CSC2420: Algorithm Design, Analysis and Theory
Fall 2420
An introductory (i.e. foundational) level graduate course.

Allan Borodin

September 28, 2022

Week 2 of Course but week 3 of term

During this course, we will be seeing a number of different problems and different algorithms. However, what I think is important is to keep in mind some general themes that we need to consider in a course on "Algorithm Design, Analysis and Theory". In particular, we will hopefully keep in mind:

- Classes of algorithms (as often emphasized at least informally in undergrad texts)
- Basic problems solved in a variety of ways with different properties
- Tradeoffs
- Relation between problems; extensions, reductions
- Methods of analysis
- The power of randomization

We won't worry about implementation but rather algorithmic approaches

Todays agenda

My intention was to continue with introduing some basic algorithmic paradigms along with discussing some new problems, namely the knapsack problem and the set packing problem. But because I have to miss next week, Calum MacRury will give the lecture on October 5. I want to start today with just a quick introduction to online bipartite matching which will be the topic of Calum's class.

Then we will start considering the following topics:

- Partial enummeration greedy; combining brute force search and greedy
- Additional comments on the makespan problem and alternative machine models for makespan and other scheduling problems.
- The makespan problem with precedence; scheduling anomolies
- The knapsack problem and dynamic programming
- The priority model for greedy-like algorithms
- Extensions of the online and priority model
- Online and offline algorithms with advice; ML advice
- The set packing problem.

The general graph matching problem

I just want to briefly introduce the online bipartite matching problem as it will be the topic of Calum's class next week. (When we say "graph" we mean an undirected graph.)

In a graph $G = (V, E)$, a match is $M \subseteq E$ such that for every vertex v the degree of $v \in M$ is at most 1. In an unweighted graph, the goal is to maximize the number of edges in the match. In an edge weighted graph, the objective is to maximize the sum of weights of edges in the match.

The matching problem is one of the most fundamental combinatorial problems in terms of its importance in applications, and the theory and algorithms that have evolved for solving matching problems. See start of video (https://www.youtube.com/watch?v=D48ci95qHQY) by Vijay Vazirani. In terms of offline algorithms, one of the great results of the 1960s was Jack Edmonds' polynomial time (not online) algorithm for optimally solving the unweighted and weighted matching problem for general graphs.

Recent work pertains to finding the best running times.

Matching in bipartite graphs

In a bipartite graph $G = (U, V, E)$, the vertices are partitioned into two sets U, V and $E \subseteq U \times V$.

While the asymptotic running times for solving the unweighted and weighted bipartite matching is not better than for general graphs (as far as I know), there are conceptually much simpler (in my opinion) optimal algorithms for the bipartite case which might also be the fastest algorithm "in practice".

In terms of online algorithms, the most common online model is one where one side of the vertices (say, V) are known in advance and the other side of vertices arrive online along with their adjacent edges.
Warning: I think most articles have V as the online vertices but I tend to use U; also sometimes articles will talk about (say) online vertices L (left side) and offline nodes R.

The online bipartite matching problems

The online version of bipartite matching is especially important both in terms of its impact on the topic of online algorithms and as the start of a substantial line of work concerning variants that are used for online auctions where say items (e.g. queries) arrive online and are assigned to offline buyers (advertisers).

As you will see next week, for the basic unweighted problem, the "natural" greedy algorithm is a $\frac{1}{2}$ approximation and this is optimal for any online algorithm. What is the natural greedy algorithm?

There is a randomized online algorithm that achieves competitve ratio $1 - \frac{1}{e} \approx .632$. This is called the Ranking algorithm due to Karp Vazirani and Vazirani [1990]. This is the optimal competitive ratio for any online randomized algorithm (in the vertex arrival model). Since then there have been a number of alternative proofs for this algorithm including the primal dual algorithm by Devanur et al (in 2013) which Calum will be using.

The primal dual approach for bipartite matching

Almost all of the optimization problems we study can be formulated as IPs (integer programs). These IPs can be relaxed to LPs (linear programs) whose solutions provide optimal fractional solutions.

