CSC2421: Online and Other Myopic Algorithms
Fall 2025

Allan Borodin

September 10, 2025

Week 2

We ended last week including the last few slides to complete the discussion of the chapters.

We will just briefly go over the listing of Chapters 12-19. (I am not certain if I discussed Chapters 12 and 13.)

Then I will present a little more motivation for this course.

And then (at long last) we will proceed with some technical discussion of the makespan problem.

Why Study Online and Other Myopic Algorithms?

In many applications, there is an inherent online requirement. For example, *paging* (also called *caching*) algorithms have to decide what page to evict when the cache (or some level of the memory hierarchy) is full.

Some auctions need to be run in an online fashion. One very important example is online advertising (i.e. selling ads deriving from online search to advertisers).

In other applications it may not be a requirement but rather a desireable property. Moreover, online and other myopic algorithms tend to be conceptually simple and very efficient.

Even when online algorithms do not provide solutions that are "competitive" in performance with more complex algorithms, they can serve as a benchmark or initial solution when an initial solution is needed quickly.

The difference between online/myopic algorithm analsyis and more standard algorithm analysis as in approximation algorithms and complexity theory.

The seminal work of Cook-Karp-Levin on *NP*-completeness strongly suggests that there are many related problems that cannot be solved efficiently (i.e., in polynomial time). But this is still a conjecture.

Although "we" strongly believe $P \neq NP$, we do not have explicit problems f in *NP* for which we can *prove* that f is not computable in linear time for a sufficiently general model of computation.

In contrast, in the analysis of online and other myopic algorithms, we restrict the class of algorithms and prove negative results without any complexity (or even computability) assumptions. We use what are called *information theoretic* arguments to establish negative results. That is, because the algorithm is working with incomplete information, there are limitations to what such an algorithm can do. **Aside: This is one reason why we have debated whether or not to include Streaming and Dynamic Algorithms.**

Comments and disclaimers on the course perspective

- I consider this course to be a "foundational graduate course" even though online and other myopic algorithms are a small part of the field of *algorithm design and analysis*.
- Although the topic seems focused, the analysis of these algorithms often introduces analysis methods used more generally in algorithmic analysis.

  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 in any course.
- I already mentioned that online algorithms is currently an active part of TCS. The reason is mainly that the number of online applications keeps expanding and new algorithmic ideas and results continue to be developed. Furthermore, there is increasing interest in the many variants of online algorithms and analysis that address current applications.

Rest of todays agenda

I have posted a somewhat current draft of the text on the web page.

Pleaase do not distribute.

The text is constantly changing but the first 7 chapters are pretty stable. I will update the text draft in a few weeks.

So... before we really begin, are there any questions or comments?

Rest of todays agenda

I have posted a somewhat current draft of the text on the web page.

Pleaase do not distribute.

The text is constantly changing but the first 7 chapters are pretty stable. I will update the text draft in a few weeks.

So... before we really begin, are there any questions or comments?

We start with a little history and the makespan problem.

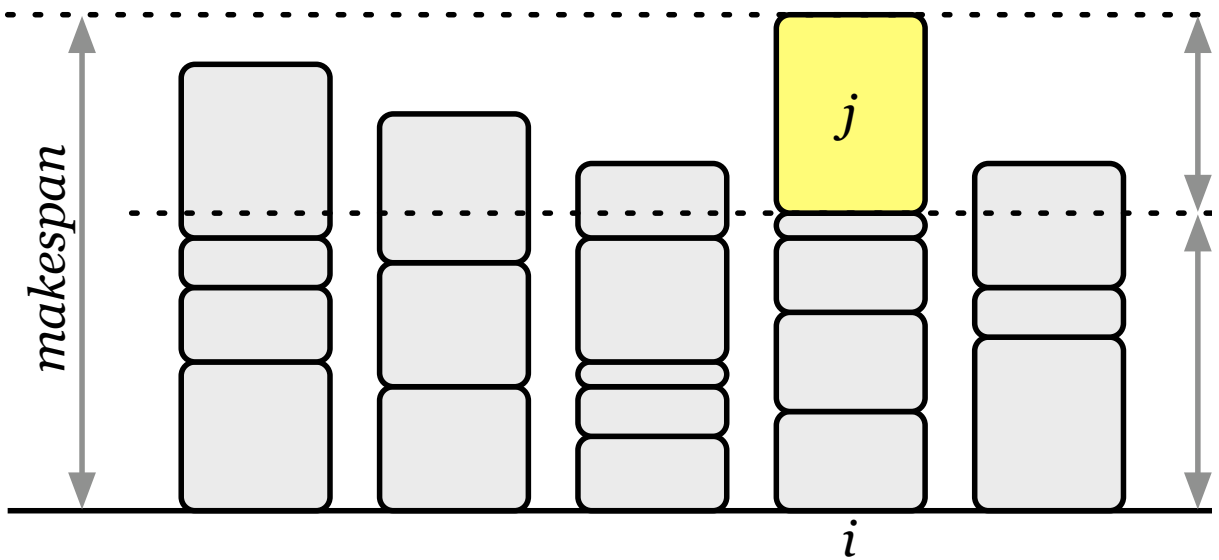
I will more or less start by following the chapter sequence but will interject some more recent developments.

