CSC2421 Topics in Algorithms: Online and Other Myopic Algorithms Fall 2019

Allan Borodin

October 30, 2019

Announcements

Announcements

- My plan now is to move more quickly over some topics just to give a better idea of the scope of topics that all fit under the umbrella or "online computation" (or at least what we are intending to be in our proposed textbook).
- I would still like everyone, taking the course for credit or not for credit, to choose a chapter or section or topic relating to the textbook for their part of the reading course.
- We could have shorter lectures and meet individually in my office but I prefer to have weekly reports of what individuals are reading.
- At the end of the term, we can summarize all the readings.

Todays agenda

- A brief look at Chapter 9 which discusses a game-theory perspective regarding online algorithms; namely,
  1. Game theory argues in favour of individuals making decision based on self interest and that is inconsistent with our view thus far where a central authority (i.e., the online algorithm) is determining how to make individual decisions. In this regard, we will discuss the k-server on the line problem and a simplified parking problem.
  2. Some online analysis (in particular, primal dual algorithms can be interpreted in terms of pricing of items). At the risk of discussing the Ranking algorithm too much, we will present a game-theorectic interpretation and analysis of the Ranking algorithm.
- Chapter 10 : Online algorithms with advice.

Game theory and mechanism design

In many offline and online applications, events are being generated by *self-interested agents* who make their own individual decisions and do what is best for themselves (i.e. optimize their own "utility"). This is a basic assumption (but often challenged) in game theory; namely that agents are rational and will act so as to optimize their utility. In doing so these individual actions can result in significant harm to some desired "social welfare" objective. This is often refered to as *the tragedy of the commons*.

The field of *mechanism design* refers to algorithms that use incentives (usually but not always money) so as to lead to desired outcomes when agents are self-interested. It is understood that an agent is self-interested (but not malicious).

In a later Chapter, we plan to further discuss online mechanisms in the more traditional, setting of auctions and markets. Here we will just provide two related examples showing how costs can incentivize an online agent to simulate a centralized decision making algorithm resulting in substantially improved social welfare.

A mechanism for the online k-server on the line problem

We recall that the natural deterministic greedy algorithm for the k server problem on the line would result in an arbitrarily bad competitive ratio. We also noted that any k server algorithm can be made into an equally competitive lazy algorithm (where only one server moves to serve a request) by using "virtual locations" for the servers to know where the servers would be in the non-lazy algorithm.

In particular, the lazy DC algorithm for the line metric can be derived from any non-lazy algorithm achieving the same competitive ratio. Moreover, if the transformation to a lazy algorithm is done carefully (in terms of which server will be chosen when there is a tie) in the DC algorithm, then the lazy version will inherit two nice properties ("locality" and "monotonicity") from the non-lazy DC algorithm. Namely, a request will always be served by an adjacent server (locality), and if a new request r would be served by a server s, then every request in the closed interval between r and s would be served by s (monotonicity).

Requests as agents

We suppose that requests for service at a location r are being made by agents arriving online. The agent wants to minimize its cost for service which we can initially think of as the distance that some server will have to travel. Now we know that if agents act greedily and choose the closest server, the overall *social welfare* (that is, the total cost for all requests and hence the average time per server) can suffer arbitrarily.

We want to show how simulate the lazy DC algorithm by dynamically assigning a cost $p(s)$ to each server s so that when an agent decides to use server s, the total cost to get service at location r will become $d(s, r) + p(s)$.

Given these server costs, when an agent decides to use server s, the total cost to provide service at location r will become $d(s, r) + p(s)$. So when an online agent arrives and needs service at location r, the greedy decision is to choose a server so as to minimize the sum of the distance to the desired location plus the cost for the server.

Using prices to simulate the DC algorithm

With an approriate pricing mechanism, the resulting greedy online algorithm will "weakly simulate" the lazy DC algorithm and inherit the k-competiveness of the DC algorithm. Here "weakly simulate" means that even though the lazy DC algorithm satisfies the locality and monotonicity properties stated above and has a well defined way to break ties, greedy agents are free to break ties unpredictably. There is a somewhat delicate argument as to how to insure that the movement costs will remain approximately the same even if the greedy algorithm breaks ties differently than the DC algvkorithm.