The fractional solution can often be converted by some kind of "rounding" into an approximaten integral solution.

Alternatvely, in the primal dual apporach we use the dual of the LP formualtion to provide a way in which we can sequentially set the primal variables.

Primal

$$\begin{array}{lll} \text{minimize} & \sum_{u_i \in U} \beta_i + \sum_{v_j \in V} \alpha_j & \\ \text{subj. to} & \beta_i + \alpha_j \geq 1 & (u_i, v_j) \in E \\ & \beta_i \geq 0 & u_i \in U \\ & \alpha_j \geq 0 & v_j \in V \end{array}$$

Dual

$$\begin{array}{lll} \text{maximize} & \sum_{(u_i, v_j) \in E} x_{i,j} & \\ \text{subj. to} & \sum_{j:(u_i, v_j) \in E} x_{i,j} \leq 1 & u_i \in U \\ & \sum_{i:(u_i, v_j) \in E} x_{i,j} \leq 1 & v_j \in V \\ & x_{i,j} \geq 0 & (u_i, v_j) \in E \end{array}$$

[The dual program is a formulation of bipartite matching]

Online bipartite matching continued

Ranking can alternatively be viewed as a deterministic algorothm in the random order arrival model. It is not known if $1 - \frac{1}{e}$ is the optimal competitive ratio for deterministic bipartite matching in the ROM model. There is a better randomized algorithm in the ROM model.

The Ranking algorithm can be extended to provide a $1 - \frac{1}{e}$ algorithm for offline vertex weighted graphs.

There is no bounded competitive ratio for edge weighted bipartite matching; the ratio must necessarily depend on the weights of the edges.

There are edge weighted variants of bipartite matching that do have constant competitive ratios (but still not optimal).

The AdWords and Display Ads variants.

In AdWords, edge weights represent bids b_{ij} by an offline advertsiser v_j for a query impression u_i. Each offline advertiser v_j has a budget B_j. The objective is to find a matching M so as to maximize the following objective: $\sum_j \min\{B_j, \sum_{i:e_{ij}\in M} b_{ij}\}$. That is each advertiser does not generate more revenue than its budget.

In the Display Ads problem (with free disposal), each offline v_j has an integer capacity C_j and we only insist on a 1-sided matching where each online vertex can have degree at most one and each offline v_j can have any degree. However, the vertex (advertiser) v_j (or the auctioneer) only gets credit for the most valuable C_j weighted items assigned to it. That is, the objective is to find a 1-sided matching M with some subset S_j of edges assigned to vertex v_j such that $|S_j \leq |C_j|$ so as to maximize the objective: $\sum_j \sum_{e_{ij}\in S_j} b_{ij}$.

For AdWords and Display Ads problems, the "natural greedy" algorithm is $1/2$ competitive and this improves to $1 - \frac{1}{e}$ for "small bids" and (respectively) large capacities. What is the natural greedy algorithm?

Partial Enumeration Greedy: back to intended agenda

- Not sure if I mentioned that the makespan problem is an NP-hard optimization problem but there are (offline) algorithms that provide good worst case approximations.
- Combining the LPT idea with a brute force approach improves the approximation ratio but at a significant increase in time complexity.
- I call such an algorithm a "partial enumeration greedy" algorithm.

Optimally schedule the largest k jobs (for $0 \leq k \leq n$) and then greedily schedule the remaining jobs (in any order).

- The algorithm has approximation ratio no worse than $\left(1 + \frac{1-\frac{1}{m}}{1+\lfloor k/m \rfloor}\right)$.
- Graham also shows that this bound is tight for $k \equiv 0 \mod m$.
- The running time is $O(m^k + n \log n)$.
- Setting $k = \frac{1-\epsilon}{\epsilon} m$ gives a ratio of at most $(1 + \epsilon)$ so that *for any fixed* m, this is a PTAS (polynomial time approximation scheme). with time $O(m^{m/\epsilon} + n \log n)$.

