of the access and then deciding if it wants to move the file. We note that the way this problem is formulated is that the algorithm has "0-lookahead" (as it is for expert learning) whereas the convention for online algorithms (i.e., as in the request-answer framework) is "1-lookahead" where the algorithm can make its decsion after seeing the request.

In the k-migration problem there are k copies of the file. We note that in the single file migration problem, we do not have to distinguish between read and write requests as each incurs the same cost. However, in the k-migration problem, a write request requires updating the k different copies.

A more general problem is the *file allocation* problem where copies of a file can be replicated or deleted in response to read and write requests.

Two file management considerations

In a dynamic network, the costs of communication between adjacent nodes can change and in the extreme a link might fail for some period of time.

One final consideration is whether or not the nodes (i.e., the processors) have unlimited or bounded memory. In the case of bounded memory, file allocation can be naturally viewed as an extension or modification of traditional paging/caching systems

We will only consider file migration in the setting of a static network and we will assume unbounded memory so as to focus on the communication aspect of data management. We will consider arbitrary networks, noting that as in any graph theoretic problem, one can often obtain improved competitive results for some specific networks.

One can view file migration as an extension of the ski rental problem where now a access is like a day of renting. Indeed for a network of two nodes the problem (deciding when to move a file) is equivalent to deciding when to buy in the ski rental problem (ignoring the issue of 0 lookahead).

A simple randomized algorithm

We begin with a simple 3-competitive randomized algorithm motivated by the randomized ski rental algorithm. We will see that no deterministic file migration algorithm can be better than 3-competitive whereas in the ski-rental problem we have an optimal 2-competitive algorithm. This inconsistency is reconciled by the difference between the 0-lookahead for file migration and 1-lookahead for ski rental. We need to extend ski rental to an arbitrary network with metric edge costs.

Consider the following randomized memoryless $CoinFlip_p$ algorithm: After a request at a node v^* , with probability p, if the server is not on v^*, migrate the file from its current location to node v^*. Otherwise, just access the required data. We define *CoinFlip* to be $CoinFlip_p$ for $p = \frac{1}{2D}$.

Theorem:
Algorithm *CoinFlip* is 3-competitive against an adaptive adversary. This is the best possible ratio for any randomized online algorithm *ALG* against an adaptive online adversary. It follows that no deterministic algorithm can be better than 3-competitive.

The lower bound against an adaptive adversaary

For the lower bound against an adaptive online adversary, it suffices to consider the simplest network, a single edge with edge cost $c(u, v) = 1$. The file migration cost is then $D \cdot c(u, v) = D$. Consider an input sequence of length n. The adversary knows at all times where the file is located and therefore the adversary can simply request the file at the node where the file is not currently located.

Let ALG be any online algorithm. The adversary knows the distributions that any ALG will use to service each request and therefore knows at what nodes each request will be made when the algorithm will migrate the file. More precisely, the adversary can simulate the algorithm on all possible instantiations of the random distributions and determine the expected number of requests at each of the nodes u and v.

Continuation of lower bound for *CoinfFlip* against an adaptive adversary

If $D > n/2$, the adversary will initially place the file at the node most often requested (in expectation) and never move the file again. The expected cost of the adversary is $D + n/2$ while the expected cost of ALG is at least $3n/2$ so that the asymptotic competitive ratio is at least 3.

If $D \leq n/2$, the adversary will always insure that the file is requested at the node that does not contain the file. Now the expected cost of the adversary is D while $ALGs$ expected cost is $D + n \geq 3D$ so that once again the competitive ratio is at least 3.

Analysis of the CoinFlip algorithm

The analysis uses a potential function argument. We have a game between the adversary and the algorithm. The adversary possibly moves the file, initates a request at a node v_r and then services this request. The algorithm services the request, and possibly moves the file.

Let v_c (respectively, v_a) denote the current location of the file (respectively, the current location of the adversary) just prior to a request for the file at node v_r. Noting that the *CoinFlip* (ALG) algorithm is independent of both v_c and v_a, we define the potential function $\Phi = 3D \cdot d(v_c, v_a)$.

We want to show that $\mathbb{E}[ALG_i + \Delta\Phi_i] \leq 3 \cdot \Delta OPT_i$ where ALG_i is the algorithms cost for the i^{th} request, $\Delta\Phi_i = \Phi_i - \Phi_{i-1}$ is the change in potential due to the i^{th} request and OPT_i is the cost of OPT for the i^{th} request.

