CSC2420: Algorithm Design, Analysis and Theory
Fall 2023
An introductory (i.e. foundational) level graduate course.

Allan Borodin

September 26, 2023

Week 3

1. Announcements:
   - Two (multi-part) questions for Assignment 1 are now posted (and reposted) on the course web page. There was a typo on the due date. It is due Wednesday, October 11 at 11 AM. Hopefully all other typos have been fixed.
   - Consider signing up for piazza
   - I won't set fixed office hours at this time but feel free to drop by SF2303B or email for a fixed time to meet. I should be on campus every dauy this term with the exception of some Thursdays when I try to work at home.
2. Todays agenda.
   - Answer any questions on assignment.
   - A discussion about proving negative results ("lower bounds") for randomized online, priority (and other) algorithms.
   - Review and expand on definition of priority algorithms.
   - Extensions of the priority model.
   - The set cover problem
   - The set packing problem

Proving negative results for randomized algorithms

As I indicated, we will usually assume an *oblivious adversary* when discussing randomized algorithms.

Without revoking, the $\frac{1}{2}$ randomized competitive ratio for the proportional weight knapsack problem is an optimal ratio. That is, no randomized algorithm can achieve a $\frac{1}{2} + \epsilon$ competitive ratio for any $\epsilon > 0$.

This is Fact 3.4.4 in the text. Assume (without loss of generality) that the knapsack capacity is 1. We will only state one value for an input item since the value of an item is its size. Consider two input sequences $I_1 = (\epsilon)$ and $I_2 = (\epsilon, 1)$. Suppose the algorithm accepts the first ϵ item with probability p. Then the algorithm obtains the expected ratio $p \cdot \epsilon$ on the sequence I_1 while OPT has value ϵ so that the competitive ratio for I_1 is p. On sequence I_2, the algorithm obtains the ratio $p \cdot \epsilon + (1 - p) \cdot 1$ while OPT has value 1. This implies that the expected competitive ratio is (at most) $\rho(p) = \min\{p, p\epsilon + (1 - p)\}$ since the competitive ratio is a worst case (over all sequences) ratio. $\rho(p)$ is maximized when $p = \frac{1}{2}$ inwhich case it follows that $\rho(\frac{1}{2}) = \frac{1}{2}(1 + \epsilon)$.

Proving randomized negative results continued

Often (if not most often) we prove negative results about randomized algorithms using what is known as the Yao Minimax Principle. The Yao Principle can be applied in different applications (not just for competitive ratios). But for our purposes it says that we can obtain a negative result (e.g. that the competitive ratio is at most c for a maximization problem) for a randomized online algorithm by constructing a distribution D on online input sequences and proving that for any deterministic online algorithm ALG, we have $\frac{\mathbb{E}[ALG]}{\mathbb{E}[OPT]} \leq c$. For minimization problems (ands when ratios are greater that 1 for maximization problems), we need to shopw that $\frac{\mathbb{E}[OPT]}{\mathbb{E}[ALG]} \geq c$.

Here the expectation is taken with respect to the distribution D. We will illustrate the same negative result for the proportional knapsack problem (no revoking) using the Yao principle. The simple proof (due to Böckenhauer et al [TCS 2014]) is not an asymptotic ratio. The Yao based proof is (independently) due to Han et al [TCS 2015].

Note that Böckenhauer et al and Han et al express ratios to be ≥ 1

The proportional knapsack without revoklng: Using Yao principle

For a given n, consider the following distribution on n input sequences, each chosen with probability $\frac{1}{n}$:

$I_1 : \frac{1}{2} + \epsilon, \frac{1}{2} - \epsilon$

$I_2 : \frac{1}{2} + \epsilon, \frac{1}{2} + \frac{\epsilon}{2}, \frac{1}{2} - \frac{\epsilon}{2}$

$I_3 : \frac{1}{2} + \epsilon, \frac{1}{2} + \frac{\epsilon}{2}, \frac{1}{2} + \frac{\epsilon}{3}, \frac{1}{2} - \frac{\epsilon}{3}$

...