Makespan: Some additional comments

- There are many refinements and variants of the makespan problem.
- There was significant interest in the best competitive ratio (in the online setting) that can be achieved for the identical machines makespan problem.
- The online greedy gives the best online ratio for m $= 2,3$ but better bounds are known for $m \geq 4$. For arbitrary m, as far as I know, following a series of previous results, the best known approximation ratio is 1.9201 (Fleischer and Wahl) and there is 1.88 inapproximation bound (Rudin). **Basic idea:** leave some room for a possible large job; this forces the online algorithm to be non-greedy in some sense but still within the online model.
- Randomization (and random order arrivals) can provide somewhat better competitive ratios. We will see many examples where randomization is very useful.
- Makespan has been actively studied with respect to three other machine models.

The uniformly related machine model

- Each machine i has a speed s_i
- As in the identical machines model, job J_j is described by a processing time or load p_j.
- The processing time to schedule job J_j on machine i is p_j/s_i.
- There is an online algorithm that achieves a constant competitive ratio.
- I think the best known online ratio is 5.828 due to Berman et al following the first constant ratio by Aspnes et al.
- Ebenlendr and Sgall establish an online inapproximation of 2.564 following the 2.438 inapproximation of Berman et al.

The restricted machines model

- Every job J_j is described by a pair (p_j, S_j) where $S_j \subseteq \{1, \ldots, m\}$ is the set of machines on which J_j can be scheduled.
- This (and the next model) have been the focus of a number of papers (for both online and offline) and there has been some relatively recent progress in the offline restricted machines case.
- Even for the case of two allowable machines per job (i.e. the *graph orientation problem*), this is an interesting problem and we will look at some recent work later.
- Azar et al show that $\log_2(m)$ (resp. $\ln(m)$) is (up to ± 1) the best competitive ratio for deterministic (resp. randomized) online algorithms with the upper bounds obtained by the "natural greedy algorithm".
- It is not known if there is an offline greedy-like algorithm for this problem that achieves a constant approximation ratio. Regev [IPL 2002] shows an $\Omega(\frac{\log m}{\log \log m})$ inapproximation for "fixed order priority algorithms" for the restricted case when every job has 2 allowable machines.

The unrelated machines model

- This is the most general of the makespan machine models.
- Now a job J_j is represented by a vector $(p_{j,1}, \ldots, p_{j,m})$ where $p_{j,i}$ is the time to process job J_j on machine i.
- A classic result of Lenstra, Shmoys and Tardos [1990] shows how to solve the (offline) makespan problem in the unrelated machine model with approximation ratio 2 using LP rounding.
- There is an online algorithm with approximation $O(\log m)$. Currently, this is the best approximation known for greedy-like (e.g. priority) algorithms even for the restricted machines model although there has been some progress made in this regard (which we will discuss later).
- NOTE: All statements about what we will do later should be understood as intentions and not promises.

Makespan with precedence constraints; how much should we trust our intuition

Graham also considered the makespan problem on identical machines for jobs satisfying a precedence constraint. Suppose $\prec$ is a partial ordering on jobs meaning that if $J_i \prec J_k$ then J_i must complete before J_k can be started. Assuming jobs are ordered so as to respect the partial order (i.e., can be reordered within the priority model) Graham showed that the ratio $2 - \frac{1}{m}$ is achieved by "the natural greedy algorithm", call it $\mathcal{G}_{\prec}$.

Makespan with precedence constraints; how much should we trust our intuition

Graham also considered the makespan problem on identical machines for jobs satisfying a precedence constraint. Suppose $\prec$ is a partial ordering on jobs meaning that if $J_i \prec J_k$ then J_i must complete before J_k can be started. Assuming jobs are ordered so as to respect the partial order (i.e., can be reordered within the priority model) Graham showed that the ratio $2 - \frac{1}{m}$ is achieved by "the natural greedy algorithm", call it $\mathcal{G}_{\prec}$.

