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

Allan Borodin

September 12, 2023

Week 1

Course Organization:

1. **Sources**: No one text; lots of sources including specialized graduate textbooks, our current or past posted lecture notes (beware typos), lecture notes from other Universities, and papers. This is a very active field. Foundational course but we will discuss some recent work and research problems.
2. **Lectures and Tutorials**: One two hour lecture per week with tutorials as needed and requested. The TA is Victor Rong.
3. **Grading**: Will depend on how many students are taking this course for credit. I think we will have three assignments (tentative dates are October 10, November 7 and December 5) with an occasional opportunity for some research questions.
4. **Office hours**: TBA but we welcome questions. So feel free to drop by and/or email to schedule a time. Borodin contact: SF 2303B; bor@cs.toronto.edu. The course web page is www.cs.toronto.edu/~bor/2420f23

A little more organization

I will be using Quercus for announcements. You should be seeing those announcements if you are registered for the course. If you only intend to audit, that is fine with me but still you should audit officially so that you will see the announcements.

I will mainly be posting papers on the web page. However, I did use Quercus to post a draft of the text I am co-authoring with Denis Pankratov. I did so as I do not want this draft being distributed. Of course, I welcome any comments on this text and indeed you might find the text and particular chapters to be a starting point for research in the topic "Online and Other Myopic Algorithms". The file for the text is titled "main-no-comments.pdf".

We will be using Markus for submitting and grading assignments. Login to Markus at https://markus.teach.cs.toronto.edu/2023-09 and select 2420. I believe you can login using your utorid.

Please let me know if you cannot access Quercus or Markus.

What is an appropriate background?

- In short, a course like our undergraduate CSC 373 is essentially the prerequisite.
- Any of the popular undergraduate texts. For example, Kleinberg and Tardos; Cormen, Leiserson, Rivest and Stein; DasGupta, Papadimitriou and Vazirani.
- It certainly helps to have a good math background and in particular understand basic probability concepts, and some graph theory.

BUT any CS/ECE/Math graduate student (or mathematically oriented undergrad) should find the course accessible and useful.

Comments and disclaimers on the course perspective

- This is a graduate level "foundational course". Our focus will be on combinatorial problems using mainly discrete combinatorial algorithmic paradigms. And even more specifically, there is an emphasis on "conceptually simple algorithms".
- Perhaps most graduate algorithms courses are biased towards some research perspective.
- Given that CS might be considered (to some extent) The Science and Engineering of Algorithms, one cannot expect any comprehensive introduction to algorithm design and analysis. Even within theoretical CS, there are many focused courses and texts for particular subfields.
- The word theory in the course title reflects the desire to make some generally informal concepts a little more precise.

Why foucs on conceptually simple algorithms

Here is what I said last Friday in the theory seminar:

In practice, conceptually simple algorithms are almost always computationally efficient. Moreover, it is often the case that simple algorithms are often competitive (in performance) with the best algorithms we know for natural problems and input instances arising in common applications. And time to derive a solution can sometimes be more important than getting a better result.

However, in spite of the simplicity of some algorithms, their analysis is often anything but simple.

Moreover, for applications in mechanism design (e.g., auctions) and social choice theory (e.g., fair allocation, consensus (voting), group formation) where algorithms interact with individuals that have their own self interests and preferences, it is almost necessary to have conceptual simplicity in order for people to engage. In a sense conceptual simplicity is almost a fairness condition.

Some possible topics

Some possible topics include:

- online and greedy algorithms and their extensions; for example, the ability to revoke previous decisions, algorithms with advice.
- stochastic models including the random order model
- LP rounding; contention resolution schemes
- primal dual algorithms
- streaming algorithms
- sublinear time algorithms
- parameterized complexity, fine grained complexity
- multiplicative weights algorithm
- algorithmic mechanism design

Reviewing some basic algorithmic paradigms

We begin with some "conceptually simple" search/optimization algorithms.

The conceptually simplest "combinatorial" algorithms

Given an optimization problem, it seems to me that the conceptually simplest approaches are:

- brute force search
- divide and conquer
- greedy
- local search
- dynamic programming

Comment

- We usually dismiss brute force as it really isn't much of an algorithm approach but might work for small enough problems.
- Moreover, sometimes we can combine some aspect of brute force search with another approach as we will soon see.

