CSC2421 Topics in Algorithms: Online and Other Myopic Algorithms Fall 2019

Allan Borodin

September 11, 2019

Week 1

Course Organization:

1. **Sources**: The text is a book currently being written by Allan Borodin and Denis Pankratov. Lots of additional sources including
   - various specialized graduate textbooks
   - my posted and sktechy lecture notes (beware typos)
   - lecture notes from other Universities and short courses, and
   - research papers.
2. **Lectures and Tutorials**: One two-three hour lecture per week with tutorials as needed and requested; not sure if we will have a TA. Wednesdays are a little problematic for me but we will wait until October to see if we need to or can change the day when this course meets.
3. **Office hours**: TBA but I always welcome questions (in class or otherwise). So feel free to drop by and/or email to schedule a time. My contact information: SF 2303B; bor@cs.toronto.edu. The course web page is www.cs.toronto.edu/~bor/2421f19

Course Description

In a seminal 1985 paper, Sleator and Tarjan argued for a worst case analysis of online algorithms such as paging and list accessing. This became known as competitive analysis. Not surprisingly, such worst analysis was already present in earlier works such as

- Graham [1966,1969] makespan
- Garey, Graham and Ullman [1972] bin packing
- Yao bin packing

Since these earlier works, there has been a continuing and growing interest in online algorithms, in terms of applications (eg online advertising and other auctions, graph colouring and matching, maximum satisfiability, etc.), alternative online models (eg small space streaming, sequential and parallel streams), extensions to the basic online model (revocable decisions, greedy-like algorithms) and alternatives to the competitive analysis framework (eg, a return to stochastic input models).

Course Description continued

This course relates to major themes of interest within theoretical computer science that emphasize ”conceptually simple algorithms”, “beyond worst case analysis”, and “decision making under uncertainty”.

A preliminary table of contents and some preliminary chapters for our textbook will be available. The text is constantly being modified but I will try to indicate when there have been any significant changes.

I have password protected the text chapters since the text thus far is in a very preliminary stage and is bound to have technical mistakes and improper or incomplete references. hence I do not want the text to be widely distributed.

More on course focus and a disclaimer

- The Design and Analysis of Algorithms is a very active field. As part of this activity, I think it is fair to say that there is a growing interest in conceptually simple algorithms (e.g. a new conference SOSA aligned with SODA). Furthermore, one might even claim that online algorithms and other "myopic algorithms" (e.g. greedy algorithms) is having a renaissance in interest. Our course is a foundational course in the sense that even though our focus is very specific, the topic is broad enough to have direct relation with concepts used more generally in the design and analysis of algorithms (e.g., primal dual algorithms and analysis).
- Disclaimer: We will not try to "cover" all the recent developments within this topic. Some of the results are quite technical. We will often just mention some results or just sketch the main ideas. Our goal is to try to present a spectrum of concepts and results. For the same of understanding, we will often forgo the best known result for a result that is "good enough" to establish basic concepts and results.

What is appropriate background? Grading

- 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 (see our Probability Primer), and some graph theory.

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

- **Grading**: Will depend on how many students are taking this course for credit. I am thinking that we may run half of this course as a reading course related to the new text depending on your interest in the material relating to the text. I will soon provide an initial assignment to help insure that everyone has a reasonable idea of the technical depth of the course. The reading part of the course will allow us to follow some of the current research within our topic.

Some informal definitions

Most of what I am saying today appears in chapters 1 and 2 of the text.

- For the main part, we will consider *online algorithms* in the context of *request-answer games* and optimization. Namely, input items arrive sequentially in discrete steps and an online algorithm must respond to each input item before the next item arrives. The goal is to achieve some objective. Of course, one cannot expect an online algorithm to perform as well as an offline algorithm that initially has complete knowledge of the entire input. *Competitive analysis* is a *worst case* analysis that tries to understand how well an online algorithm can do with respect to an optimal solution for every possible input sequence.