Consider the lazy DC algorithm. After some initial requests, we can assume that the k servers are occupying distinct locations. Consider how each request would be served by a specific server. The mapping of servers to locations induces a partition of the line into intervals of potential requests that each server will be resposible for serving. Let $I(s)$ denote the interval that would be served by server s. If servers $I(s)$ and $I(s')$ are adjacent then $I(s)$ and $I(s')$ are adjacent intervals.

The way to price servers

Furthermore, for each pair of adjacent intervals s, s', there is a threshold point $\tau(s.s')$ such that any request in $[s, \tau(s, s')$ will be served by s and requests in $[\tau(s, s'), s')$ will be served by s'.

Let the left to right order of servers be $s_1, \ldots, s_k$. Once we set the price $p(s_1)$ for the left most server s_1, we can then set the remaining server prices so as to satsify the following equation:
$p(s_i) + |s_i - \tau(s_i, s_{i+1}| = p(s_{i+1}) + |s_{i+1} - \tau(s_i, s_{i+1})|$

Lemma

If agents act greedily with respect to the combined server pricing and distance movement, then the greedy decentralized response of the agent will weakly simulate the DC algorithm.

To prove the lemma, we can show that the utility of the agent (i.e. server) that DC uses to serve a request minimizes the total cost (i.e., the disutility of the agent) amongst all servers.

The Parking Problem

An obvious example where agents are creating online events is when drivers look for a legal place to park. They naturally want to park as near to their intended destination as possible. We will study a simplified version of this problem. Namely, we will consider a static problem rather than drivers coming and going for different amounts of time.

The static version is not totally unrealistic as it does model the situation when say indidviduals are driving to an event and everyone leaves after that event or situations where parking is available for the evening (i.e., from 7PM to 6AM).

Many jurisdictions will charge for parking spaces. Note that if the charge is fixed for all spaces, then from our perspective, this is the same as free parking as drivers would only care about the distance between where they park and their desired location.

Our final and main assumption is that the parking spaces are located on discrete points on the line and that there are enough parking spaces for all drivers.

Parking on a line

More specifically, suppose we are given the locations of n offline points $\{x_i\}$ on the real line and let $d(x, y)$ denote the distance between x and y; that is, $d(x, y) = |x - y|$. Suppose $n \leq m$ agents (e.g., shoppers) arrive and will decide to park at some unoccupied location x_i so as to shop at some location y_i.

A central authority (i.e. a mechanism) will determine a cost $p_i(x_i)$ for each parking spot x_i based on the current location of available spaces. An agent parking at x_i incurs a total cost of $d(x_i, y_i) + p_i(x_i)$. The social welfare objective is to minimize $\sum_{i=1}^{n} d(x_i, y_i)$.

The social objective then becomes the special case of min cost metric matching restricted to the line metric. Of course, this is a different social objective than when the goal is to generate revenue for the central authority.

The cost of free parking

We can assume that $n = m$. An offline optimum algorithm can compute a min cost perfect matching in polynomial time for any weighted graph problem. For the special case of matching on the line, there is a very simple linear time optimal algorithm. Namely if the locations are sorted so that $x_1 \leq x_2 \ldots \leq x_n$ and $y_1 \leq y_2 \ldots \leq y_n$, then match x_i to y_i.

In the absense of parking costs, when an agent arrives, the natural greedy decision is to choose an available location closest to the goal destination. (Without loss of generality, we can assume that all distances between adjacent vertices are distinct so as to not need a tie-breaking rule.)

This suggests two immediate questions; namely, what is the competitive ratio of the best online algorithm and what is the competitive ratio when decisions are being made greedily by agents. We will first observe that the natural greedy algorithm has a very poor performance.

The cost of free parking continued

Fact

The competitive ratio for the natural greedy algorithm (and hence if agents act greedily) is exponential in n where n is the number of online requests.

Proof.

Consider the following instance for any $\epsilon > 0$: $y_1 = 1 - \epsilon$, $y_i = 2^{i-1}$ for $2 \leq i \leq n, x_1 = 1.5$, and $x_i = 2^{i-1}$ for $i = 2, \ldots, n$. The optimum matching has cost $.5 + \epsilon$ (for matching x_1 with y_1 and 0 for matching $x_i = y_i$ for $i \neq 1$). However, the greedy solution will match x_1 with y_2 forcing x_i to match with y_{i+1} for $i = 2, \ldots, n-1$ and x_n to match with y_1. The cost of the greedy solution will be $\approx 2^{n+1}$.