Graham's 1969 paper is entitled "Bounds on Multiprocessing Timing Anomalies" pointing out some very non-intuitive anomalies that can occur.

Consider $\mathcal{G}_{\prec}$ and suppose we have a given an input instance of the *makespan with precedence* problem. Which of the following should never lead to an increase in the makepan objective for the instance?

- Relaxing the precedence $\prec$
- Decreasing the processing time of some jobs
- Adding more machines

Makespan with precedence constraints; how much should we trust our intuition

Graham also considered the makespan problem on identical machines for jobs satisfying a precedence constraint. Suppose $\prec$ is a partial ordering on jobs meaning that if $J_i \prec J_k$ then J_i must complete before J_k can be started. Assuming jobs are ordered so as to respect the partial order (i.e., can be reordered within the priority model) Graham showed that the ratio $2 - \frac{1}{m}$ is achieved by "the natural greedy algorithm", call it $\mathcal{G}_{\prec}$.

Graham's 1969 paper is entitled "Bounds on Multiprocessing Timing Anomalies" pointing out some very non-intuitive anomalies that can occur.

Consider $\mathcal{G}_{\prec}$ and suppose we have a given an input instance of the *makespan with precedence* problem. Which of the following should never lead to an increase in the makepan objective for the instance?

- Relaxing the precedence $\prec$
- Decreasing the processing time of some jobs
- Adding more machines

In fact, all of these changes could increase the makespan value.

The knapsack problem

The $\{0,1\}$ knapsack problem

- Input: Knapsack size capacity C and n items $\mathcal{I} = \{I_1, \ldots, I_n\}$ where $I_j = (v_j, s_j)$ with v_j (resp. s_j) the profit value (resp. size) of item I_j.
- Output: A feasible subset $S \subseteq \{1, \ldots, n\}$ satisfying $\sum_{j \in S} s_j \leq C$ so as to maximize $V(S) = \sum_{j \in S} v_j$.

Note: I would prefer to use approximation ratios $r \geq 1$ (so that we can talk unambiguously about upper and lower bounds on the ratio) but many people use approximation ratios $\rho \leq 1$ for maximization problems; i.e. $ALG \geq \rho OPT$. For certain topics, this is the convention.

- One can show that the most natural greedy methods (sort by non-increasing profit densities $\frac{v_j}{s_j}$, sort by non-increasing profits v_j, sort by non-decreasing size s_j) will not yield any constant ratio.

The knapsack problem

The {0,1} knapsack problem

- Input: Knapsack size capacity C and n items $\mathcal{I} = \{I_1, \ldots, I_n\}$ where $I_j = (v_j, s_j)$ with v_j (resp. s_j) the profit value (resp. size) of item I_j.
- Output: A feasible subset $S \subseteq \{1, \ldots, n\}$ satisfying $\sum_{j \in S} s_j \leq C$ so as to maximize $V(S) = \sum_{j \in S} v_j$.

Note: I would prefer to use approximation ratios $r \geq 1$ (so that we can talk unambiguously about upper and lower bounds on the ratio) but many people use approximation ratios $\rho \leq 1$ for maximization problems; i.e. $ALG \geq \rho OPT$. For certain topics, this is the convention.

- One can show that the most natural greedy methods (sort by non-increasing profit densities $\frac{v_j}{s_j}$, sort by non-increasing profits v_j, sort by non-decreasing size s_j) will not yield any constant ratio.
- Can you think of nemesis sequences for these three greedy methods?
- What other orderings could you imagine?

Can there be any online algorithm for the knapsack problem?

Note that the knapsack problem can be called the budget problem; that is, the size can be viewed as a budget.
Given the importance of the knapsack problem , it has been studied with respect to many different online and offline models. We briefly mention a few known results:

- The problem is weakly NP complete even if all values are proportional; that is $v_i = s_i$ for all i.
- There is a randomized 2-competitive ratio for the knapsack problem with proportional weights.
- There is no randomized online algorithm with a constant ratio for the (general values) knapsack problem.
- There is no deterministic online algorithm even if the algorithm can remove previously accepted items (but still remain within the knapsack size constraint).
- There is a 2-approximation online algorithm that is randomized and allows removing of items.

