CSC2421: Online and Other Myopic Algorithms
Fall 2025

Allan Borodin

October 8, 2025

Week 6

Annoucements

- First problem set is due October 22 at noon.

Todays agenda

- Some project suggestions
- The deterministic and randomized two-side online algorithms for unconconstrained non-monotone submodular maximization (USM)
- De-randomizing the two-side algorithm into a parallel online program
- When greediness is not good. We will look at some examples where a greedy solution is too simple. And we will also look at some cases where being greedy is more than we want.

  Discussion of some algorithms in chapters 5 and 7.

  1. k server on the line metric
  2. Bipartite matching: priority choice of next items vs random order
  3. Min cost perfect matching for arbitrary metrics and min cost perfect matching for the line metric.

Some possible project topics

Note: I do not expect projects to have new results. Of course, that would be great but is definitely not expected. But for a given topic, what is expected is a good sense of what has been done, what are open problems, and possible future directions. A project could make a nice contribution by setting up some experiments on "real data" or synthetic data.

Going through the chapters while thinking of later topics.

- I only briefly mentioned the line search problem. That is one example of a navigation problem. Good to understand more about naivigation problems. Games as navigation problems; e.g., WORDLE
- Temporary items. See section 2.8. Besides the identical machines model, consider other machine models (e.g., related and restricted machines) for makespan
- Bounded memory and memoryless algorithms. (See section 3.13.) Memory constraints apply to both deterministic and randomized algorithms.

More project suggestions

- Since chapter 4 discusses some classical problems, open problems have been around for a while. But for sure, not all the classical problems have been studied with regard to the ROM and priority models. One of the first priority results was a lower bound for a fixed order priority algorithm with regard to the makespan problem for the restricted machines model. It is not quite a tight result and no lower bounds for adaptive priority for this problem. I am not aware of ROM and priority algorithms for the other machine models.
- Chapter 5 is a more abstract chapter dealing with metric space problems and the MTS model. There is a relation between the MTS model and expert learning in chapter 6. In general these are well studied problems. There are some open problems concerning variants of the k-server problem and these might be worth more study. See also section 8.3 for k-server extensions. As for the randomized k-server conjecture, it would be interesting to know for what metric spaces can we obtain $O(\log^{\ell} k)$ competitive ratios. The line metric is of special interest.

And more possible project suggestions

- The min cost matching (for arbitrary and specific metric spaces) and min colouring problems could be interesting projects. Even the line metric is not fully understood for randomized min cost matching. More generally, there are a number of open problems mentioned in chapter 7. And here we explicitly raise the issue of how many random bits are needed to obtain various competitive ratios. When delqing with metric space results there often a gap between what is known for specific metric spaces and arbnitrary metric spaces. Similarly for graph problem, results for arbitrary graphs are mostly quite negative but more positive results can often be obtaied for various classes of graphs.
- I welcome any project that can simplify and make resuilts more understandable. One such result is the competitive ratio for vector bin packing by the FirstFit algorithm. So far we just state the First Fit competitive ratio result for vector bin packing in Theorem 8.2.10 without proof.

Online mechanism design and social choicd theory

- In chapter 9, we discuss online mechansim design and social choice thoery. In both of the topics, we have self interested agents who are either providing inputs (which they may not do truthfully) or making decisions (which may be good for themselves but bad for the more general "social good". (This tension between indiviudal actions and overall social objecives is often referred to as the tragedy of the commons. Online fair division and more generally online, ROM, real-time and priority in the context of mechanism design and social choice theory is a relatively new topic with regard to online algorithms.
- Since we moved this chapter forward we have not yet spent the time it deserves and that makes chapter 9 a good source for a project.
- One section that needs to be redone is section 9.3. Here online decisions are being made by agents. Drivers want a parking spot and will find the closest spot upon arrival This is simplified and modelled as a min cost on the line problem where now a mechanism will dynamically charge prices for parking spots to insure an overall good matching.