□

Using parking costs to allow for greedy parking

- There is a natural (non-greedy) randomized online algorithm (*Harmonic*) that achieves a competitive ratio of $O(\log n)$ for matching on the line. Harmonic is motivated by the deterministic *double coverage* algorithm for the k-server problem with respect to the line metric space and its canonical randomization (analogous to the canonical randomization in the double sided greedy USM algorithm).
- There is a randomized dynamic pricing algorithm for parking locations that will incentivize the shopper to simulate the Harmonic algorithm and hence achieve a competitive ratio of $O(\log n)$. The pricing algorithm does not know the arrival location nor the desired location for the i^{th} arrival but only knows which locations are now occupied given the previous $i-1$ arrivals.
- Each arriving agent knows the cost of each available space and chooses a location x_i so as to minimize the total cost function (i.e., distance to travel and cost of parking spot).

The Harmonic algorithm for matching on the line

If there are n' shopping locations y is to the left (respectively, right) of all available parking places, then match the leftmost $\{x_i\}$ (resp. rightmost $\{x_i\}$) to those locations.

Assume then that a desired shopping location y lies beteen two available spots x_j and x_{j+1}. If d_ℓ (resp., d_r) is the distance of the available spots to the left (resp. right) of y, then match y with x_j with probability $\frac{d_\ell}{d_\ell+d_r}$ and match y with x_{j+1} with probability $\frac{d_r}{d_\ell+d_r}$.

The Harmonic algorithm achieves a competitive ratio of $O(\frac{d_{max}}{d_{min}})$ where d_{max} (resp. d_{min}) is the maximnum (resp. minimum) distance between two vertices.

Using a doubling strategy to modify estimates of OPT and then mlodifying the metric distances accordingly the algorithm can be modified to obtain competitive ratio $O(\log n)$.

An economic interpretation and analysis of the Ranking algorithm

We saw last lecture a primal dual analysis for the Ranking algorithm. Instead of using the notation Y_j (in determining the random ranking of the offline vertices), we will use p_j.

In economic terms, we now interpret each p_j as a price that an online buyer u_i is willing to pay for matching with an offline item v_j; that is, choosing the edge (u_i, v_j) if it exists. In game theory terminology, each buyer has a unit demand valuation $val_{i,j}$ and utility $util_i = val_{i,j} - p_j$ for purchasing item v_j and 0 if u_i does not purchase any item. Here $val_{i,j} = 1$ iff (u_i, v_j) is an edge and 0 otherwise. The revenue rev_j for item v_j is p_j if the item is sold and 0 otherwise.

Economic interpretation of Ranking continued

The goal is to maximize the *social welfare* of a matching M which is defined as $|M| = \sum_{u_i,v_j)\in M} val_{i,j}$. Equivalently, social welfare can be defined as $\sum_i util_i + \sum_j rev_j$; that is, the total utility of the buyers plus the total revenue of the sellers for all items sold.

In this interpretation, for any matching M, the social welfare is $\sum_{v_j \in M}(1 - p_j) + \sum_{v_j \in M} p_j = \sum_{v_j \in M} 1 = |M|$. The desired competitive ratio will follow from the following insightful lemma.

Lemma

Let the r_j be chosen uniformly at random and independently in $[0, 1]$ and let $p_j = e^{r_j - 1}$. Then for every ordering of the online u_i vertices and for every edge (u_i, v_j) in the input bipartie graph, we have :
$\mathbb{E}_{\{r_j\}}[\sum_{u_i} util_i + \sum_j p_j] \geq 1 - \frac{1}{e}$.

Sketch of Ranking proof in the economic interpretation

We consider a fixed arrival order, buyer u_i and item v_j. We need to bound $\mathbb{E}_{\{r_j\}}[util_i]$ and $\mathbb{E}_{\{r_j\}}[p_j]$. Let $(u_i, v_j) \in E$ and let $p = e^{y-1}$ for some y be the price of the item purchased by u_i and let $p = 1$ if u_i does not purchase any item. We bound the utility of u_i and the expected revenue of item v_j.