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

- Let's start with these two greedy algorithms (one online and one "semi-online") that date back to Graham's 1966 and 1969 papers.
- These are good starting points since (preceding NP-completeness) 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). As pointed out in Chapter 1, there are other works that precede even the Graham paper.
- There are some general concepts to be observed in Graham's work and (even after more than 50 years) still 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]

Redux: 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 i.i.d models are common in analysis. *Stochastic analysis* is well studied in OR.
- In the i.d. and i.i.d models, we can assume that the distributions are *known* by the algorithm or *unknown*.
- In the *random order model* (ROM), 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)

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 previous 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.

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 ROM model is an example of where the ordering of the inputs is randomized.)

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.

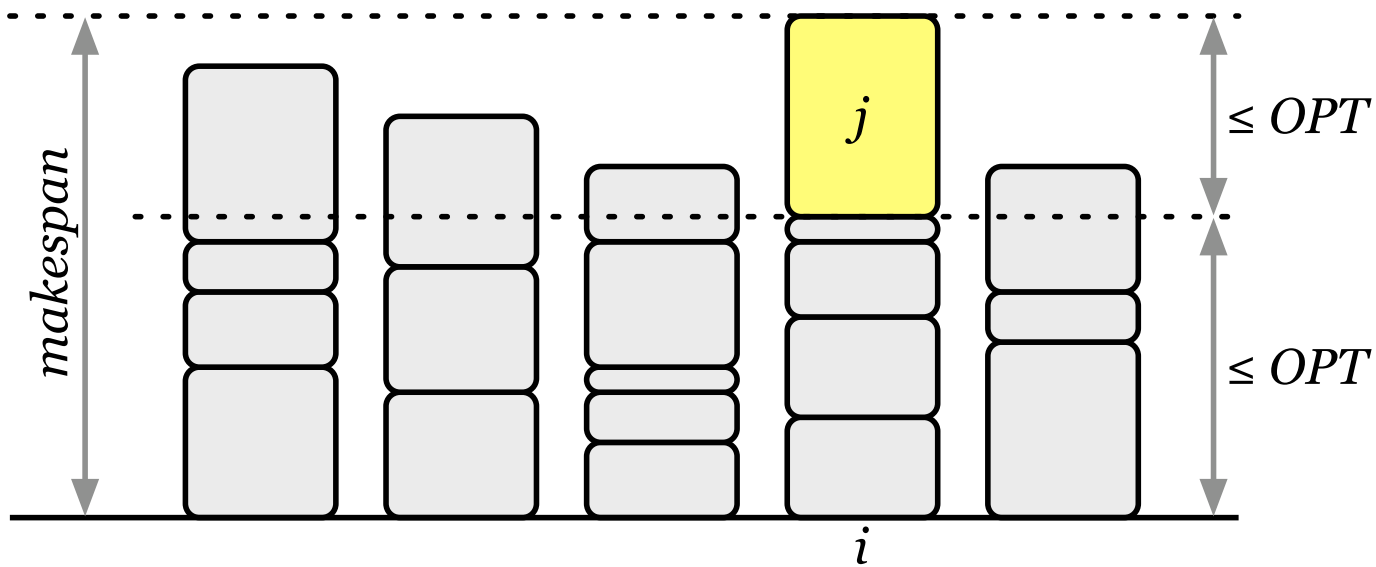
However, it has long been recognized that as a measure of performance, competitive analysis is often at odds with what seems to be observable in practice. Therefore, many alternative measures have been proposed. An overview of a more systematic study of alternative measures (as well as relaxed versions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