Expected algorithm cost and possible change in potential

We bound the costs and the change in the potential function in terms of two events.

1. The algorithm serves a request.
   The expected cost to the algorithm is
   $\mathbb{E}[ALG_i] = d(v_r, v_c) + \frac{1}{2D} \cdot Dd(v_r, v_c) = \frac{3}{2} d(v_r, v_c)$.
   The potential for this event changes if and only if the algorithm migrates the file to the request v_r so that
   $\mathbb{E}[\Delta\Phi_i] = \frac{1}{2D}[3Dd(v_r, v_a) - 3Dd((v_c, v_a)]$. Then
   $\mathbb{E}[ALG_i + \Delta\Phi_i]] = \frac{3}{2}[d(v_r, v_c) + d(v_r, v_a) - d((v_c, v_a)] \leq$
   $\frac{3}{2}[d(v_r, v_a) + d(v_r, v_a)] = 3OPT_i$ using the trangle inequality.
2. The adversary moves the file.
   Suppose the adversary moves the file from v_a to v'_a. The algorithm ALG incurs no cost while $OPT_i = Dd(v_a, v'_a)$. The change in potential is
   $\Delta\Phi = 3Dd(d(v_c, v'_a) - 3Dd(v_c, v_a) \leq 3D((v'_a, v_a) = 3OPT_i$
   where again the inequality follows by the triangle inequality.

The lower bound for the CoinFlip algorithm

The *CoinFlip* algorithm is no better than 3-competitive against an oblivious adversary for any choice of the probability $p > 0$ for moving the file.

Consider the $CoinFlip_p$ algorithm where the probability of moving the file is p. We will use the fact that $\frac{1}{p}$ is the expectation of a random variable with probability p of success for each trial. Consider the single edge two node network with edge cost $c(u, v) = 1$. Assume that both the algorithm and the adversary start at node v.

Suppose $p \leq 1/2D$. Let the input be a sequence σ consisting of n requests by node u. The optimum cost for σ is $D + 1$. As $n \to \infty$, the probability $= 1$ that the algorithm eventually moves the file. In the limit, the expected cost for the algorithm is then $D + \frac{1}{p} \geq (D + 2D)$ where the $\frac{1}{p}$ is the cost of accesses until the file is moved. This implies that the lower bound converges to 3 as $n \to \infty$.

Finishing the lower bound argument for $Coinflip_p$

Suppose $p \geq 1/2D$. Let the input be a sequence σ consisting of a single request by node u followed by n requests by node v. The optimum cost for σ is 1. The expected cost (in the limit as $n \to \infty$) for the algorithm is $p \cdot (1 + D + 1/p + D) + (1 - p) \cdot 1 = 2 + 2p \cdot (2D)$ which is $\geq 3D$ for $p \geq \frac{1}{2D}$.

This again implies that the lower bound converges to 3.

Note that the competitive ratio is greater than 3 for any choice of $p \neq \frac{1}{2D}$.

The best known randomized algorithm for the file migration problem

We consider a randomized algorithm motivated by a Markov process. The *RandomReset* algorithm maintains a counter with value $C \geq 1$. When an access request is made to a file in node v, the counter is decremented by 1. If the counter reaches value 0, the page is transfered to the node v and the counter is randomly reset to value i with probability α_i where $\mathcal{D} = (\alpha_1, \alpha_2, \ldots, \alpha_k)$ is an appropriately chosen finite distribution and $k = \max_i\{i : \alpha_a > 0\}$.

For an appropriately distribution $\mathcal{D}$, the *RandomReset* algorithm obtains a competitive ratio $1 + \phi \approx 2.618$ ($\phi = \frac{1+\sqrt{5}}{2}$) in the limit as the file transfer factor D increases. Furthermore, there is an appropriate distribution $\mathcal{D}$ for which only α_{k-1} and α_k are non-zero.

Currently the best known lower bound is $2 + \frac{1}{D}$ for all $D \geq 1$. In fact, the lower bound already holds for a two node network. We note that the $2 + \frac{1}{D}$ ratio can be obtained for the special classes of trees, hyper-cubes, meshes, and uniform networks.