- $util_i \geq 1 - p$
- $\mathbb{E}[rev\ \ v_j] = \mathbb{E}[p_j \cdot \mathbb{I}_{v_j \text{ is sold}}] \geq \mathbb{E}[p_j \cdot \mathbb{I}_{p_j < p}] = \int_0^y e^{r_j - 1} dr_j = p - \frac{1}{e}$

Let M^* denote a maximum matching and M^R the matching given by the Ranking algorithm. Then $\mathbb{E}_{r_j}[|M|] \geq (1 - \frac{1}{e})|M^*|$.

We have $|M^R| = |\mathbb{E}_{\{r_j\}}[\sum_i util_i + \sum_j p_j] \geq \mathbb{E}_{\{r_j\}}[\sum_{(u_i, v_j) \in M^*}(util_i + p_j)] \geq (1 - \frac{1}{e})|M^*|$
where the first inequality is an inequality rather than an equality as the matching M might include an edge where $p_j = 1$,

Online algorithms with advice

In practice, an algorithm designer often has some side information about the input, e.g., input length, maximum weight of an input item, maximum degree of a vertex in a graph, number of distinct input item types, and so on. This side-knowledge can be formally captured by the notion of *advice*, which we study in Chapter 10.

There are two prominent models by which an algorithm receives advice from a trusted oracle, namely, the *per request* model and the *tape* model. (Last Friday, Spyros Angelpoulos spoke about online algorithms with untrusted advice.)
Aside I personall favour the tape model but that is personal taste.

As an example of the tape model, recall the Time-Series Search problem. Prior to receiving any input items (daily exchange rates p_j), an algorithm is supplied a lower bound L and an upper bound U on exchange rates. This is an example of side information about future input items, i.e., $L \leq p_j \leq U$ for all j. This side information is necessary to design a non-trivial online algorithm for the Time-Series Search.

The per request model

The oracle fixes a universe of answers $\mathcal{U}$. When the algorithm receives an online input item x_i, it receives some side information $a_i \in \mathcal{U}$ from the oracle alongside x_i. The side information a_i is written in binary using $\lceil \log |\mathcal{U}| \rceil$ bits.

The decision d_i of the algorithm can now depend on all previously observed input items as well as all oracle answers received so far. More formally, d_i is a function $d_i := d_i(x_1, \ldots, x_i, a_1, \ldots, a_i)$.

If the input length is n, then the total length of advice, also known as *the advice cost* of the algorithm, is $\lceil \log |\mathcal{U}| \rceil n$ in this model.
Aside: This why I favour the tape model, where sometimes even 1 bit of advice can have great impact.

The tape model

The oracle populates an infinite advice tape. The algorithm reads from the advice tape sequentially. At any point in time during the execution of the online algorithm, the algorithm can decide to read more advice bits. *The advice cost* of the algorithm on inputs of length n is the maximum number of bits read by the algorithm to process an input sequence, where the maximum is taken over all input sequences of length n.

The decision d_i of the algorithm is now a function of all previously observed input items as well as all the advice bits that have been read by the algorithm so far.

A feature of this model is that an algorithm does not have an a priori bound on the number of advice bits. But I prefer to just think of this more simply that the oracle looks at the entire input and compresses its knowledge into some "small" number of advice bits.

I like to more simply think of the algorithm seeing the imput and prividing all the advice bits at once. The reason for the "technical aspect of an infinite tape" is to emphasize that the algorithm does not have to know n.

The good and bad of the advice model

Our main focus will be the competitiev ratio that can be obtained (for a given problem) with regard to a given amount of advice. We will see that sometimes very little advice can dramatically change the quality of the competitive ratio. In other cases, we can show that to achieve good performance we need $\Omega(n)$ bits to achieve a good ratio for an n bit input.

We will also be interested in the relation between advice and randomization.

While the basic motivation for the advice models is to model realistic situation where it is likely that reasonable information about the input would be known. The idea is to abstract this by saying that the amount of advice is limited so that the oracles has to compress what it knows.

Tha bad news is that when we use a purely information theoretic view of advice, we allow the oracle to do an arbitrary amount of computation (e.g., to learn an optiomal solution) in creating the advice. But still I find the results interesting.

Equivalent views of advice

Besides the oracle view of advice, there are some alternative views of advice.