$I_n : \frac{1}{2} + \epsilon, \frac{1}{2} + \frac{\epsilon}{2}, \ldots \frac{1}{2} + \frac{\epsilon}{n}, \frac{1}{2} - \frac{\epsilon}{n}$

OPT has value 1 since it will accept the inputs $\frac{1}{2} + \frac{\epsilon}{k}, \frac{1}{2} - \frac{\epsilon}{k}$ for some k. Consider any detertministic online algorithm. Assume the algorithm first accepts the item with size $\frac{1}{2} + \frac{\epsilon}{j}$ for some $j \leq n$. Then the algorithm must reject all items with size $\frac{1}{2} + \frac{\epsilon}{\ell}$ for $\ell > j$.

The expected return of the algorithm is at most
$(\frac{1}{2} \cdot \frac{j-1}{n}) + (1 \cdot \frac{1}{n}) + (\frac{1}{2} + \frac{\epsilon}{j} \cdot \frac{n-j}{n}) \leq (\frac{1}{2} + \epsilon) \cdot \frac{n+1}{n}$
allowing us to conclude that the competitive ratio is no better than $\frac{1}{2} - \epsilon$ for arbitrarily small ϵ as $n \to \infty$.

Proportional knapsack with revoking: Uing the Yao Principle

We'll give one more impossibility proof which (currently) is worse than the best known algorithm which has competitive ratio $\frac{7}{10}$. Here we consider randomized online algorithms (with revoking) to show that no such algorithm can be better than $\frac{4}{5} + \epsilon$ competitive for the proportional knapsack problem.

Consider the following distribution on input sequences:
$I_1 : \frac{2}{3} + \epsilon, \frac{1}{3}, \frac{2}{3}$ with probability $\frac{1}{2}$
$I_2 : \frac{2}{3} + \epsilon, \frac{1}{3}$ with probability $\frac{1}{2}$

OPT hae expected value $1 \cdot \frac{1}{2} + (\frac{2}{3} + \epsilon) \cdot \frac{1}{2} = \frac{5}{6} + \epsilon$
Consider any deterministic algorithm.

If the algorithm rejects the second input, then it obtains value $\frac{2}{3} + \frac{\epsilon}{2}$
If the algorithm accepts the second input, it must discard the first input so that the algorithm has expected value $1 \cdot \frac{1}{2} + \frac{1}{3} \cdot \frac{1}{2} = \frac{2}{3}$

Therefore, the competitive ratio is at most $\frac{4}{5} + \epsilon$

Some additional results for randomized online algorithms with revoking for knapsack problems

We will move on from this topic but perhaps will return to it later.

- For proportional weights Han et al [TCS 2015] provide a $\frac{7}{10}$ competitive algorithm (with revoking) that is not that simple nor easy to analyze.
- For the general knapsack problem, we have seen a $\frac{1}{2}$ randomized algorithm with revoking. Han et al provide a negative $\frac{1}{1+1/e} \approx .731$ result using the Yao Principle.
- Online algorithms with revoking but with a cost to revoke items has been studied (sometimes called "buyback" problems). This problem is motivated and studied by Babaioff et al [EC 2009] in the context of selling ad campaigns. They proved the lower $\frac{4}{5}$ negative result (without costs) for the general knapsack whereas Han et al improve this negative result by showing that it holds for proportional weights.

The priority algorithm model and variants

As part of our discussion of greedy (and greedy-like) algorithms, I want to present the priority algorithm model and how it can be extended in (conceptually) simple ways to go beyond the power of the priority model.

- What is the intuitive nature of a greedy algorithm as exemplified by the CSC 373 algorithms we mentioned? With the exception of Huffman coding (which we can also deal with), like online algorithms, all these algorithms consider one input item *in some well defined ordering of the items* in each iteration and make an irrevocable "greedy" decision about that item..
- We are then already assuming that the class of search/optimization problems we are dealing with can be viewed as making a decision D_k about each input item I_k (e.g. on what machine to schedule job I_k in the makespan case) such that $\{(I_1, D_1), \ldots, (I_n, D_n)\}$ constitutes a feasible solution.