The best known deterministic algorithm for the file migration problem

Bartal et al [2001] provide the following *MoveToLocalMinimum* algorithm which acheives the current determinstic ratio 4.086 for file migfration: Partition the input sequence into phases of $\eta \cdot D$ requests for some appropriatley chose constant η. Suppose v_b is the node at the start of the phase. During a phase, the file remains at v_b. At the end of the phase, transfer the file to node $v^* = argmin_v[\sum_{i=1}^{D} d(v, v_i) + \delta D d(v_b, v)]$ for another appropriately chose constant δ, and then begin a new phase. The additional term $\delta D d(v_b, v)]$ helps to insure that the move to node v^* will not be too expensive.

The current best deterministic lower bound for file migration is 3.164 due to Matsubayashi and Kawamura [2008]

The Capital Investment problem

There are different variations for this problem. In its generality, the problem is defined as follows: There is a sequence of buying options $\{O_1, O_2, \ldots\}$ where $O_i = (t_i, b_i, r_i)$ and t_i (respectively, b_i, r_i) is the monotonically non-decreasing time step in which the i^{th} option becomes available (respectively, the buying cost of the i^{th} option, the rental or production cost of the i^{th} option.

The motivation is that of a company that wishes to produce some fixed number of goods each day $i = 1, 2, \ldots$ at a cost of r_i using some machine available before or at time t_i and has been bought.

We will assume that at least one option is available at time 1. We will purchase one of the available machines at time $i = 1$ For $i = 2, \ldots,$ an algorithm can choose to continue renting with any available purchased machine or rent with the last machine purchased.

In the capital investments problem we study in section 8.1.1 of the text, we assume that $r_j < r_k$ implies $b_j \geq b_k$.

Other variants

There are a number of variants of the problem we are calling Capital Investement. See Azar et al [1999] and Lotker, et al [2012].

- Time can be continuous instead of discrete.
- We can let the purchasing cost be an incremental cost (i.e trading-in for a new machine). In this case we would just assume that the cost to buy machine M_j is $b_j - b_{j-1} > 0$ and $r_j < r_{j-1}$.

The classic ski rental problem starts with one option at day 1 and there are no new options.

In the problem we call multi-slope ski rental, time is continuous and we have all options initially available. The muti-slope ski rental is also studied for incremental buying costs.

The Capital Investment algorithm

Theorem
There is a $1 + (\alpha + \frac{1}{\beta})$-competitive deterministic algorithm for the capital investments problem.
The algorithm has two parameters α and β which we will set later.

The algorithm starts by buying a machine thar minimizes $b_j + r_j$ amongst all machines availabe at time $i = 1$. We can call this option $O_1 = (b_1, r_1)$. For the puposes of the analysis we will refer to $b_1 + r_1$ the production costs for day 1. After day 1, we will keep track of the buying and production costs separately.

We have two conditions for buying a new machinei M_j at some time i. (Of course, M_j must be available at time i.) These conditions define the algorithm.

- After buying M_j at some time i, we must spend at least βb_j on production costs before being able to purchase another machine.
- The buying cost b_j is at most α times the total production costs thus far.

The Captial Investments algorithm continued

If there are several machines at time time i that meet these conditions, then we choose one with the smallest production cost. If no such option exists we stay with the current machine.
Here follows the pseudo code:

Algorithm 8.1.1 Deterministic algorithm for Convex Capital Investment problem.

procedure $ConvexCapitalInvestment(\alpha, \beta)$
$M_1 \leftarrow$ the machine that minimizes $b_j + r_j$ for machines that are available at time $i = 1$
▷ We can assume that at least one machine is available initially.
$k \leftarrow 1$ ▷ k will be the index of the current machine M_k being used.
$R_0 \leftarrow 0$ ▷ R_i will be the amount spent on production costs up to and including day i.
$R_1 \leftarrow r_1 + b_1$ ▷ For the purpose of the proof, we include the buying cost b_1 in the production cost for day 1.
$\tilde{i} \leftarrow 1$ ▷ $\tilde{i}$ will be the time step when the current machine M was bought.
$i \leftarrow 2$
while there is a demand for production on day i **do**
if $R_{i-1} - R_{\tilde{i}-1} \geq \beta \cdot b_k$ and there exists an available machine M_j on day i (i.e., $t_j \leq i$) such that $b_j \leq \alpha \cdot R_{i-1}$ **then**
$k \leftarrow \arg\min_j [r_j | b_j \leq \alpha \cdot R_{i-1}]$ ▷ We will now be using the machine M_k at a higher buying cost but lower rental cost.
$\tilde{i} \leftarrow i$
$R_i = R_{i-1} + r_k$
$C_i = C_{i-1} + b_k$
else
$R_i = R_{i-1} + r_k$ ▷ We stay with the current machine.