Some informal definitions continued

- There are, however, alternative meanings for an algorithm being "online". In scheduling, online usually means a *real-time algorithm* in the sense that jobs arrive in continuous time and the algorithm can respond to a job arrival at any time after it arrives but delays in responding will usually impact the desired objective.
- In searching an unknown environment, an online algorithm has to discover the environment as it is searching.
- A *myopic algorithm* generalizes the online concept in that the algorithm may have some limited knowledge of the future. *Greedy algorithms* are a primary example of a myopic algorithm. We shall study an abstraction of greedy algorithms called *priority algorithms* which in hindsight could have been called *myopic algorithms*.

Some informal definitions continued

- We will also consider *streaming algorithms* where input items are arriving online but there may or may not be a requirement to respond immediately to each new input item. The focus in streaming algorithms is the impact of limited space. Often the objective is to maintain statistics for very large streams of data while using only say $O(\log n)$ space. More recently, there has been some interest in *semi-streaming* algorithms where for exanple we consider graph optimization problems using space $\tilde{O}(n)$ space rather that $O(m)$ space where $n = |V|$ and $m = |E|$. .
- We may also discuss *dynamic algorithms* where input requests are updates (e.g., add or delete an edge) and queries (e.g. what is the current diameter of the graph). Here we are usually interested in the tradeoff between the time for updates and the time for queries.

Streaming and dynamic algorithms are clearly online algorithms, also usually studied in the worst case scenario, but with a different focus than online algorithms as studied in competitive analysis.

Why restrict ourselves to conceptually simple algorithms and in particular why restrict ourselves to online and myopic algorithms?

In some applications, it is more important to produce a "reasonably good" solution quickly rather than getting a better solution or best solution that may take much longer to create or will take much longer to execute. Additionally, in some applications (e.g., auctions), users want solutions (e.g., who gets what and at what cost) they can understand.

Some applications are necessarily online (e.g., paging, real time scheduling) and hence there is no alternative.

Even if an application is not necessarily online, an online algorithm may provide a conceptually simple solution, or a solution that can be easily modified using additional offline information to create a good solution.

Greedy and priority algorithms are an extension of online algorithms where the algorithm has some limited ability to create the sequence (but not the set) of input items but still has to make immediate decisions for each item.

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.
- 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 online algorithm for unweighted vertex cover via maximal matching.
- The "natural greedy" $\ln(m)$ approximation algorithm for set cover.

Lets start with examples from Chapters 1 and 2.

In Chapter 1, we discuss what seems like a toy problem, the *ski rental problem*. We also discuss the *paging problem*. These both fall within the request answer framework discussed in Chapter 2.

In Chapter 2, we discuss two minimization problems (makespan and bin packing) and two maximization problems (time series search and one-way trading) and provide deterministic online algorithms for these problems.

We analyze these algorithms from the perspective of their competitive ratio as defined in Chapter 2.

The ski rental problem

We start with a problem that can be considered a toy problem but does model some realistic scenarios. Furthermore, in Chapter 4 we consider some of the extensions of this problem (multislope ski rental, the Bahncard problem, and the TCP acknowledgement problem) adding additional motivation. The *ski rental* or the *leasing* problem is as follows:

A skier (i.e. an online algorithm) has to decide every day (or every time thinking about a ski day) whether to buy (at some price b) a pair of skis or to rent (at some price $r < b$ per day) for the day. The problem is that the skier doesn't know if and when the weather will change and the season will end (or he/she will just lose interest). If the skier knew the season would last long enough it would clearly pay to buy but "today" could be the last day that the skier will ever ski again in which case the skier would clearly rent for the day. What to do?

Aside: I have faced this problem recently when on sabbatical trying to decide if I should buy a bike (and hopefully resell before leaving) or rent whenever I wanted. Have you faced this problem in some form?