See, for example, the discussion of the *accommodating function* measure (for the dual bin packing problem) and the *relative worst order* meaure for the bin packing coloring problem.

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 sequence of jobs forcing this ratio.
- The negative result (i.e. there is "bad" input sequence for the algorithm) is not an asymptotic result. Is there a "meaningful" asymptotic result?
- This bad input sequence suggests a better algorithm, namely the LPT (offline or sometimes called semi-online) greedy 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 the next slide 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 $n \geq m$.
  - the LPT algorithm would also have time complexity $O(n \log n)$.

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 unit profit 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.
- The 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.

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.
- Makespan has been actively studied with respect to three other machine models.
- Randomization, ML-advice, and recourse can provide better competitive ratios for makespan applications (and other applications).

A Not So New Result for the Makespan problem on Identical machines

In 2002, Susanne Albers gave a randomized algorithm that uses only one initial unbiased random bit to decide between two deterministic algorithms. This results in a randomized algorithm that (in expectation wrt to this random bit) is 1.916 competitive beating the best known deterministic algorithm. This is still currently the best randomized algorithm for the makespan problem on identical machines. Albers also gave a $\frac{1}{(1-\frac{1}{m})^m}$ lower bound for all m. This ratio limits to $\frac{e}{e-1} \approx 1.5819$ as $m \to \infty$.

Why I am interested in a result that makes such a small improvement and probably is not the final word on randomized makespan algorithms?

A Not So New Result for the Makespan problem on Identical machines

In 2002, Susanne Albers gave a randomized algorithm that uses only one initial unbiased random bit to decide between two deterministic algorithms. This results in a randomized algorithm that (in expectation wrt to this random bit) is 1.916 competitive beating the best known deterministic algorithm. This is still currently the best randomized algorithm for the makespan problem on identical machines. Albers also gave a $\frac{1}{(1-\frac{1}{m})^m}$ lower bound for all m. This ratio limits to $\frac{e}{e-1} \approx 1.5819$ as $m \to \infty$.

Why I am interested in a result that makes such a small improvement and probably is not the final word on randomized makespan algorithms?

I am generally interested in results that lead to more general observations. For example, I am interested in when a randomized algorithm can be simulated by a deterministic algorithm that extracts randomness from the random input string in the random order model. Can we obtain better bounds using more than 1 bit of randimness?

Two Relatively New Results for the Makespan Problem on Identical Machines

Randomness extraction from a ROM input sequence is an issue that Chris Karavasilis and I began to recently study. David Zhang is looking at the Albers paper with regard to this kind of online random bit extraction.

The issue here is how soon can we extract the random bit and use that to simulate the "barely random" algorithm.

In a 2021 paper, Albers and Janke show that for the makespan problem on identical machines there is a deterministic algorithm in the ROM model that in expectation (over the randomness in the input sequence) is 1.8478 competitive provably beating the determnistic lower bound that holds for adversarial input sequences and better than the best known randomized ratio. Osborn and Torng [2008] show that Graham's online greedy algorithm is not better than 2 in ROM (asymptotically as $m \to \infty$).
Note: We will see examples (e.g., the Secretary problem) where ROM can even more significanty beat randomized algorithms acting on adversarial sequences.

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 (and still not well undderstood) problem.
- 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.
- NOTE: Any statement I make 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 an algorithm $\mathcal{A}$ and a given input instance for the *makespan with precedence* problem. Which of the following should never lead to an increase in the makepan objective?

- 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 an algorithm $\mathcal{A}$ and a given input instance for the *makespan with precedence* problem. Which of the following should never lead to an increase in the makepan objective?

- 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.