Sketch of analysis for Captial Investments algorithm

Repeating the theorem, we want to show that $ALG \leq \left(1 + \alpha + \frac{1}{\beta}\right) OPT$.

To do so, we need two lemmas:

Lemma 1: $ALG^b \leq \left(\alpha + \frac{1}{\beta}\right) \cdot ALG^r$ where ALG^b is the total buying cost and ALG^r is the total production or rental cost.

Lemma 2 By induction on t, $ALG^r_{\leq t} \leq OPT_{\leq t}$ where $ALG^r_{\leq t}$ (respectively $OPT_{\leq t}$) is ALGs total production costs (repsectively OPTs total cost) up to and including time step t.

The Bahncard problem

This problem is motivated by discount passes for the German railway system. Passes are purchased on any given day and a pass is good for some fixed number T of days. There is a known buying cost b for buying a pass, and a known discount factor β for traveling while the pass is valid. That is, if the full cost of a travel request is p, then the discounted cost is $\beta \cdot p$.

The input to the Bahncard problem is an online sequence of $\{(p_1, t_1), (p_2, t_2), \ldots, (p_n, t_n)\}$ where p_i is the full fare for the i^{th} request and t_i is the time step of the i^{th} request. We assume $0 \leq \beta < 1$ since there would be no point in purchasing a pass if $\beta \geq 1$. We also assume that $t_1 < t_2, \ldots < t_n$.

We will assume without loss of genetality (as long as a pass is valid on the day of purchase) that a new pass is only bought when the last pass has expired.

The basic ski rental problem is the special case of the Bahncard problem when $\beta = 0$, $t_i = i$ and $T = \infty$. We could have had arbitrary t_i as the i^{th} day of skiing (rather than the i^{th} day).

Extending the optimal competitive ratio for ski rantal to the Bahncard problem

As in the ski rental problem, the idea is to keep paying full cost until it would have been better to have purchased a pass some time before. In ski rental, the critcal renting cost for buying is b. For the Bahncard problem, the critical cost is $\frac{b}{1-\beta}$. We will then show that the optimal deterministic competitive ratio is $2-\beta$. Since for $\beta = 0$, we have the ski rental problem, and the Bahncard ratio is the ski rental ratio. We will now assume $0 < \beta < 1$ ignoring $\beta = 0$. .

For the lower bound it is sufficient to just consider travel requests until the first purchase of a pass. By making a request every ϵ steps, we can force thew algorithm to buy a pass since otherwise the competitive ratio would be $\frac{1}{\beta} > 2-\beta$ for $0 < \beta < 1$.

We then let s be the time when the pass is purchased and argue by cases that the ratio is $2-\beta$ as $\epsilon \to 0$.
Case 1: $s + \epsilon < \frac{b}{1-\beta}$
Case 2: $s + \epsilon \geq \frac{b}{1-\beta}$

The Bahncard algorithm

Theorem: The following deterministic algorithm achieves competitive ratio $2 - \beta$

Algorithm 8.1.2 The deterministic algorithm for the Bahncard problem based on the critical cost threshold.

---

procedure $BreakEvenBahncard$

$k \leftarrow 0; \tau_0 \leftarrow -T$

▷ τ_k (for $k \geq 1$) will be the time of purchasing the k^{th} bahncard

$i \leftarrow 1$

while $i \leq n$ **do**

$I \leftarrow (\tau_k + T, t_i] \cap (t_i - T, t_i]$ ▷ time interval not covered by last bahncard within last T time units, could be empty

if $\sum_{j:t_j \in I} p_j \geq \frac{b}{1-\beta}$ **then** ▷ the non-discounted cost of tickets during I meets or exceeds the critical cost

$k \leftarrow k + 1; \tau_k = t_i$ ▷ buy a new bahncard

return $\{\tau_1, \ldots, \tau_k\}$

Sketch of analysis for Bahncard competitive ratio

The analysis is in terms of blocks of time where a block is the time following the purchase of a pass up until and including the purchase of a new pass. Since we are considering the asymptotic ratio we can ignore the first and last block.