The ski rental problem continued

As simple as this problem is, it does raise some concepts and issues that are basic to our topic and more generally to algorithm design and complexity theory. . Here are basic concepts and issues that we will often encounter:

1. We view the weather as an *adversary* whose goal is make the skiers' decision to look bad in hindsight. This extent to which the online decisions are bad in hingsight will be captured by the *competitive ratio*. This is analgous to the concept of *regret* as used in online learning which is the subject of Chapter 18. For deterministic algorithms in the worst case setting, note that the adversary knows an optimal solution for any instance.
2. Does it help for an online algorithm to use randomization?
3. As soon as randomization is introduced, there are different concepts as to the power of an adversary.
4. In Chapter 16 we depart from the worst case setting and consider stochastic inputs. This raises questions as to what are appropriate "benchmarks" against which an online algorithm is competing.

The best online algorithm

For this "classic" version of the ski rental problem, the adversary's power is simply to determine when is the last day of skiing.

We will first show that there is a simple deterministic algorithm that can insure that *on every input instance*, the algorithm will pay no more than twice the cost of an optimum solution. Then we will show that this is the best possible competitive ratio (i.e., ratio of algorithm to optimal cost).

Note that in this problem there is a simple and efficient offline algorithm that an adversary (knowing the number of ski days which it determines) can use to compute an optimal solution OPT. What is an optimal algorithm for the adversary? But for the concept of the competitive ratio, it might be the case that an optimal solution cannot be computed efficiently or might not even know any optimal algorithm. But we do need to know something about properties of an optimal algorithm.

The ski rental algorithm

Here is the online algorithm (or one might simply say strategy in this case) that achieves competitive ratio $\rho \leq 2 - \frac{1}{b} \rightarrow_{b \rightarrow \infty} 2$.

Note: I am going to abuse notation and use ALG and OPT to denote both a solution and its cost.

Competitive ski rental algorithm ***ALG***

Rent for up to $b - 1$ days and then buy.

Proof of the competitive bound for this simple strategy

Without loss of generality (why?) let $r = 1$ and hence $b \geq 1$. Let $g =$ number of ski days. Then $OPT = \min\{g, b\}$ depending on whether $g \leq b$ or $g > b$. Note: $g > b$ includes never buying.

- If $g < b$, then ALG just rents and its cost is g which is also the optimal cost so $ALG = g \leq (2 - \frac{1}{b})g = (2 - \frac{1}{b})OPT$,
- If $g \geq b$ then ALG's cost is $b - 1 + b = (2b - 1)$ and OPT pays b so that $\frac{ALG}{OPT} = \frac{2b-1}{b} = 2 - \frac{1}{b}$.

Reflections on the ski rental algorithm and analysis

It is easy to see that this *analysis of the algorithm is tight*. But is this the best we can do? That is, is there any deterministic algorithm that can do better (in terms of the worst case ratio)?

Reflections on the ski rental algorithm and analysis

It is easy to see that this *analysis of the algorithm is tight*. But is this the best we can do? That is, is there any deterministic algorithm that can do better (in terms of the worst case ratio)?

The answer is that this is the best determininstic algorithm algorithm. Suppose that an algorithm ALG decides to buy on some day i (or decides to never buy).

- If $i \leq b-1$ then the adversary decides that day i is the last day of skiing. OPT pays i and ALG pays $i-1+b \geq i-1+(i+1) = 2i$
- If $i \geq b$, then the adversary ends skiing after the first $2b$ days. Now OPT pays b and ALG pays $i-1+b \geq b-1+b = 2b-1$ so that the ratio $\frac{ALG}{OPT}$ is at least $2 - \frac{1}{b}$

More reflections on the analysis

In principle, in competitive analysis we need not care if the online algorithm is efficient. Note that is the above analysis we didn't care how the algorithm decided to chose i. We are just trying to uderstand the limitations imposed by the myopic online requirement. Of course, in practice, we do care about efficient algorithms and although the analysis may ignore computational efficiency, online algorithms tend to be very efficient. The negative result is an example of an *information theroretic argument*.