Greedy algorithms in CSC373

Some of the greedy algorithms we study in different offerings of CSC 373

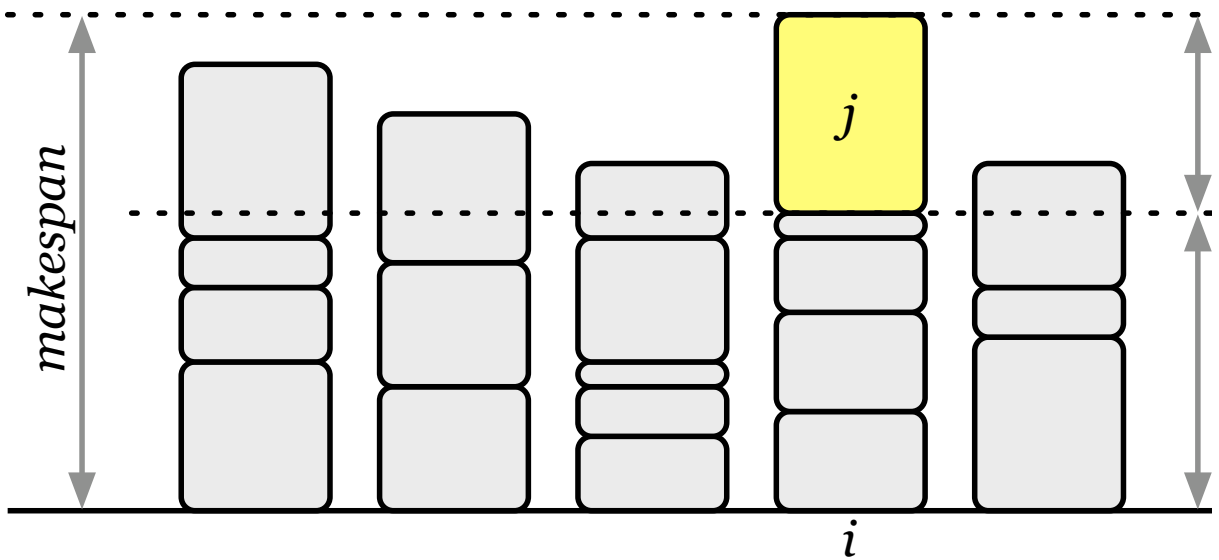
- The optimal algorithm for the fractional knapsack problem and the approximate algorithm for the proportional profit knapsack problem.
- The optimal unweighted interval scheduling algorithm and 3-approximation algorithm for proportional profit interval scheduling.
- The 2-approximate algorithm for the unweighted job interval scheduling problem and similar approximation for unweighted throughput maximization.
- Kruskal and Prim optimal algorithms for minimum spanning tree.
- Huffman's algorithm for optimal prefix codes.
- Graham's online and LPT approximation algorithms for makespan minimization on identical machines.
- The 2-approximation for unweighted vertex cover via maximal matching.
- The "natural greedy" $\ln(m)$ approximation algorithm for set cover.

Greedy and online algorithms: Graham's *online* and LPT makespan algorithms

- We start with two greedy algorithms dating back to 1966 and 1969.
- These are good starting points (preceding NP-completeness) since Graham conjectured that these are hard (requiring exponential time) problems to compute optimally but for which there were worst case approximation ratios (although he didn't use that terminology).
- This might then be called the start of worst case approximation algorithms. One could also even consider this to be the start of online algorithms and competitive analysis (although one usually refers to a 1985 paper by Sleator and Tarjan as the seminal paper in this regard).
- **NOTE:** Giving credit for any idea or result is problematic. The secretary problem and its approximation ratio precede Graham. Graham's work and the Sleator and Tarjan paper are surely seminal.
- There are some general concepts to be observed and even after 55 years still many open questions concerning the many variants of makespan problems.

The makespan problem for identical machines

- The input consists of n jobs $\mathcal{J} = J_1 \ldots, J_n$ that are to be scheduled on m identical machines.
- Each job J_k is described by a processing time (or load) p_k.
- The goal is to minimize the latest finishing time (maximum load) over all machines.
- That is, the goal is a mapping $\sigma : \{1, \ldots, n\} \to \{1, \ldots, m\}$ that minimizes $\max_k \left(\sum_{\ell:\sigma(\ell)=k} p_\ell \right)$.



[picture taken from Jeff Erickson's lecture notes]

Aside: The Many Variants of Online Algorithms

As I indicated, Graham's algorithm could be viewed as the first example of what has become known as *competitive analysis* (as named in a paper by Manasse, McGeoch and Sleator) following the paper by Sleator and Tarjan which explicitly advocated for this type of analysis. Another early (pre Sleator and Tarjan) example of such analysis was Yao's analysis of online bin packing algorithms.

In competitive analysis we compare the performance of an online algorithm against that of an optimal solution. The meaning of *online algorithm* here is that input items arrive sequentially and the algorithm must make an irrevocable decision concerning each item. (For makespan, an item is a job and the decision is to choose a machine on which the item is scheduled.)

But what determines the order of input item arrivals?