Each such block consists of two time intervals, namely $I_1 = (\tau_k, \tau_k + T)$ and $I_2 = [\tau_k + T, \tau_{k+1}]$. During I_1, the algorithm's ticket price is at most that of an optimal algorithm so we need only to consider the competitive ratio I_2.

During I_2 the algorithm pays the full-price of all tickets, except for the last one, at which point it purchased the bahncard. Let s_1 denote the total cost of full-price tickets, and let s_2 denote the full-price of the last ticket. Then, during I_2 the algorithm accumulates cost $b + s_1 + \beta s_2$. By the definition of the algorithm $s_1 \leq \frac{b}{1-\beta}$, while $s_1 + s_2 \geq \frac{b}{1-\beta}$, and the duration of I_2 is at least T.

The analysis is completed by considering the cost of OPT when it purchases a pass and when it doesn't purchase a pass.

Extensions of bin packing

There are two reasonably obvious extensions to bin packing. The most obvious extension is multi-dimensional (also called geometric) bin packing; that is, packing rectangles in $\mathbb{R}^2$ into unit dimensional squares and, more generally, packing hyper-rectangles in $\mathbb{R}^d$ into d-dimensional hyper-cubes. This is a very well studied research area with special interest in two and three dimensions. There are different variants of geometric bin packing where the most commmonly stuided case is axis-aligned hyper-rectangles and hyper-cubes.
Aside: In answer to Jacob's question about "gravity' making some solutions infeaible, we can always fo what Amazon,etc do, namely use "still" to fill up and cushon the boxes.

Another very natural extension is vector bin packing; that is packing d-dimensional vectors $\mathbf{v}_i$ into bins $\{B_j\}$ so that sum of vectors in each bin B_j satisfies $\sum_{\mathbf{v}_i \in B_j} \leq \mathbf{1}$ where $\mathbf{1}$ denotes the all 1's vector.

A perhaps less obvious extension of bin packing is *renting servers in the cloud*. We will start with this extension.

Renting servers in the cloud (RSIC)

The inputs for the RSIC problem are a sequence of jobs $\mathbf{x}_i = (a_i, f_i, s_i)$ where a_i (respectively, f_i, s_i) is the arrival time (respectively, the finishing time, the size of the job). We assume the real time model so that $a_1 \leq a_2 \ldots \leq a_n$). We assume that the job sizes (e.g. memory requirements) $s_i \in (0, 1]$. Each cloud server is viewed as a bin of size 1. We will assume an unlimited number of servers as in bin packing.

Similar to the standard bin packing problem, jobs have to be packed into servers so that at any point in time, the server capacity is not violated. A new server needs to be *opened* if no server has capacity for a newly arriving jobs. A server is active if it is hosting any job that has not yet finished. If all jobs are finished on a server, the server becomes inactive.

There are two natural objective functions:
(1) The objective is to minimize the total (over all servers) amount of time that servers are active.
(2) The objective is to minimize the number of servers that are active at any time.

Renting servers in the cloud continued

We shall only consider the first objective which relates to the amount of energy being consumed by the servers.

There are two settings for RSIC. In the clairvoyant setting, we know the finishing time of a job when the job arrives. In the non-clairvoyant version, we only find out about termination when the job finishes. The optimum solution is the same for both versions since the adversary knows the finishing times.

We shall see that there is no constant competitive ratio for the clairvoyant (and hence non-clairvoyant) settings. We can expect that the performance in the clairvoyant setting will be significantly bettter than in the non-clairvoyant setting.

We will start with the non-clairvoyant setting as that is closer to the classical bin packing problem where jobs have infinite duration.

End of Wednesday, November 12 class

We ended here but I am including the rest of the slides for the RISC problem since I am not sure if there will be any more time for me to speak given that we have 9 projects presentations.

Measuring the competitive ratio in terms of the ratio of the maximum to minimum duration of jobs

.