More reflections on the analysis

In principle, in competitive analysis we need not care if the online algorithm is efficient. Note that is the above analysis we didn't care how the algorithm decided to chose i. We are just trying to uderstand the limitations imposed by the myopic online requirement. Of course, in practice, we do care about efficient algorithms and although the analysis may ignore computational efficiency, online algorithms tend to be very efficient. The negative result is an example of an *information theroretic argument*.

Indeed the competitive ski rental algorithm is exceptionally computational simple. The algorithm only needs to have memory to remember what day it is. However, we allow online algorithms to keep as much information about previous inputs as it wishes and make decisions based on all this information.

More reflections on the analysis

In principle, in competitive analysis we need not care if the online algorithm is efficient. Note that is the above analysis we didn't care how the algorithm decided to chose i. We are just trying to uderstand the limitations imposed by the myopic online requirement. Of course, in practice, we do care about efficient algorithms and although the analysis may ignore computational efficiency, online algorithms tend to be very efficient. The negative result is an example of an *information theroretic argument*.

Indeed the competitive ski rental algorithm is exceptionally computational simple. The algorithm only needs to have memory to remember what day it is. However, we allow online algorithms to keep as much information about previous inputs as it wishes and make decisions based on all this information.

In the analysis it seems like the adversary has to be observing what ALG is doing to know when to end the skiing. But since ALG is deterministic, the adversary knows in advance which day (if ever) ALG will buy.

Can randomization help?

In a randomized online algorithm, the algorithm can make decisions as a probabilistic function of all the previous information. Can randomization help?

Can randomization help?

In a randomized online algorithm, the algorithm can make decisions as a probabilistic function of all the previous information. Can randomization help?
The answer is yes but the algorithm and its analysis is a little more involved. Essentially the algorithm uses a partitcular probability density function to choose the time to buy. We also have to be a little careful in what we mean by the competitve ratio of a randomized algorithm. But for now here is the statement of the result (which also holds for the generalizations of ski rental previously mentioned):

Randomized ski rental competitive ratio

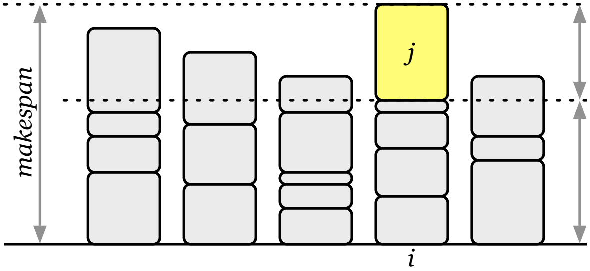
There is a randomized algorithm for the ski rental problem achieving competitive ratio $\frac{e}{e-1} \approx 1.58$ against an *oblivious adversary* (i.e., an adversary that see the algorithm but not the random bits and hence not the actual decisions of the algorithm).

Graham's *online* and LPT makespan algorithms

- Let's continue with two greedy algorithms that date back to 1966 and 1969 papers.
- These are also good starting points since (preceding NP-completeness) Graham conjectured that makspan is a hard (requiring exponential time) problem 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).
- Moreover, there are again some general concepts to be observed in this work and even after nearly 50 years, there are 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]

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.

As we already stated, 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.)
Let's review and expand upon what we already mentioned in regard to the ski rental problem.

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 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.i.d model is common in analysis. Stochastic analysis as 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 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)

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 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.)

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.