The partial enumeration greedy PTAS for knapsack

The $PGreedy_k$ Algorithm

Sort $\mathcal{I}$ so that $\frac{v_1}{s_1} \geq \frac{v_2}{s_2} \ldots \geq \frac{v_n}{s_n}$
For every feasible subset $H \subseteq \mathcal{I}$ with $|H| \leq k$
    Let $R = \mathcal{I} - H$ and let OPT_H be the optimal solution for H
    Consider items in R (in the order of profit densities)
    and greedily add items to OPT_H not exceeding knapsack capacity C.
        % It is sufficient for bounding the approximation ratio to stop
            as soon as an item is too large to fit
End For
Output: the OPT_H having maximum profit.

Sahni's PTAS result

Theorem (Sahni 1975): $V(OPT) \leq (1 + \frac{1}{k})V(PGreedy_k)$.

- This algorithm takes time kn^k and setting $k = \frac{1}{\epsilon}$ yields a $(1 + \epsilon)$ approximation running in time $\frac{1}{\epsilon}n^{\frac{1}{\epsilon}}$.
- An *FPTAS* is an algorithm achieving a $(1 + \epsilon)$ approximation with running time $poly(n, \frac{1}{\epsilon})$. There is an FPTAS for the knapsack problem (using dynamic programming and scaling the input values) so that the PTAS algorithm for knapsack was quickly subsumed. But still the partial enumeration technique is a general approach that is often useful in trying to obtain a PTAS (e.g. as mentioned for makespan).
- This technique (for $k = 3$) was also used by Sviridenko to achieve an $\frac{e}{e-1} \approx 1.58$ approximation for monotone submodular maximization subject to a knapsack constraint. It is NP-hard to do better than a $\frac{e}{e-1}$ approximation for submodular maximization subject to a cardinality constraint (i.e. when all knapsack sizes are 1).
- Usually such inapproximations are more precisely stated as "NP-hard to achieve $\frac{e}{e-1} + \epsilon$ for any $\epsilon > 0$".

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 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

End of Wednesday, September 28 class

We ended the class with a brief introdution to the priority model for myopic algorithms.
I am keeping the remaining slides in case anyone ants to see the end of the discussion on priority algorithms.
Next week Calum will be discussing online bipartite matching. ‘

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

Inapproximations with respect to the priority model

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.

1. For the weighted interval selection (a *packing problem*) with arbitrary weighted values (resp. for proportional weights $v_j = |f_j - s_j|$), no priority algorithm can achieve a constant approximation (respectively, better than a 3-approximation).
2. For the knapsack problem, no priority algorithm can achieve a constant approximation. We note that the maximum of two greedy algorithms (sort by value, sort by value/size) is a 2-approximation.
3. For the set cover problem, the natural greedy algorithm is essentially the best priority algorithm.
4. 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.

More on provable limitations of the priority model

The above mentioned inapproximations are with respect to deterministic priority algorithms. For an adaptive algorithm, the game between an algorithm and an adversary can conceptually be naturally viewed an alternating sequence of actions;

- 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 an adaptive algorithm, the game between an algorithm and an adversary can conceptually be naturally viewed an alternating sequence of actions;

- 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, there is a difference between an *oblivious adversary* that creates an initial subset $\mathcal{I}$ of items vs an *adaptive adversary* that is playing the game adaptively reacting to each decision by the algorithm. Why?

Unless stated otherwise we usually analyze randomized algorithms (for any type of algorithm) with respect to an oblivious adversary.

Extensions of the priority order model

In discussing more general online frameworks, we already implicitly suggested some extensions of the basic priority model (that is, the basic model where we have one-pass and one irrevocable decision). 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 (while always maintaining a feasible solution), then we can achieve a 4-approximation. (but provably not optimality).
- While the knapsack problem cannot be approximated to within any constant, we can achieve a 2-approximation by taking the maximum of 2 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. The partial enumeration greedy algorithm for the makespan and knapsack problems are an example of this type of extension.