Let $d_i - f_i - s_i$ be the duration of job i, Define $\mu = \frac{\max_i d_i}{\min_i d_i}$. Without loss of generality, we can assume that $\min_i d_i = 1$ to simplfy the discussion so that $\mu = \max_i d_i \geq 1$.

Theorem: Consider any μ and any $\epsilon > 0$, In the non-clairvoiyent setting, every online algorithm ALG has competitive ration $\Omega(\mu)$. More precisely, the determinstic competitve ratio is no better than $\frac{\mu}{1+\epsilon(\mu-1)}$.

An adversary can force the competitive ratio by choosing a set of $(\frac{1}{\epsilon})^2$ jobs each of size ϵ. This forces the algorithm to open at least $\frac{1}{\epsilon}$ servers. The algorithm observes which jobs are on each server. For each server, the adversary will set one job to be a long duration job with duration μ, all other jobs have short duration $= 1$.

By placing all long duration jobs on one server, the optimal cost is $\mu + (\frac{1}{\epsilon} - 1)$. Alg's cost will be at least $\frac{1}{\epsilon}\mu$.

Deterministic Algorithms having competitive ratio $O(\mu)$

As we mentioned the non-clairvoyant setting is quite simular to bin packing (having no job durations) so we can apply almost any bin packing algorithm to non-clairyoyant RSIC. ‘ It turns out that First Fit and the

MTF algorithm (desicribed in the text) have ratio $O(\mu)$. Strangely, Best Fit does not have a constant bound for any μ. We show that a modified Next Fit algorithm has a precise ratio of at most $\mu + 2$. (I don't know if Next Fit is also $O(\mu)$.) The Next Fit opens a new whenever a new job doessn't fit on the most recently openied (and still active) server.

The modified Next Fit algorithm is parameterized with parameter K. We apply Next Fit separately to large jobs of size at least $1/K$ and to small jobs having size less that $1/K$.
Theorem:
For any $K \geq 2$, the competitive ratio of Modkified Next Fit is at most $K \max\{1, \frac{\mu}{K-1}\} + 1 = O(\mu)$. For $K = \mu + 1$, the competitive ratio is $\mu + 2$,

Sketch of analysis for Modified Next Fit RSIC algorithm

First we state a lower bound on any algorithm including OPT. These hold for both clairvoyent and non-clairvotant algorithms.

Fact:
For any onliine sequence $\mathbf{x}$, $OPT(\mathbf{x}) \geq \max(span(\mathbf{x}), util(\mathbf{x})$ where $span(\mathbf{x}) = |\bigcup_i [a_i, f_i)|$ and $util(\mathbf{x}) = \sum_i s_i d(x_i)$.

We want to separately bound Next Fit on the large jobs $\mathbf{x}^L$ and small jobs $\mathbf{x}^S$. Clearly $ModifiedNextFit(\mathbf{x}) = NextFit(\mathbf{x}^L) + NextFit(\mathbf{x}^)S$.

$NextFit(\mathbf{x}^L) \leq \sum_{x_i \in \mathbf{x}^L} d(x_i) \leq \sum_{x_i \in \mathbf{x}^L} K \cdot s_i \cdot d(x_i) = K \cdot util(\mathbf{x}^L)$
The first inequlaity follows from the observation that the the highest cost comes from assigning each large job to a new server.

Finishing the proof for Modified

It remains to bound $NextFit(\mathbf{x}^S)$.

We need to distinguish between a server being closed and a server being released. In the former case, we may close a server if its current size is at least $1 - 1/K$ meaning that the algoirhtm is not allowed to assign any more jobs to this server. Released servers are not part of the objective function. We will refer to a server as being replaced if some job caused it to be closed before being released.

Suppose that the algorithm uses m servers to serve $\mathbf{x}^S$ Let S_i be any server used to serve small jobs. We can break the duration of S_i into the duration $d^-(S_i)$ from the opening of the server until it is closed, and $d^+(S_i)$, the duration from the time S_i was closed until it was released. Then

$NextFit(\mathbf{x}^S) = \sum_{i=1}^{m} d(S_i) = \sum_{i=1}^{m} d^-(S_i) + d^+(S_i)$

Continuing the bound for $\mathit{NextFit}(\mathbf{x}^S)$

Since NextFit has at most one open server at any time, we have $\sum_{i=1}^{m} d^-(S_i) \leq \mathit{span}(\mathbf{x}^S)$.