- In the tape model, if we have b bits of advice then this is equivelnt to having 2^b parallel streams $\{ALG_1, ALG_2, \ldots, ALG_{2^b}\}$ of online algorithms and taking the best solutuion ALG_i at the end.
- In the per request model, we can view an algorithm using say b bits per request as a tree of online algorithms where upon receiving the i^{th} input the algorithm performs a 2^b way branch.
- Advice algorithms in either model model be viewed as non-deterministic algorithms that guess each advice bit.

As we point out, there is a model of advice in say Turing machine offline computation that is simimilar to the tape model. In the Turing machine tape model there is a separate advice tape. However, that advice only depends on the length of the input whereas in the online tape advice model the advice is based on the entire input.

An example where 1 bit of advice is critical

Recall the proportional knapsack problem that we used in Chapter 3 to illustrate the power of randomness. We are given the sequence $w_1, w_2, \ldots, w_n$ with $w_i \in (0,1]$ for all i. The goal is to find choose a subset $S \subseteq \{1, 2 \ldots n\}$ so as to maximize $\sum_{i \in S} w_i$ subject to $\sum_{i \in S} w_i \leq 1$.

We showed that without any advice, any deterministic online algorithm has unbounded competitive ratio. But just as we used 1 but of rendomness to obtain a $\frac{1}{4}$ competitive ratio, we can use 1 bit of advice to obtain competitive ratio $\frac{1}{2}$. Namely, the one bit of advice indicates whether or not there is an input $w_j \geq \frac{1}{2}$. If so, it waits for this item before packing any other item. Otherwise, it packs greedily accepting all items that fit.

More generally, any online algorithm using b bits of randomness to achieve competitive ratio ρ (against an oblivious adversary) is easily converted into a deterministic algorithm with b bits of advice achieving ratio (at least as good as) ρ.

Advice vs randomness

We just saw that b random bits can always be replaced with b advice bits, which works really well for "barely random" algorithms. There is a rather trivial converse for maximization problems that also works well for small values of b. Namely, consider a deterministic algorithm with b bits of advice. We can replace the advice bits by b random bits and then run the advice algorithm using the b random bits as the advice. Provided that the value of the objective function is always nonnegative, the expected competitive ratio of this randomized algorithm is at least $\rho/2^b$.

What if a randomized algorithm is not barely random and uses some potentially very large number of random bits? Turns out that it is still possible to replace random bits with very few advice bits provided that the number of distinct input sequences of length n is not too high and that we allow a slight deterioration in the competitive ratio.

Converting a randomized minimization algorithm into a determninistic advice algorithm

Consider a minimization problem. Let $\mathcal{I}(n)$ denote all possible input sequences of length n. Set $I(n) := |\mathcal{I}(n)|$. Suppose there exists a randomized algorithm ALG that achieves worst-case expected competitive ratio $\rho(n) \geq 1$ on inputs of length n. Then for any $\epsilon > 0$ there exists a deterministic algorithm ALG' that uses $\log n + 2\log\log n + \log\left(\frac{\log I(n)}{\log(1+\epsilon)}\right)$ bits of advice and achieves competitive ratio $(1+\epsilon)\rho(n)$.

Consider a maximization problem. Let $\mathcal{I}(n)$ denote all possible input sequences of length n. Set $I(n) := |\mathcal{I}(n)|$. Suppose there exists a randomized algorithm ALG that achieves worst-case expected competitive ratio $\rho(n) \leq 1$ on inputs of length n. Then for any $\epsilon > 0$ there exists a deterministic algorithm ALG' that uses $\log n + 2\log\log n + \log\left(\frac{\log I(n)}{\log \delta}\right)$ bits of advice and achieves competitive ratio $(1-\epsilon)\rho(n)$, where $\delta = \frac{1-(1-\epsilon)\rho(n)}{\rho(n)}$.

The proof of the minimization result

The idea behind the proof is as follows. Although ALG can potentially use a lot of randomness, we can identify a small set G of "good strings" of randomness such that on each input x there is at least one string $r \in G$ that guarantees $ALG_r(x)/OPT(x) \leq (1+\epsilon)\rho(n)$. Both the oracle and the algorithm ALG' can construct the set G by knowing $n = |x|$ only. Thus, the oracle can specify the value n on the advice tape using $\log n + 2\log\log n$ bits (using one of the uniquely decodable codes with such guarantees). In addition, the oracle specifies the index of a good string of randomness relative to G. This requires $\log|G|$ more advice bits. The proof is completed by using a standard counting argument to show that $|G| \leq \frac{\log I(n)}{\log(1+\epsilon)}$.