The Many Variants of Online Algorithms continued

- In the "standard" meaning of online algorithms (for CS theory), we think of an adversary as creating a nemesis input set and the ordering of the input items in that set. So this is traditional worst case analysis as in approximation algorithms applied to online algorithms. If not otherwise stated, we will assume this as the meaning of an online algorithm and if we need to be more precise we can say *online adversarial input model*.
- We will also sometimes consider an *online stochastic model* where an adversary defines an input distribution and then input items are sequentially generated. There can be more general stochastic models (e.g., a Markov process) but the i.d. and especially the i.i.d model is common in analysis. Stochastic analysis is often seen in OR.
- In the i.i.d model, we can assume that the distribution is *known* by the algorithm or *unknown*.
- In the *random order arrival (ROA) model*, an adversary creates a size n nemesis input set and then the items from that set are given in a uniform random order (i.e. uniform over the $n!$ permutations).

Second aside: more general online frameworks

In the standard online model (and the variants we just mentioned), we are considering a one pass algorithm that makes one irrevocable decision for each input item.

There are many extensions of this one pass paradigm. For example:

- An algorithm is allowed some limited ability to revoke decisions.
- There may be some forms of lookahead (e.g. buffering of inputs).
- The algorithm may maintain a "small' number of solutions and then (say) take the best of the final solutions.
- The algorithm may do several passes over the input items.
- The algorithm may be given (in advance) some *advice bits* based on the entire input. Recently, lots of interest in online algorithms with predictions (e.g., often called ML advice).

Throughout our discussion of algorithms, we can consider deterministic or randomized algorithms. In the online models, the randomization is in terms of the decisions being made. (Of course, the ROA model can also be viewed an example of a randomized online algorithm where the ordering of the inputs is randomized but this is not the standard meaning.)

A third aside: other measures of performance

The above variants address the issues of alternative input models, and relaxed versions of the online paradigm.

Competitive analysis is really just asymptotic approximation ratio analysis applied to online algorithms. Given the number of papers devoted to online competitive analysis, it is the standard measure of performance.

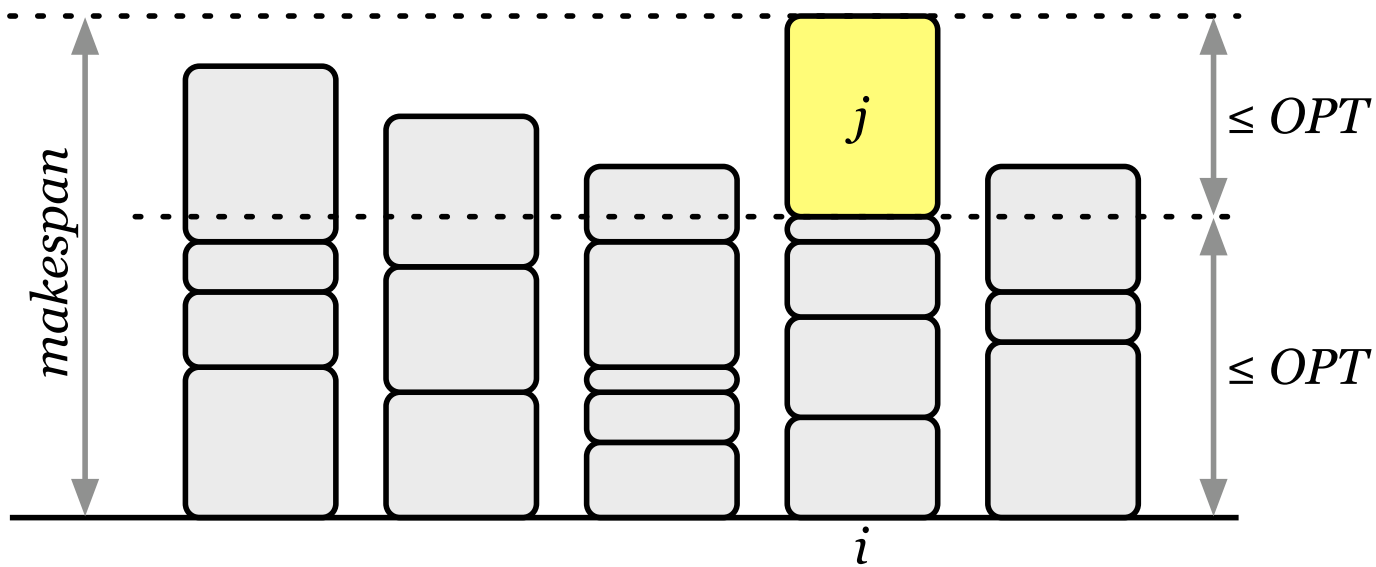
However, it has long been recognized that as a measure of performance, worst case competitive analysis is often at odds with what seems to be observable in practice. Therefore, many alternative measures have been proposed. An overview of a more systematic study of alternative measures (as well as relaxed versions of the online paradigm) for online algorithms is provided in Kim Larsen's lecture slides that I have placed on the course web site.