Priority model continued

- Note: that a problem is only fully specified when we say how input items are represented. (This is usually implicit in an online algorithm.)
- We mentioned that a "non-greedy" online algorithm for identical machine makespan can improve the competitive ratio; that is, the algorithm does not always place a job on the (or a) least loaded machine (i.e. does not make a greedy or locally optimal decision in each iteration). It isn't always obvious if or how to define a "greedy" decision but for many problems the definition of greedy can be informally phrased as "live for today" (i.e. assume the current input item could be the last item) so that the decision should be an optimal decision given the current state of the computation.

Greedy decisions and priority algorithms continued

- For example, in the knapsack problem, a greedy decision always takes an input if it fits within the knapsack constraint and in the makespan problem, a greedy decision always schedules a job on some machine so as to minimize the increase in the makespan. (This is somewhat more general than saying it must place the item on the least loaded machine.)
- If we do not insist on greediness, then priority algorithms would best have been called myopic algorithms.
- We have both fixed order priority algorithms (e.g. unweighted interval scheduling and LPT makespan) and adaptive order priority algorithms (e.g. the set cover greedy algorithm and Prim's MST algorithm).
- The key concept is to indicate how the algorithm chooses the order in which input items are considered. We cannot allow the algorithm to choose say "an optimal ordering".
- We might be tempted to say that the ordering has to be determined in polynomial time but that gets us into the "tarpit" of trying to prove what can and can't be done in (say) polynomial time.

The priority model definition

- We take an information theoretic viewpoint in defining the orderings we allow.
- Lets first consider deterministic fixed order priority algorithms. Since I am using this framework mainly to argue negative results (e.g. a priority algorithm for the given problem cannot achieve a stated approximation ratio), we will view the semantics of the model as a game between the algorithm and an adversary.
- Initially there is some (possibly infinite) set $\mathcal{J}$ of potential inputs. The algorithm chooses a total ordering π on $\mathcal{J}$. Then the adversary selects a subset $\mathcal{I} \subset \mathcal{J}$ of actual inputs so that $\mathcal{I}$ becomes the input to the priority algorithm. The input items $I_1, \ldots, I_n$ are ordered according to π.
- In iteration k for $1 \leq k \leq n$, the algorithm considers input item I_k and based on this input and all previous inputs and decisions (i.e. based on the current state of the computation) the algorithm makes an irrevocable decision D_k about this input item.

The fixed (order) priority algorithm template

$\mathcal{J}$ is the set of all possible input items
Decide on a total ordering π of $\mathcal{J}$
Let $\mathcal{I} \subset \mathcal{J}$ be the input instance
$S := \varnothing$ % S is the set of items already seen
$i := 0$ % $i = |S|$
while $\mathcal{I} \setminus S \neq \varnothing$ **do**
    $i := i + 1$
    $\mathcal{I} := \mathcal{I} \setminus S$
    $I_i := \min_\pi \{I \in \mathcal{I}\}$
    make an irrevocable decision D_i concerning I_i
    $S := S \cup \{I_i\}$
end

Figure: The template for a fixed priority algorithm

Some comments on the priority model

- A special (but usual) case is that π is determined by a function $f : \mathcal{J} \to \Re$ and and then ordering the set of actual input items by increasing (or decreasing) values $f()$. (We can break ties by say using the input identifier of the item to provide a total ordering of the input set.) N.B. We make no assumption on the complexity or even the computability of the ordering π or function f.
- NOTE: Online algorithms are fixed order priority algorithms where the ordering is given *adversarially*; that is, the items are ordered by the input identifier of the item.
- As stated we do not give the algorithm any additional information other than what it can learn as it gradually sees the input sequence.
- However, we can allow priority algorithms to be given some (hopefully easily computed) global information such as the number of input items, or say in the case of the makespan problem the minimum and/or maximium processing time (load) of any input item. (Some inapproximation results can be easily modified to allow such global information.)