And still more project suggestions

- Chapter 11 will become chapter 10. It includes the max-sat problem which we discussed last class. The competitive ratio for Max-Sat (or other constraint satisfaction problems) with regard to online, ROM and priority algorithms is of interest especially for max-sat and input model 3. Today we will talk about the "two-sided online greedy" algorithm. Not sure what other problems can be viewed in this extended framework.
- We are planning to make chapters 1-9 + 11 into volume 1 and rhe remaining chapters will become volume 2.
- The current chapter 10 will become chapter 11. This chapter concerns online primal dual algorithms in general. The chapter also returns to the bipartite max matching problem and provides an elegant way to understand the ranking algorithms and the pertubed ranking algorithm (for offline vertex weighted bipartitie matching). What other well studied problems can be elegantly understood within this framwework? (There is a great monograph concerning online primal dual algorithms.)

The two sided online algorithm for non-monotone submodular maximization

There are alternative ways to define submodular functions. The definition that leads itself to online and priority algrotihms is one of diminishing marginal gains. More precisely, a function f is submodular if it satisfies the following property for all $S \subseteq T \subseteq U$ and $x \notin T$: $f(S \cup \{x\}) \geq f(T \cup \{x\}$.

In considering submodular functions, we usually have the additional properties that the functions are normalized (i.e., $f(\varnothing) = 0$) and monotone (i.e. $f(S) \leq f(T)$ whenever $S \subseteq T$) . A common optimization problem is to find a subset S that maximizes $f(S)$ subject to some constraint. There are also examples of non monotone submodular functions and in the non-monotone case, it is also meaningful to study the unconstrained maximization problem. The two most prominent examples of non-monotone submodular functions are finding maximum cuts in graphs and directed graphs; that is, given a graph (or digraph) $G = (V, E)$, find a subset S that maximizes $|\{(u, v) \in E : u \in S, v \in V \setminus S\}|$.

How to represent submodular functions

For a universe U of n elements, the input for the problem consists of the 2^n possible subset values. Hence to be able to consider more efficient algorithms, we need to have a model that allows access to information about the function f without having to explicitly list all subset values.

The most common type of access is called a *value oracle* which allows an algorithm to ask for the value of $f(S)$ for any specified subset S. The complexity of algorithms that utilize a value oracle is often measured by the number of oracle calls, ignoring other computational steps.

For an explicitly defined function (such as max cut), we do not have to assume access to an oracle and time is measured in the usual way by counting all time steps. For Max-Cut and Max-Di-Cut, substantially better offline apporoximations based on SDP are known than for arbitrary non-monotone submodular function algorithms.

Two natural priority algorithms that fail for maximizing non-monotone submodular functions

We first note that the natural greedy algriththms that provides a $\frac{1}{2}$ ratio for monotone submodular maximization (subject to a matroid constaint) does not provide a constant ratio for the unconstrained submldular maximization (USM) problem. Namely, let S_i be the solution at the start of the i^{th}. The natural greedy (adfaptive order priority) algorithm would select an element to maximize $f(S_i \cup \{x\}) - f(S_i))$.

Another natural algorithm would be to start with $S_0 = U$ and keep eliminating the element than gave the least gain. That also fails to achieve a constant ratio.

In the offline line world it is known that for arbitrary non-monotone submosular maximization, any algorithm using value oracles requries exponentially many oracle calls to obtain a ratio better than $\frac{1}{2}$. There is also a complexity assumption that shows that no polynomial time algorithm can beat the $\frac{1}{2}$ ratio.

The deterministic two-sided online algorithm for the unconstrained non-monotone submodular maximization (USM)

There was a series of results for improving offline approximation results but the best known ratio was below $\frac{1}{2}$ until ...

The deterministic two-sided online algorithm for the unconstrained non-monotone submodular maximization (USM)

There was a series of results for improving offline approximation results but the best known ratio was below $\frac{1}{2}$ until ...