And now to bound $\sum_{i=1}^{m} d^+(S_i)$, let there be m' replaced servers. Since the duration of each job is at most μ, we have $\mathit{NextFit}(\mathbf{x}^S) \leq \mathit{span}(\mathbf{x}^S) + m'\mu$.

And since the size of a closed server is at least $1 - 1/K$ and the duration of each job is ≥ 1, we have $m' \leq \frac{\mathit{util}(\mathbf{x}^S)}{1-1/K}$.

Combining all the inequalities

Combining all the inequalities we have

$$\begin{aligned}
\mathit{ModifiedNextFit}(\mathbf{x}) &= \mathit{NextFit}(\mathbf{x}^L) + \mathit{NextFit}(\mathbf{x}^S) \\
&\leq K \cdot \mathit{util}(\mathbf{x}^L) + \mathit{span}(\mathbf{x}^S) + \frac{\mathit{util}(\mathit{bfx}^S)}{1 - 1/K} \cdot \mu \\
&\leq \max\left(K, K\frac{\mu}{K-1}\right)(\mathit{util}(\mathbf{x}^L) + \mathit{util}(\mathbf{x}^S)) + \mathit{span}(\mathbf{x}^S) \\
&\leq K \max\left(1, \frac{\mu}{K-1}\right) \mathit{OPT}(\mathbf{x}) + \mathit{OPT}(\mathbf{x}).
\end{aligned}$$

And as we already said, if μ is known, we can set $K = \mu + 1$ to claim a competitive ratio of at most $\mu + 2$.

A brief discussion of the clairvoyant setting for the RSIC problem

We will just state the algorithm and result for the clairvoyant settin witout any proofs. As we will see the clarivoyant setting affords a much improved ratio.

Theorem:
There is a deterministic algorithm (which the paper and text calls HA but I will call *ModifiedFirstFit* although we may have used that term before) for the clairyoyant setting with comptitive ratio $O(\sqrt{\log \mu})$. Furthermore, this is the optimal competitive ratio; that is, for every deterministic algorithm ALG, the competiive ratio is $\Omega(\sqrt{\log \mu})$

Recall that an input to RSIC is a tuple (a, f, s) When a job arrives, we know the duration of every size and now in the clairvoyant case we also know the duration of the job. we can break the set of inputs into $\log \mu$ types. A type $((k, c)$ job ix is such that duration $d(x) \in [2^k - 1, 2^k]$ and the arrival $a(x) \in [(c-1) \cdot 2^k, c \cdot 2^k)$. Since $1 \leq d(x) \leq \mu$ for every job x, we have $\log \mu$ types.

The *ModifiedFirstFit* algorithm

ModifiedFirstFit uses two kinds of servers, *CD* and *GN*. Each *CD* server is associated with a particular job type (k, c). *GN* servers can contain different types. We first give an informal description of the algorithm (with the pseudo code to follow).

When a new job of type (k, c) arrives, if there is an open server with associated with this type we assign it using FirstFit (possibly opening a new $(k, c)CD$ server if necessary. If there are no open *CD* servers, we assign the job using First Fit to a *GN* server as long as the total size of active jobs of size (k, c) is at most $\frac{1}{\sqrt{k}}$ (possibly opening a new *GN* server if necessary). if the total size of active (k, c) jobs is more than $\frac{1}{\sqrt{k}}$ we open a new *CD* server.

The pseudo code for *ModifiedFirstFit*

Algorithm 8.2.2 A clairvoyant algorithm for the RSiC problem that achieves $O(\sqrt{\log \mu})$-competitiveness.

procedure HA
  $i \leftarrow 1$
  while $i \leq n$ **do**
    new input item x_i arrives
    $(k, c) \leftarrow$ type of x_i
    if $\exists$ an open CD server of type (k, c) **then**
      pack x_i into CD servers of type (k, c) using $FirstFit$
      if necessary, open a new CD server of type (k, c)
    else
      $s_{(k,c)} \leftarrow$ total size of all active items of type (k, c)
      if $s_{(k,c)} \leq \frac{1}{2\sqrt{k}}$ **then**
        pack x_i into GN servers using $FirstFit$
        if necessary, open a new GN server
      else
        open a new CD server of type (k, c) and pack x_i into it