In the makespan problem, the objective is a *min-max* objective. One also studies the *max-min* objective which can be viewed as a **fairness measure**.

Returning to Graham's online greedy algorithm

Consider input jobs in any order (e.g. as they arrive in an *online* setting) and schedule each job J_j on any machine having the least load thus far.

- We will see that the **approximation ratio** for this algorithm is $2 - \frac{1}{m}$; that is, for any set of jobs $\mathcal{J}$, $C_{Greedy}(\mathcal{J}) \leq (2 - \frac{1}{m}) C_{OPT}(\mathcal{J})$.
  - C_A denotes the cost (or makespan) of a schedule A.
  - OPT stands for any optimum schedule.
- **Basic proof idea:** $OPT \geq (\sum_j p_j)/m$; $OPT \geq max_j p_j$
  What is C_{Greedy} in terms of these requirements for any schedule?



[picture taken from Jeff Erickson's lecture notes]

Graham's online greedy algorithm

Consider input jobs in any order (e.g. as they arrive in an online setting) and schedule each job J_j on any machine having the least load thus far.

- In the online "competitive analysis" literature the ratio $\frac{C_A}{C_{OPT}}$ is called the **competitive ratio** and it allows for this ratio to just hold in the limit as C_{OPT} increases. This is the analogy of *asymptotic approximation ratios*.

NOTE: Often, we will not provide proofs in the lecture notes but rather will do or sketch proofs in class (or leave a proof as an exercise).

- The approximation ratio for the online greedy is "tight" in that there is a nemisis sequence of jobs forcing this ratio; namely, consider a sequence of $m \cdot (m-1)$ unit jobs followed by a job of size m.
- This bad input sequence suggests a better algorithm if we can order the inputs; namely the greedy LPT (offline or sometimes ambiguously called semi-online) algorithm.

Graham's LPT algorithm

Sort the jobs so that $p_1 \geq p_2 \ldots \geq p_n$ and then greedily schedule jobs on the least loaded machine.

- The (tight) approximation ratio of LPT is $\left(\frac{4}{3} - \frac{1}{3m}\right)$.
- It is believed that this is the **best** "greedy" algorithm but how would one prove such a result? This of course raises the question as to what is a greedy algorithm.
- We will present the priority model for greedy (and greedy-like) algorithms. I claim that all the algorithms mentioned on slide 6 can be formulated within the priority model.
- Assuming we maintain a priority queue for the least loaded machine,
  - the online greedy algorithm would have time complexity $O(n \log m)$ which is $(n \log n)$ since we can assume $n \geq m$. In fact, one can assume $n >> m$ and hence the online greedy is faster than the following offline greedy LPT algorithm.
  - the LPT algorithm would have time complexity $O(n \log n)$.

We already encounter some open questions relating to the "classical" makespan problem.

We can ask the following questions about the makespan problem and realize that we can ask the same questions for other problems.

- What is the optimal online ratio?
- What is the optimal "greedy algorithm"? This questions then requires us to provide a defintion for what is a greedy algorithm.
- Is there a meaningful sense in which any of the lower bounds are "asymptotic" (with respect to n) when fixing m. It seems unsatisfying if we scale the nemisis sequence. Do the lower bounds assume we do or don't know n? What is the best c such that some algorithm obtains $ALG \leq c \cdot OPT + o(OPT)$ so that as OPT grows the competitive ratio is limiting to 1.

General themes

We will be seeing a number of different problems and different algorithms. I want to call attention to some general themes that I hope we can keep in mind. In particular, I am interested in

- Clssses of algorithms (as it is taught or introduced in most undergrad texts)
- Basic problems solved in a variety of ways with different properties; tradeoffs
- Relation between problems; extensions, reductions
- Methods of analysis

We won't worry about implementation but rather algorithmic approaches

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. We will begin this week mentioning just one way to obtain a $(1 + \epsilon)$ approximation but at a high cost.

Partial Enumeration Greedy

- 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 $O(m^{m/\epsilon} + n \log n)$.

End of Tuesday, Sept 12 class

We mostly motivated the course.

In doing so, we discussed the makespan problem (for identical machines). The proof of the competitive ratio $(2 - \frac{1}{m})$ for the natural online greedy algorithm is in section 2.2 of our text.

The proof for the approximation ratio $(\frac{4}{3} - \frac{1}{3m})$ of LPT is in section 18.4.5 of the text.

In general I will only sketch proofs (or omit proofs). Our text will usually have references in the Historical sections for each chapter. But always best to try to think about how a proof may go before trying to find a source.

We also briefly mentioned the paging problem, the k-server problem, bipartite matching, and dynamic programming, but these brief discussions are not in this weeks slides.