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

Allan Borodin

September 19, 2023

Week 2

Todays agenda
We will discuss the following topics:

- Some more comments on Makespan problems
- The proportional and general knapsack problem
- Online algorithms with revocable items
- Partial enumeration greedy for the general knapsack problem.
- The priority model as a model for greedy (and "potential-based greedy") algorithms.

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 can provide somewhat better competitive ratios. We will see many examples where randomization improves upon the competitive ratio compared to deterministic algorithms.
- 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 an FPTAS.
- There is a randomized 2-competitive ratio for the knapsack problem with proportional weights.
- For the general knapsack problem:
  (1) There is no randomized online algorithm with a constant ratio.
  (2) There is no deterministic online algorithm even if the algorithm can remove previously accepted items (but always remaining within the knapsack size constraint).
  (3) There is a 2-approximation online algorithm that is randomized and allows removing of items.

The simple $\frac{1}{4}$ competitive algorithm for proportional knapsack

Algorithm 12 Simple randomized algorithm for Proportional Knapsack

procedure *SimpleRandom*

Let $b \in \{0, 1\}$ be a uniformly random bit ▷ C is the knapsack weight c

if $b = 0$ **then**

Pack items $w_1, \ldots, w_n$ greedily, that is if w_i still fits in the remaining weight k capacity, pack it; otherwise, ignore it.

else

Pack the first item of weight $\geq C/2$ if there is such an item. Ignore the rest of the i We can also pack the remaining items greedily (i.e. pack an item if it fits) but this is not to establish the competitive bound.

Figure: A barely random memoryless online algorithm for proportional knapsack

Using memory and randomization to achieve an optimal competitive algorithm for proportional knapsack

Algorithm 13 Improved randomized algorithm for Proportional Knapsack

```
procedure TwoBin
    Let b ∈ {0, 1} be a uniformly random bit    ▷ C is the knapsack weight capacity. The random
bit will determine the bin into which irrevocable decisions are being made for the items.
    C_1 ← 0; C_2 ← 0                            ▷ C_j will be the current load on bin j.
    while i ≤ n do
        if C_1 + w_i ≤ C then
            Place item i in bin 1
        else if C_2 + w_i ≤ C then
            Place item i in bin 2
        else
            Ignore the item
    if b = 0 then
        Output items in bin 1
    else
        Output items in bin 2
```

Figure: Using randomization and memory to achieve the optimal competitive online algorithm for proportional knapsack. Böckenhauer [TCS 2014] and Han et al [TCS 2015]

Revoking previously accepted items: A $\frac{1}{2}$ randomized competitive algorithm for the general knapsack problem

Algorithm 63 Randomized algorithm for the general knapsack problem

procedure MAX-PLUS-GREEDY

Let $b \in \{0, 1\}$ be a uniformly random bit ▷ W is the knapsack weight capacity. The random bit will determine will determine the bin into which irrevocable decisions are being made for the items.

$C_1 \leftarrow 0; C_2 \leftarrow 0$ ▷ C_j will be the current load on bin j. Bin 1 will maintain the current maximum value item and bin 2 will maintain those items with the best density ratio (i.e., value/size) that fits in bin 2.

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

Place item i in bin 1

Remove all but the largest value item in bin 1

Place item i in bin 2

while $\sum_k s_k > W$ **do**

Remove item $\ell = \arg\min_k [\frac{v_k}{s_k} : \text{item } k \text{ is in bin } 2]$ from bin 2.

if $b = 0$ **then**

Output items in bin 1

else Output items in bin 2

Figure: Using randomization, memory and revoking to achieve the optimal competitive algorithm for the general knapsack problem. Han et al [TCS 2015].

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 $|H| \leq k$
% This includes $H = \varnothing$
    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 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.

End of slides in Week 2

We basically ended at this point but I will add a few more slides that completes the discusion of priority algorithms.

Next week I will continue by starting where we left off this week.

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