Buchbinder et al derived an elegant $\frac{1}{3}$ deterministic algorithm which I will call the determnistic two sided greedy online algorithm and then showed that the *natural randomization* of that algorithm obtains a $\frac{1}{2}$ ratio. If the following sketch sounds similar to the MaxSat $\frac{3}{4}$ randomizedI algorithm, this is because the ramdomized two-sided algorithm becomes the Max-Sat algorithm when properly applied. I obviously really like this algorithm. Why?

The deterministic two-sided online algorithm for the unconstrained non-monotone submodular maximization (USM)

There was a series of results for improving offline approximation results but the best known ratio was below $\frac{1}{2}$ until ...

Buchbinder et al derived an elegant $\frac{1}{3}$ deterministic algorithm which I will call the determnistic two sided greedy online algorithm and then showed that the *natural randomization* of that algorithm obtains a $\frac{1}{2}$ ratio. If the following sketch sounds similar to the MaxSat $\frac{3}{4}$ randomizedI algorithm, this is because the ramdomized two-sided algorithm becomes the Max-Sat algorithm when properly applied. I obviously really like this algorithm.
Why?
Because is is elegant, it beat the existing offline algorithms, and it is optimal for non-monotone submodular maximization.

Brief sketch of the two sided algorithms for the USM problem

We maintain two partial solutions X_i and Y_i based on the first i elements in U. $X_0 = 0$ and $Y_0 = U$. X_i will be non-decreasing and Y_i will be non increasing.

For $i \leq n = |U|$
$t_i = f(X_{i-1} \cup \{u_i\}) - f(X_{i-1})$;
$f_i = f(Y_{i-1} \setminus \{u_i\}) - f(Y_{i-1})$
t If $t_i \geq f_i$ then $X_i = X_{i-1} \cup \{u_i\}$; $Y_i = Y_{i-1}$;
otherwise $X_i = X_{i-1}$; $Y_i = Y_{i-1} \setminus \{u_i\}$
s The solutions X_i and Y_i agree on the first i elements in U. Thus

$X_n = Y_n$ is the solution. This yields a $\frac{1}{3}$ ratio.

We obtain a randomized $\frac{1}{2}$ algorithm by randomly assigning u_i proportionally as in the Max-Sat algiorithm. And as in that analysis, we have to prove that $t_i + f_i \geq 0$.

Can we de-randomize the $\frac{1}{2}$ ratio?

What do we mean by de-randomize?

Can we de-randomize the $\frac{1}{2}$ ratio?

What do we mean by de-randomize? Ideally, we would like to simulate a randomized algorithm by the same (or closely related) online or myopic algorithm. Norman Wang and I gave evidence that "a large class of algorithms resembling the two-sdied algorithms" cannnot achieve the $\frac{1}{2}$ ratio for arbitrary non-monotone submodular functions even if we use a priority ordering for the elements of u_i.

One naive way to de-randomize the algorithm is to form the randomized tree which creates a levelled tree of depeth $n = |U|$where the width of nodes at the i^{th} leavel is 2^i.

Buchbinder and Feldman use linear programming and extremal solutions to show that they can replace the $\frac{1}{2}$ randomized tree by a levelled DAG. In their DAG, the $(i+1)^{st}$ level will only have 2 more nodes than the i^{th} level. This then is a polynomial time algorithm (with value oracles). Similarly, there is a width $2n$ parallel algorithm achieving a $\frac{3}{4}$ ratio for MaxSat. This is the kind of parallel online algorihms I was alluding to at the start of the last class.

Greed is good, or is it?

"Greed, for lack of a better word, is good. Greed is right, greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit. Greed in all of its forms – greed for life, for money, for love, knowledge – has marked the upward surge of mankind".
Source: This iconic line is spoken by the character Gordon Gekko, played by Michael Douglas, in the 1987 film Wall Street.