The adaptive priority model template

$\mathcal{J}$ is the set of all possible input items
$\mathcal{I}$ is the input instance
$S := \varnothing$ % S is the set of items already considered
$i := 0$ % $i = |S|$
while $\mathcal{I} \setminus S \neq \varnothing$ **do**
    $i := i + 1$
    decide on a total ordering π_i of $\mathcal{J}$
    $\mathcal{I} := \mathcal{I} \setminus S$
    $I_i := \min_{\leq_{\pi_i}} \{I \in \mathcal{I}\}$
    make an irrevocable decision D_i concerning I_i
    $S := S \cup \{I_i\}$
    $\mathcal{J} := \mathcal{J} \setminus \{I : I \leq_{\pi_i} I_i\}$
    % some items cannot be in input set
end

Figure: The template for an adaptive priority algorithm

Some deterministic priority algorithm approximations and inapproximations

Once we have a precise model, we can then argue that certain approximation bounds are not possible within this model. Such inapproximation results have been established with respect to priority algorithms for a number of problems but for some problems much better approximations can be established using extensions of the model.

- The Greedy algorithm for unweighted interval selection is optimal. Extending the problem to the *NP*-hard job interval selection problem, the greedy algorithm is a $\frac{1}{2}$ approximation.
- For weighted interval selection with arbitrary weighted values (resp. for proportional weights $v_j = |f_j - s_j|$), no priority algorithm can achieve a constant approximation (respectively, a $\frac{1}{3} + \epsilon$ -approximation).

Charging arguments for the unweighted and proportionally weighted interval selection problem.

Let first consider the unweighted interval selection problem where each input item is an interval $I_j = [s_i, f_i)$ with s_i (resp. f_i) the starting time (finishing time) of the interval. The greedy algorithm sorts the intervals so that $f_1 \leq f_2 \ldots \leq f_n$. We will construct a 1-1 charging function $h : OPT \rightarrow Greedy$ (which implies $|OPT| \leq |Greedy|$) as follows:

We charge any interval in $OPT \cap Greedy$ to itself. So now we need to charge an interval I_j in $OPT \setminus Greedy$ to an interval selected by the greedy algorithm. The function h will charge I_j to the leftmost interval I_i with which it conflicts. Since $I_i \notin Greedy$, it must be that I_i exists and $f_i \leq f_j$ or else Greedy would have taken I_j. This shows that h is well defined. To show that h is 1-1 is well defined, we need to show that there cannot be two OPT intervals being charged to I_i. But any I_k with $k \geq i$ must intersect I_i at f_i which means that I_j and I_k would intersect.

The job interval scheduling problem and the greedy algorithm

In the NP-hard job interval scheduling problem, every interval belongs to exactly one job class and now a feasible solution means that accepted intervals do not intersect and at most one interval from any job class is accepted.

We again sort the intervals so $f_1 \leq f_2 \ldots \leq f_n$. We use essentially the same charging argumnent except now the charging function h is a 2-1 function which implies that $|OPT| \leq 2|Greedy|$. That is an interval $I_j \in OPT \setminus Greedy$ either intersects an interval I_i or I_j and I_i are in the same job class.

A charging argument for the proportionally weighted interval selection problem

Here we assumne that the weight (i.e., the profit) of a scheduled interval is equal to its length $f_i - s_i$. The claim is that LPT (Longest Processing Time first) provides a $\frac{1}{3}$ approximation. Moreover, this is the optimal approximation for any priority algorithm; more specifically $\frac{1}{3} + \epsilon$ for any $\epsilon > 0$.

We use a charging argument to establish the positive result. Namely, we provide a charging function $f; OPT \to LPT$ such that f charges at most 3 times the weight of intervals in OPT to an interval in LPT.

Once again if $I \in OPT \cap LPT$, then charge I to itself. Otherwise consider an interval $I \in OPT \setminus LPT$. Charge I to the interval I' in LPT that intersects with I having the earliest finishing time. (Here we could charge I to I' with the earliest starting time or really charge in any way that provides a unique I'.). We know that such an I' must exist or else LPT (being greedy) would have taken the interval I.

Completing the charging argument for proprtionally weighted interval scheduling

I can intersect I' by overlapping at an endpoint s_i and/or f_i of I' or it can be that I is contained in I'. We can deal with each of these cases.