The argument is very similar to the argument that $RP \in P/poly$ (i.e., that decision problems computable by 1-sided error randomized time computation can be computed by polynomial size circuits for each input length n). The argument for online algorithms is due to Böckenhauer et al [2014].

Details of the conversion of randomness to advice for a minimization problem

Initially, G is set to be $\varnothing$. We identify the first "good string" of randomness as follows. Suppose that ALG uses $b(n)$ random bits. Let $B(n) := 2^{b(n)}$ denote the number of possible random strings. Consider an $I(n) \times B(n)$ matrix M such that $M(x,r)$ is the competitive ratio of ALG on input x when random bits are fixed to the string r. That is $M(x,r) := ALG_r(x)/OPT(x)$. By the definition of competitive ratio, for each $x \in \mathcal{I}(n)$ we have

$$\frac{1}{B(n)} \sum_r M(x,r) = \mathbb{E}_r(ALG_r(x)/OPT(x)) \leq \rho(n).$$

Therefore, we have $\sum_r M(x,r) \leq B(n)\rho(n)$. Adding these inequalities over all values of x we get that $\sum_{x,r} M(x,r) = \sum_r \sum_x M(x,r) \leq I(n)B(n)\rho(n)$. Thus, there exists a string r such $\sum_x M(x,r) \leq I(n)\rho(n)$. We add r to G: $G \leftarrow G \cup \{r\}$.

Proof for minimization result continued

An input x such that $M(x, r) \leq (1 + \epsilon)\rho(n)$ is said to be "covered" by r, since ALG' knowing r on input x can guarantee that competitive ratio of at most $(1 + \epsilon)\rho(n)$ by simulating $ALG_r(x)$. Thus, these inputs can be excluded from further consideration. We shall continue the above approach of identifying "good strings" of randomness with respect to those inputs x that are not covered by r. It is left to see that each time we identify a "good string" of randomness, we cover a large fraction of the inputs. This will bound the number of times we need to iterate the process of identifying the next "good string" until *all* inputs $\mathcal{I}(n)$ are covered.

Proof for minimization result continued

Let s denote the number of strings that are covered by r. Then $I(n) - s$ are not covered by r, so for those inputs x we have $ALG_r(x) > (1+\epsilon)\rho(n)$. Therefore, we have

$$\sum_x ALG_r(x) > (I(n) - s)(1+\epsilon)\rho(n).$$

Since r was chosen so that $\sum_x M(x,r) \leq I(n)\rho(n)$, we get

$$(I(n) - s)(1+\epsilon)\rho(n) < I(n)\rho(n).$$

Simplifying, we obtain that $s > \frac{I(n)\epsilon}{1+\epsilon}$. Therefore, each time we find a "good string" of randomness, the number of *uncovered* instances reduces by a multiplicative factor $\frac{1}{1+\epsilon}$. Therefore, after $\frac{\log I(n)}{\log(1+\epsilon)}$ iterations there will be no more uncovered instances. Hence, it follows that $|G| \leq \frac{\log I(n)}{\log(1+\epsilon)}$.

A consequence of this randomness to advice result

For many optimization problems, we have the bound $I(n) \leq 2^{n^{O(1)}}$. Therefore, the above theorems let you convert a randomized algorithm with competitive ratio $\rho(n)$ into a deterministic algorithm with $O_\epsilon(\log n)$ bits of advice with competitive ratio $(1+\epsilon)\rho(n)$ (resp ratio $(1-\epsilon)\rho(n)$) for a minimization (resp. maximization) problem.

Consider the Ranking algorithm for the BMM problem. Specifically lets assume n online and n offline nodes. Then $I(n) = n! \approx n^n = 2^{n \log n}$ so that we can apply the randomness to advice result to obtain the following.

There is a deterministic online algorithm with $O(\log n)$ bits of advice for the BMM problem with competitive ratio $(1-\epsilon)(1-\frac{1}{e}$ for any $\epsilon > 0$.