Bust not for some problems in the context of online and priority algorithms

First what do we mean by "greedy"? As I suggested early in the course, our intuitive meaning of greedy is "live for today" in the following sense: Just before the i^{th} item σ_i arrives, we are in some state S_{i-1} of the computation. Based on S_{i-1} and the current and σ_i, an algorithm makes an optimum decision disregading what future items might arrise (even if we know that more items are going to arrive). An adaptive order priority algorithm can be greedy in both its choice of the next item and the decision it will make for that item.
Note: Often there is more than one greedy choice.

Some examples where it is not good to be greedy

Although greedy decisions can be good decisions, there are many instances of online and greedy algorithms where greedy decisions are not optimal and can somethimes be provably extremely bad.

Note: I just became aware of an August 2025 paper "Approximate Proprotionality in Online Fair Division" by Choo et al where the authors show that various greedy strateiges do not acheive a notion of fairness called Prop1. This motivated me to want to mention some examples where greedy decisions are provably bad.

- We just mentioned that for maximizaing a non-monotone submodular function, it is not good to be greedy in the sense of a priority algorithm choosing the next item to be one that maximizes the marginal gain. Instead we hedge our bets and pursue two greedy strategies simultaneously for the elementis u_i arriving online.

More examples where it is not good to be greedy

- The k-server problem on the line and deterministic online algorithms. Here we have a line metric. We can consider the case of the continuous line (an infinite metric space) or the finite metric space induced by N points on the line. We have k servers that at any time are located at k (not necessarily distinct) points in the metric space.

  A greedy decision would be to serve an online request x_i by a server closest to x_i. This greedy strategy will result in a competitive ratio that is arbitrarily bad in terms of the competitive ratio. The *DoubleCoverage* presented in Section 5.2 algorithm moves two servers at equal speeds until one reaches the request. *DoubleCoverage* achieves a k-competitive ratio which is the optimal ratio. Note: For every metric space on $N \geq k+1$ points, the *cruel adversary* forces every deterministic online algorithm be at best k-competitive. (We use the term greedy (instead of stingy) for minimization as well as maximization problems.)

  The *DoubleCoverage* algorithm extends to tree metrics.

Bipartite Matching and Greediness

Consider deterministic online and priority algorithms for the unweighted one sided $n \times n$ bipartite matching problem.

The seminal Ranking algorithm is a greedy algorithm usually expressed as a randomized online algorithm. It can also be viewed as a deterministic ROM algorithm making greedy decision when we have random order arrivals. This random order ignores any information we might have of the names of the adjacent offline vertices for each online vertex and achieves a $1 - \frac{1}{e}$ competitive ratio. It seems reasonable to believe that we should be able to "intelligently" use some deterministic fixed or adaptive priority ordering to at least achieve the same approximation.

Along with many other results, Nicholas Pena in his MSc thesis/project showed that any determinstic adapative priority algorithn wil not be (asymptotically) better than $\frac{1}{2}$. More precisely, the algorithm will be forced to match at most $\lceil \frac{n+1}{2} \rceil$ online vertices while there is a perfect matching of the n online vertices. His thesis and our paper would be good to look at for anyone interested in max-sat and bipartite matching.

A sketch of the priority inapproxiamtion for bipartite graph matching

The adversary will construct a bipartite graph where every online vertex has degree $(n+1)/2$ (ignoring ceilings). This alone would seem to defeat fixed order priority as every online node looks the same. Since the algorithm knows the names of vertices, this is not quite a formal proof. We are interested in defeating deterministic adaptive order algorithms.

Using the notation in the paper, the adverary keeps track of the offline vertices M that have been matched as well as offline vertices U that can no longer be matched because of previous online matches, and offline vertices R that can no longer be matched because of online vertex rejections.

The adversary maintains the invariant that $|M| = |U|$ after each onine vertex has been processed. There will always be enough potential online vertices available so that whatever new online vertex v is chosen by the algorithm, and whether or not v is matched, the adversary will be able to maintain the invariant.