- I subsumes I'. This can't happen since the the length (and therefore profit) of I would be more than I' and hence LPT would have taken it.
- I is subsumed by I'. The sum of the lengths of all such intervals I is at most the length of I'.
- I' intersects at the start time s_i (or finishing time f_i) of I'. There can be at most one interval in OPT intesecting at s_i (resp. f_i). By the LPT ordering, $|I| \leq |I'|$.

The negative result for proportional weighted interval selection fn priority algorithms

For the negative result consider input instances where we have long jobs (as depicted in Figure 1) and enough short jobs to fill up any long job. Each small job has negligible size compared to the interval that subsumes it. Any priority algorithm must take the first interval I it decides to process or else the adversary stops the input sequence and OPT takes I.

If it is a small job, the adversary just gives the long the long job subsuming it.

If I is a large job $J_k \neq J_1$, the adversary gives all the small jobs within I and the two adjacent intervals.

If I is a large job $J_k = J_1$, the adversary gives all the small jobs within I and the one adjacent interval.

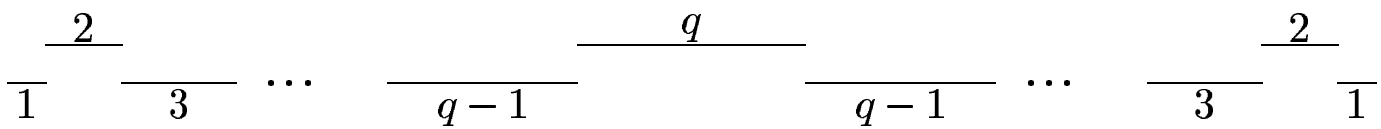

Figure 1: The "long jobs" from the worst case sequence for any priority algorithm for $m = 1$.

Some more deterministic priority algorithm approximations and inapproximations

- For the knapsack problem, no deterministic adaptive priority algorithm can achieve a constant approximation. However, there are randomized competiitve competitive algorithms. This is relevant to the second assignment question.
- For the set cover problem, the "natural" greedy algorithm obtains the ratio $H_n \approx \ln n$ and this is essentially the best priority algorithm and the best approximation by any polynomial time algorithm assuming a reasonable complexity conjecture.
  What is the natural greedy algorithm?

Some more deterministic priority algorithm approximations and inapproximations

- For the knapsack problem, no deterministic adaptive priority algorithm can achieve a constant approximation. However, there are randomized competiitve competitive algorithms. This is relevant to the second assignment question.
- For the set cover problem, the "natural" greedy algorithm obtains the ratio $H_n \approx \ln n$ and this is essentially the best priority algorithm and the best approximation by any polynomial time algorithm assuming a reasonable complexity conjecture.
  What is the natural greedy algorithm? In the j^{th} iteration, we choose the set that minimizes $\frac{c_i}{r_i}$ where c_i (respectively, r_i) is the cost of set S_i (respectively, the number of remaining uncovered items in S_i at the start of the j^{th} iteration. This is clearly, an adaptive priority algorithm.

And more deterministic priority algorithm approximations and inapproximations

- See Section 16.4.3 in the text for the unweighted set cover problem in which case the natural greedy algorithm sorts the sets so that $c_1 \leq c_2 \ldots \leq c_m$. Here we show the positive H_n result and a priority algorithm $\frac{\log n - 1}{2}$ inapproximation.
- The analysis of the natural greedy algorithm for the weighted set cover problem is given in the Vazirani Approximation Algorithms text.
- As previously mentioned, for deterministic fixed order priority algorithms, there is an $\Omega(\log m / \log \log m)$ inapproximation bound for the makespan problem in the restricted machines model. The competitive ratio for the naturaal greedy algorithm is $\log m$ so that a gap "small" gap remains for fixed priority and we don not have an adaptive priority inapproximation.

More on provable limitations of the priority model

The above mentioned inapproximations are with respect to deterministic priority algorithms. For a deterministic adaptive algorithm, the game between an algorithm and an adversary can conceptually be naturally viewed an alternating sequence of actions with respect to some initial set of all possible input items.