Nicolas Pena and I showed that $\Omega(\log \log n)$ advice bits are required to asymptotically improve upon the $\frac{1}{2}$ competitive ratio achieved by any deterministic greedy BMM algorithm.

Repeatable problems: Converting an advice online algorithm into a randomized online algorithm

For some problems, there is a converse result due to Mikkelsen [2016] which is both somewhat technical to state and requires a non-trivial proof.

While the exact definition of repeatable problems is slightly technical, the main idea is rather straightforward. Consider r input sequences $I_1, \ldots, I_r$. A problem is called repeatable if processing the combined sequence $I = I_1 \cdots I_r$ essentially amounts to processing each I_1, I_2, …, I_r individually. In other words, no matter which online algorithm is used to process the prefix $I_1 \ldots I_j$, the past information and decisions about $I_1 \ldots I_j$ cannot significantly affect the performance of the algorithm on I_{j+1} as compared to running the algorithm on I_{j+1} alone.

What is and what is not repeatable

Consider the paging as an example of a repeatable problem. Processing I_1 can affect the number of cache misses on I_2 by $\pm k$ only. Introduce the notation $ALG(I_1 I_2)|_{I_2}$ which corresponds to running ALG on the combined input $I_1 I_2$ and counting the number of cache misses just on the I_2 part of the input. Thus, we have $ALG(I_2) = ALG(I_1 I_2)|_{I_2} \pm k$. Provided that $OPT(I_1), OPT(I_2) \to \infty$ the term $\pm k$ becomes negligible. Thus, past history doesn't significantly affect ALG.

A typical example of a non-repeatable problem in the literature is Bin Packing. Consider $I_1 = (1/2 - \epsilon, \ldots, 1/2 - \epsilon)$ of length n followed by $I_2 = (1/2 + \epsilon), \ldots, (1/2 + \epsilon)$ also of length n. If the first sequence is processed by placing the items in pairs into $n/2$ bins, then the second input sequence requires n additional bins. However, if the first input sequence is processed by placing the items into n different bins, then the second input sequence requires 0 extra bins, since an item $1/2 + \epsilon$ can be placed into a bin with an item $1/2 - \epsilon$.

Two types of repeatable problems

- Σ-repeatable problems : the value of the objective achieved on $I_1 \dots I_n$ is approximately the *sum* over j of the objective value achieved on I_j individually.
- $\vee$-repeatable: the value of the objective achieved on $I_1 \dots I_n$ is approximately the *maximum* over j of the objective value achieved on I_j individually.

We will not state the formal definition of repeatable problems. The general definition is quite technical for several reasons: firstly, all our informal statements involving words"approximately" and "significantly" need to be formally quantified; secondly, the general definition of repeatable allows arbitrary transformations f of input sequences $I_1, \dots, I_n$ to obtain the combined input sequence $I = f(I_1, \dots, I_n)$. In our examples above, we have only considered the simplest kind of f, namely concatenation.

The Mikkelsen results for repeatable prolems

Informal result for Σ-repeatable

Let P be a Σ-repeatable problem. Suppose that the number of input sequences of length n is finite. Fix $c > 0$. If randomized online algorithms cannot achieve competitive ratio better than c for P even when the length n of the input is known to the algorithm in advance, then every deterministic algorithm with $o(n)$ bits of advice cannot achieve competitive ratio better than c.

Informal result for $\vee$-repeatable

Let P be a $\vee$-repeatable problem. Suppose that the number of input sequences of length n is finite. Fix $c > 0$. If deterministic online algorithms cannot achieve competitive ratio better than c for P even when the length n as well as the value of OPT are known in advance, then every (possibly randomized) algorithm with $o(n)$ bits of advice cannot achieve competitive ratio better than c.

A consequence for Σ repeatable problems

The Σ repeatable theorem allows you to transfer lower bounds on randomized online algorithms to lower bounds on deterministic algorithms with $o(n)$ bits of advice.

We note that the biprtite matching problem is a Σ-repeatable problem. It follows that any online algorithm with $o(n)$ advice bits can substantially improve on the $(1-\epsilon)\frac{1}{e}$ competiive ratio using $O_\epsilon(\log n)$advice bits.

It should be clear that $n \log n$ advice bits provides an optimal solution. Moreover, Miyazaki [2014] has shown that $\Omega(n \log n)$ advice bits are required to obtain optimality.