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 mean by "online (priority) with advice." Online with ℓ advice bits is equivalent to the max of $k = 2^\ell$ online (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 mean by "online (priority) with advice." Online with ℓ advice bits is equivalent to the max of $k = 2^\ell$ online (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 onine 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".

The perspective now is not the tradeoff betwewen the amount of advice vs performance but rather to consider advice that may not be completely reliable. In particular, the advice might come from experience of an ML algorithm that has been learning from thousands of trials.

The goal is to exploit such advice to obtain an algorithm that iinformally is "robust" (in the sense of the algorithm not performing too badly if the advice is wrong and is "consistent" in the sense that the algorithm's performance is much better than what can be done without advice if the advice is accurate or near accurate. There are different ways to formulate tnd quantify hese requirements and (not surprisingly) there can be a tradeoff between robustness and consistency.

- 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).

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.

The first natural greedy algorithm for set packing

Greedy-by-weight (*Greedy*$_{wt}$

Sort the sets so that $w_1 \geq w_2 \ldots \geq w_n$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If S_I does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For

- In the unweighted case (i.e. $\forall i, w_i = 1$), this is an online algorithm.
- In the weighted (and hence also unweighted) case, greedy-by-weight provides an s-approximation for the s-set packing problem.
- The approximation bound can be shown by a charging argument where the weight of every set in an optimal solution is charged to the first set in the greedy solution with which it intersects.

The second natural greedy algorithm for set packing

Greedy-by-weight-per-size

Sort the sets so that $w_1/|S_1| \geq w_2/|S_2| \ldots \geq w_n/|S_n|$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If S_I does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For

- In the weighted case, greedy-by-weight provides an s-approximation for the s-set packing problem.
- For both greedy algorithms, the approximation ratio is tight; that is, there are examples where this is essentially the approximation. In particular, these algorithms only provide an m-approximation where $m = |U|$.
- We usually assume $n >> m$ and note that by just selecting the set of largest weight, we obtain an n-approximation. So the goal is to do better than $\min\{m, n\}$.

Improving the approximation for set packing

- In the unweighted case, greedy-by-weight-per-size can be restated as sorting so that $|S_1| \leq |S_2| \ldots \leq |S_n|$ and it can be shown to provide an $\sqrt{m}$-approximation for set packing.
- On the other hand, greedy-by-weight-per-size does not improve the m-approximation for weighted set packing.

Greedy-by-weight-per-squareroot-size

Sort the sets so that $w_1/\sqrt{|S_1|} \geq w_2/\sqrt{|S_2|} \ldots \geq w_n/\sqrt{|S_n|}$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If S_I does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For

Theorem: Greedy-by-weight-per-squareroot-size provides a $2\sqrt{m}$-approximation for the set packing problem. And as noted earlier, this is asymptotically the best possible approximation assuming $NP \neq ZPP$.

Another way to obtain an $O(\sqrt{m})$ approximation

There is another way to obtain the same aysmptototic improvement for the weighted set packing problem. Namely, we can use the idea of partial enumeration greedy; that is somehow combining some kind of brute force (or naive) approach with a greedy algorithm.

Partial Enumeration with Greedy-by-weight ($PGreedy_k$)

Let Max_k be the best solution possible when restricting solutions to those containing at most k sets. Let G be the solution obtained by $Greedy_{wt}$ applied to sets of cardinality at most $\sqrt{m/k}$. Set $PGreedy_k$ to be the best of Max_k and G.

- Theorem: $PGreedy_k$ achieves a $2\sqrt{m/k}$-approximation for the weighted set packing problem (on a universe of size m)
- In particular, for $k = 1$, we obtain a $2\sqrt{m}$ approximation and this can be improved by an arbitrary constant factor $\sqrt{k}$ at the cost of the brute force search for the best solution of cardinality k; that is, at the cost of say n^k.