- The adversary eliminates some possible input items
- The algorithm makes a decision for the item with highest priority and chooses a new ordering for all possible remaining input items.

However, we note that for deterministic algorithms, since the adversary knows precisely what the algorithm will do in each iteation, it could initially set the input $\mathcal{I}$ once the algorithm is known.

More on provable limitations of the priority model

The above mentioned inapproximations are with respect to deterministic priority algorithms. For a deterministic adaptive algorithm, the game between an algorithm and an adversary can conceptually be naturally viewed an alternating sequence of actions with respect to some initial set of all possible input items.

- The adversary eliminates some possible input items
- The algorithm makes a decision for the item with highest priority and chooses a new ordering for all possible remaining input items.

However, we note that for deterministic algorithms, since the adversary knows precisely what the algorithm will do in each iteation, it could initially set the input $\mathcal{I}$ once the algorithm is known.

For randomized algorithms, we again (as for online algorithms) usually assume an oblivious adversary. And again, we often use the Yao principle to prove randomized results.

Extensions of the priority order model

We already implicitly suggested some extensions of the basic priority model (e.g., the basic one-pass, irrevocable decisions model) The following online or priority algorithm extensions can be made precise:

- Decisions can be *revocable* to some limited extent or at some cost. For example, we know that in the basic priority model we cannot achieve a constant approximation for weighted interval scheduling. However, if we are allowed to permanently discard previously accepted intervals , then we can achieve a $\frac{1}{4}$-approximation. (but provably cannot achieve optimality unlike unweighted interval selection).
- While the knapsack problem cannot be approximated (by a deterministic priority algorithm) to within any constant, we can achieve a $\frac{1}{2}$-approximation (respectively, a $\frac{1}{4}$-approximation) by taking the maximum of 2 greedy algorithms (by randomly choosing between two greedy algorithms). More generally we can consider some "small" number k of priority (or online) algorithms and take the best result amongst these k algorithms (e.g., the partial enumeration greedy algorithms for the makespan and knapsack problems are examples).

Extensions of the priority order model continued

- Closely related to the "best of k online (priority)" algorithms is the concept of online (priority) algoirthms with "advice". There are two advice models, a model where one measures the maximum number of advice bits per input item, and a model where we are given some number ℓ of advice bits at the start of the computation. The latter model is what I will usually mean by "online (priority) with advice." Online with ℓ advice bits is equivalent to the max of $k = 2^\ell$ online each priority algorithm operating independent of the other. (priority) model.

Extensions of the priority order model continued

- Closely related to the "best of k online (priority)" algorithms is the concept of online (priority) algoirthms with "advice". There are two advice models, a model where one measures the maximum number of advice bits per input item, and a model where we are given some number ℓ of advice bits at the start of the computation. The latter model is what I will usually mean by "online (priority) with advice." Online with ℓ advice bits is equivalent to the max of $k = 2^\ell$ online each priority algorithm operating independent of the other. (priority) model.

  NOTE: This model is a very permissive in that the advice bits can be a function of the entire input. Of course, in practice we want these advice bits to be "easily determined" (e.g., the number of input items, or the ratio of the largest to smallest weight/value) but in keeping with the information theoretic perspective of online and priority algorithms, one doesn't impose any such restriction.

Online algorithms with ML advice

Relatively recently, the advice model has gained new attention in the form of "online algorithms with ML advice". Of course, saying ML attracts attention and the more accurate (and also used) terminology is "online algorithms with predictions". And more generally, this can be seen as "online algorithms with untrusted advice"[1].

The perspective in online algorithms with predictions is the tradeoff between the amount of error in the predictions and performance (e.g., the competitive ratio). The predictions might come from experience of an ML algorithm learning from thousands of trials or it can come from a prediction that the inputs are coming from a known distribution, or really any prediction about the data.

[1] This latter termninology was defined to study the number of untrusted bits of advice vs performance but the current focus is on prediction error vs performance.

Online algorithms with predicitions

The goal is to exploit predictions to obtain an algorithm that informally is "robust" (in the sense of the algorithm not performing too badly if the prediction is arbitrarily bad and is "consistent" in the sense that the algorithm's performance is much better than what can be done without predictions if the prediction is accurate or near accurate (i.e., the "error" is small).