Returning to the paper that motivated me to decidee to talk about greediness. Alternativeky, making up an analogy when it doesn't exist.

I view fact that no adaptive prioirty algorithm can beat the naive greedy $\frac{1}{2}$ competitive ratio for deterministic online algiorithms whereas naively random ordering the inout itenms to be similar in spirit to the fact that greediness can fail to give good results. As I said, one would think that greedily trying to choose (within the limitations) of priority ordering) the order of arrivals shoudl be better than randomly choosing an orderimng.

The Prop1 fairness paper exhibits the same phenomena with respect to a fairness objective. I'll read what the abstract says.

Similarly no deterministic online algorithm can obtain Envy Freeness (EF) for additive valuations but naively tandomly allocationg an online item to any (offline) agent who has some positive value for the item does achieve EF ex-ante (i.e., in expectation).

The min cost (perfect) matching problem

Let $\mathcal{M} = (U, d)$ be a metric space with distance function d. Suppose we have $k \geq n$ known servers $s_1, s_2, \ldots, s_k$ located in U and a sequence of n online requests $r_1, r_2, \ldots, r_n$ in U. We need to match the k servers to the n requests with the objective being to minimize the total cost $\sum_{i=1}^{n} d(s_{\pi(i)}, r_i)$ where $s_{\pi(i)}$ is the server matched to r_i.

Note the similarity to the setting of the k server problem. But here we only move each server once to do the matching. The problem formulation also resembles one-sided maximum matching problem but now the edges are weighted (but restricted to be a metric) and this is a minimization problem.

The problem is most commonly formulated with $k = n$ and then called the perfect min cost matching problem.

The natural greedy algorithm is again to server a request with the closest server breaking ties in some determnistic way.

The greedy algorithm for min const perfect matching is pretty bad

Fact: For the real line metric space, the natural greedy algorithm is no better than $2^k - 1$ and this is the precise competitve ratio for the greedy algorithm.

Here is the argument for the lower bound inapproximation.
The adversary positions the servers as follows: $s_1 = 0$ and $s_i = 2^{i-1}$ for $2 \leq i \leq k$. Now consider a sequence of online requests $\{r_i\}$ where $r_i = 2^{i-1}$. If we break ties in favor of the server to the right, it is easy to see that the nearest server algorithm will serve r_i with server s_{i+1} for $1 \leq i \leq k-1$ and r_k will be served by s_1. This results in a total cost of $2^k - 1$. The offline optimal solution would serve r_i by s_i for $1 \leq i \leq k$ with cost 1. We can avoid the tie-breaking rule by letting $s_1 = -\epsilon$.

What is known about min cost matching problem?

The status for deterministic algorithms and arbitrary metric spaces is well determined. We use $k(= n)$ to be consistent with the literature.

- There is an algorithm (*Permutation*) that obtains the competitive ratio $2k - 1$ for every metric space. See Algorithm 7.7.1.
- For all $k \geq 2$, there is a metric space such that such that for all deterministic algorithms ALG, there is a sequence of k requests that forces ALG to have competitive ratio $2k - 1$.

The lower bound is for the star (tree) graph with one center and k leaves. The distance to the center from each leaf is 1 which induces a distance 2 between any pair of leaves. As in the k-server lower bound, we use a cruel adversary. We place the k servers on the leaves. The first request is for the center node. For each of the remaining requests, there is always one leaf that is not occupied. The cruel adversary requests the unoccupied leaf. The total cost to ALG is $1 + 2(k - 1) = 2k - 1$. After $k - 1$ leaaf nodes have been requested, there is exactly one leaf node that has not been requested. The optimum solution has cost 1 by serving the center node with the server that is on the one leaf node that was not requested.

The *Permutation* algorithm and a sketch of its competitive ratio

We first note that algorithm will be approximatng the cost of a maximum matching for any k and n. Furthermore, the algorithm does not need to know the metric space in advance.