The following result due to Dürr, Konrad, and Renault [2016] fills in the results for the BMM problem when using $\Theta(n)$ advice bits:

- There is a deterministic $(1-\epsilon)$-competitive algorithm using $O(\frac{n}{\epsilon^5})$ advice bits.
- Every deterministic $(1-\epsilon)$-competitive algorithm must use $\Omega(\log(\frac{1}{\epsilon})n)$ advice bits.

Other impossibility results

In spite of the power of the adversary to see the entire input, there are problems where the number of advice bits to obtain optimality or a (small) constant approximation requires almost as many advice bits as “the naive advice optimal solution”. Some impossibility results discuss the number of advice bits to beat the best known (poly time) offline approximations.

I am just going to mention some results where I am now taking the information from a survey by Boyar et al [2016]. I will post this survey. (I’ll also check with the authors if there are any major new developments.)

It is not surprising that the bin packing problem would be well studied given its prominent role in offline algorithms. Note that the input to the bin packing problem is the same as the input to the proportional knapsack problem (where 1 bit of advice gave a “good” approximation) but, of course, with different objective function.
When considering the advice-competitiveness needed for online bin packing we should keep in mind the current best known NP hardness and polytime (strict) approximation results, namely 1.54037 and 1.5815 (respectively).

Advice results for bin packing

Let OPT be the number of bins used by an optimal solution for an input of n items. Then $n \cdot \log(OPT)$ advice bits are clearly sufficient to obtain optimality by an online algorithm. In the text, we show that something close to this number of advice bit is necessary. Namely,

Every online algorithm with advice that is optimal for bin packing must use $(n - 2OPT) \log OPT$ advice bits.

Here are some additional results (see Boyar et al for references)

- For any $\epsilon > 0$, there is an online algorithm with ratio $1.47012 + \epsilon$ using some constant $f(\epsilon)$ advice bits
- In particular, 16 advice bits is sufficient to obtain ratio 1.530
- For any $\epsilon > 0$, $O(\frac{1}{\epsilon} \log(\frac{1}{\epsilon}))$ advice bits are sufficient to obtain a $(1 + \epsilon)$ competitive ratio.

Reductions using the string guessing problem

We will conclude for now our discussion of advice complexity. (Kaman is doing his project on this topic.)

There is a rather artificial problem where we have pretty tight necessary and sufficient bounds on the advice complexity. This problem can be used by an appropriate reduction to prove negative results for more natural problems. We state such a result for the online set cover problem in the text. (The following problem has also been useful in studying priority algorithms with advice.)

The artificial problem is the string guessing game asks an online algorithm to guess the next symbol from an alphabet Σ of size q which comes in two varieties:

- String guessing game with known history (q-SGKN) where after each guess the true symbol is relvealed
- String guessing game with unknown history (q-SGUN) where the true sequence of symbols are bot revealed until the end.

Tight results for the string guessing game

Fix $\alpha \in [1/q, 1)$. There is an online algorithm that guarantees at most $(1-\alpha)n$ mistakes for q-SGKH and q-SGUH and uses advice of length at most

$$\left\lceil \left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha\log_q\alpha\right) n\log q + \frac{3}{2}\log n + \log(\ln q) + \frac{1}{2}\right\rceil .$$

Any online algorithm with advice for q-SGKH that makes at most $(1-\alpha)n$ mistakes on inputs of length n, where $\alpha \in (1/q, 1)$, reads at least

$$\left(1 + (1-\alpha)\log_q\left(\frac{1-\alpha}{q-1}\right) + \alpha\log_q\alpha\right) n\log q$$

bits of advice.

Consequences of string guessing game for online set cover

- To obtain a strict $(1-\alpha)n$ competitive ratio requires $(1+(1-\alpha)\log(1-\alpha)+\alpha\log\alpha)\,n$ bits of advice.
- For asymptotoic results, fixing $c \leq 1.5$, any online algorithm that achieves asymptotic competitive ratio c for Set Cover must use:
  1. $b \geq (1+(c-1)\log(c-1)+(2-c)\log(2-c))\frac{n}{3}$
  2. $b \geq (1+(c-1)\log(c-1)+(2-c)\log(2-c))\frac{m}{3}$

  where $m = |\mathcal{S}|$ and $n = |X|$.