There are different ways to formulate and quantify these robust and consistency requirements; that is, defining a "useful" class of predictions and how to measuren the error of the prediction. Not surprisingly, there is usually a tradeoff between robustness and consistency.

The goal is often to exhibit Pareto optimal algorithms. In the case of say predicitions we consider an algorithm's outcome as a point in terms of the (robust,consistency) values of the algorithm and Pareto optimal here means that we cannot simultaneously improve both the robustness and the consistency.

More on advice and predicitions

- The concepts of trusted and untrusted advice, and predictions can be applied to any class of algorithms.
- As mentioned, we can consider an online (resp. priority) algorithhm with say k trusted bits of advice as a parallel model where the algorithm initially forks into 2^k online (resp. parallel) algorithms and takes the best leaf in the resulting 2^k width tree. There are more general parallel priority based models than "best of k" algorithms. Namely, parallel algorithms could be spawning or aborting threads (as in the pBT model to be discussed later which serves as a model for branch and bound algorithms and very simple dynmaic programming).
- Any lower bound on the number of advice bits needed to obtaina c-approximation immediately becomes a lower bound on the number of random bits needed to achieve a c-approximation. Moreover, under a mild assumption, a randomized algorithm with ratio c can be converted to a deterministic $(1+\epsilon)\cdot c$-competitive ratio using $O(\log n)$ advice bits. The countrapositive of this result (for proving randomized inapproximations) is perhaps the primary use of advice.

Multipass algorithms

- Another model that provides improved results is to allow multiple passes (over the input items) rather than just one pass.
- This is not a well studied model but there are two relatively new noteworthy results that we will be discussing:
  1. There is deterministic 3/4 approximation for weighted Max-Sat that is achieved by two "online passes" (i.e., the input sequence is determined by an adversary) over the input sequence whereas there is evidence that no one pass deterministic online or priority algorithm can acheive this ratio.
  2. There is a $\frac{3}{5}$ approximation for biparitie matching that is achieved by two online passes whereas no deterministic online or priority algorithm can do asymptotically better than a $\frac{1}{2}$ approximation.
- It is not clear how best to formalize these multi-pass algorithms. Why?

Multipass algorithms

- Another model that provides improved results is to allow multiple passes (over the input items) rather than just one pass.
- This is not a well studied model but there are two relatively new noteworthy results that we will be discussing:
  1. There is deterministic 3/4 approximation for weighted Max-Sat that is achieved by two "online passes" (i.e., the input sequence is determined by an adversary) over the input sequence whereas there is evidence that no one pass deterministic online or priority algorithm can acheive this ratio.
  2. There is a $\frac{3}{5}$ approximation for biparitie matching that is achieved by two online passes whereas no deterministic online or priority algorithm can do asymptotically better than a $\frac{1}{2}$ approximation.
- It is not clear how best to formalize these multi-pass algorithms. Why? What information should we be allowed to convey between passes?

Greedy algorithms for the set packing problem

One of the new areas in theoretical computer science is algorithmic game theory and mechanism design and, in particular, auctions including what are known as *combinatorial auctions*. The underlying combinatorial problem in such auctions is the set packing problem.

The set packing problem

We are given n subsets $S_1, \ldots, S_n$ from a universe U of size m. In the weighted case, each subset S_i has a weight w_i. The goal is to choose a disjoint subcollection $\mathcal{S}$ of the subsets so as to maximize $\sum_{S_i \in \mathcal{S}} w_i$. In the s-set packing problem we have $|S_i| \leq s$ for all i.

- This is a well studied problem and by reduction from the max clique problem, there is an $m^{\frac{1}{2}-\epsilon}$ hardness of approximation assuming $NP \neq ZPP$. For s-set packing with constant $s \geq 3$, there is an $\Omega(s/\log s)$ hardness of approximation assuming $P \neq NP$.
- We will consider two "natural" greedy algorithms for the s-set packing problem and a non obvious greedy algorithm for the set packing problem. These greedy algorithms are all fixed order priority.