Let $R_i = r_1, r_2, \ldots, r_i$ be the initial i requests; S_i will be the algorithm server matched to R_i. . Consider, a sequence of matchings matchings $M_0, M_1, \ldots, M_n$ that an offline OPT min cost solution matching can have between R_i and the servers in S. Here $M_0 = \varnothing$ and M_n is an optimum min cost maximum matching. We can follow the evolution of the S_i and the goal is to try to have the two sequences $\{M_i\}$ and $\{R_i\}$ be as close as possible under the online constaint of irrevocable decisions.

The algorithm will have the property that the set of servers matched to R_i by the final OPT is a permutation of the final algorithm servers matched to R_i. OPT can be seen as modifying M_{i-1} to M_i by as modifying the edges in an alternating path in $M_i \oplus M_{i-1}$starting at r_i. Here $M_i \oplus M_{i-1}$ is the symmetric difference between the edges in M_i and M_{i-1}.

Continuation of the sketch for algorthm *Permutation*

We would like to follow this evolution of the $\{M_i\}$ but cannot change previous edges in the algorithm's match. But this alternating path will end in a server s_j and then s_j will be matched to r_i.

Note: I am skipping the analysis that shows that there is a unique alternating path in $M_i \oplus M_{i-1}$ that we can define.

The competitive bound can be shown by an induction on i. Namely, letting A_i be the algorithm solution for the first i requests, we can show $w(A_i) \leq (2i-1)w(M_i)$ where $w(A_i)$ (respectively, $w(M_i)$) is the weight (i.e. total cost of edges) in A_i (respectively M_i).
$C = (M_i \cup \{(r_i, s_j)\} \oplus M_{i-1}$ is a simple alternating cycle. By the triangle inequality, it follows that
$w(r_i, s_j) \leq w(C \setminus \{(r_i, s_j)\}) \leq w(M_i) + w(M_{i-1}) \leq 2w(M_i)$ so that finally (using the induction hypothesis), we have:
$w(A_i) \leq w(A_{i-1}) + 2w(M_i) \leq ((2i-1)-1)w(M_{i-1}) + 2w(M_i) \leq (2i-1)w(M_i)$.

Randomized algorithms for min cost matching on arbitrary metric spaces

The situation for randomized algorithms is less clear for both arbitrary metric spaces and the line metric. In both cases we have a gap between the known upper and lower bounds.

For the min cost perfect matching with respect to arbitrary metric spaces, there is a randomized algorithm with competitiev ratio $O(\log^2 k)$. The bound uses the radnomized embedding of an arbitrary metric space onto an HST as defined in Chapter 5. The boud inproves to $O(\log k)$ if the metric space is a 2-HST.

An $\Omega(\log k)$ lower bound can be proven (perhaps surprisingly) for both the star graph metric and the uniform metric.

This leaves a gap between the $\Omega(\log k)$ lower bound and the $O(\log^6 k)$ competitive algiorithm.

The line metric and a "magical" parameterized algorithm *RobustMatching*(t)

The robust algorithm (Algorithm 7.7.6) in the text provides the following competitive bounds;

- For the line mytric when the parameter $t = 3$, the competitive ratio is $\Theta((\log k)$

  NOTE: This is a doubly exponential improvement over the greedy algorithm for the line metric.
- When $t = k^2 + 1$, the competitive ratio is $2k + 1$, the ratio of the optimal online algorithm *Permutation*. Like *Permutation*, the algorithm also uses an alternating path analysis.
- For the random order model, with $t = kH_k + 1$, the competitive ratio is $\Theta(\log k) + o(1)$

For randomnized algoeithms, the current best competitive ratio for the line metric is $O(\log k)$ and so no better than the deterministic ratio.

For the line metric, the current lower bound is $\Omega(\sqrt{k})$. This then leave a gap between the known upper and lower bounds.