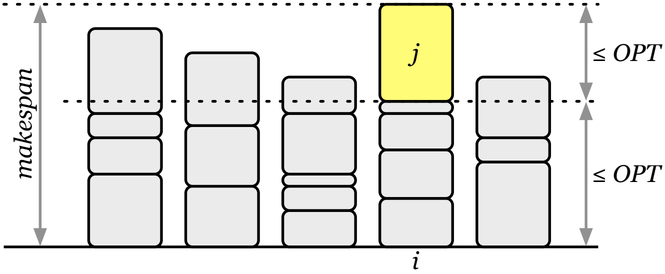
However, it has long been recognized that as a measure of performance, competitive analysis is often at odds with what seems to be observable in practice. Therefore, many alternative measures have been proposed. An overview of a more systematic study of alternative measures (as well as relaxed versions of the online paradigm and restricted input instances) 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), the *relative worst order* meaure for the bin packing coloring problem, and the page fault rate measure for paging.

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.
- This bad input sequence suggests a better algorithm, namely the LPT (offline or sometimes 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 10 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 have time complexity $O(n \log n)$.

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

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.
- Makespan has been actively studied with respect to three other machine models.
  We plan to consider these other models when we get to Chapter 4.

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 deterministic (resp. randomized) online ratio is 5.828 (resp. 4.311) due to P. Berman et al [2000] following the first constant ratio by Aspnes et al.
- Ebenlendr and Sgall [2015] establish a deterministic online inapproximation of 2.564 following the 2.438 deterministic online inapproximation of Berman et al. who also proved a 1.8372 inapproximation for any randomized online algorithm.

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 bin packing problem

Our next classic minimization problem is the one-dimensional *bin packing* problem. In this problem we have a set of items each having a size or weight x_j and these items have to all be packed into bins of a fixed size B. The objective is to minimize the number of bins. (This problem is sometimes referred to as *the cutting stock problem*). Without loss of generality, we can set $B = 1$ and then assume that $x_j \leq 1$ for all j.

Online bin packing was first studied in a 1972 STOC conference paper by Garey, Graham and Ullman, and then in a 1973 Johnson FOCS paper, and in a 1974 journal article combining Garey et al with David Johnson and Alan Demers. There are many subsequent bin packing papers dealing with online and greedy type algorithms. This work (proceeded by Graham's earlier makespan results) became the driving force for the area of approximation algiorithms following the introduction of NP-completeness. Johnson's PHD thesis was devoted to approximation algorithms.

Bin packing continued

There are some natural online algorithms for the bin packing problem which in turn have natural greedy extensions when initially ordering the x_j. Furthermore, although the problem is also NP-complete, (i.e., deciding if an instance can be packed into 2 bins is NP complete), there are offline algorithms ALG such that $ALG \leq OPT + o(OPT)$ so that the asymptotic approximation ratio is 1. As far as I know, as an offline problem there may be an algorithm that achieves $ALG \leq OPT + 1$.

There are also higher dimensional versions. For example, in dimension 2, the goal is to pack rectangles (usually axis aligned) into say 1 by 1 square bins.

Three natural online algorithms

Consider the following three online algorithm:

1. *NextFIT*: If the next item x_j does not fit into the *most recently opened* bin, then open a new bin and place the new item in that bin. See Algorithm 2 in the text for pseudocode. Note this is a myopic online algorithm but not a greedy algorithm in the following "live for today sense": for each input, we make a decision so as to optimize the objective function.
2. *FirstFit*: Find the first bin (if any exists) among *all opened bins* that has enough remaining space to accommodate the newly arriving item. If such a bin exists, place the new item there. Otherwise, open a new bin and place the new item in the new bin. See Algorithm 3 in the text for pseudocode.
3. *BestFit*: Find a bin among *all opened bins* that has *minimum* remaining space among all bins that have enough space to accommodate the current arriving item. If there are no bins that can accommodate the current arriving item, open a new bin and place the new item in the new bin. See Algorithm 4 for the pseudocode.

The competitveness of the online algorithms

1. The competitive ratio is NextFit $= 2$.
2. The competitive ratio for FirstFit and BestFit is 1.7. Recently this was shown to be a strict competitive ratio.
3. These (asymptotic) ratios are tight for these algorithms.
4. There are now better asymoptotic competitive ratio where the current best has competitiveness 1.57829 as compared with the best known lower (i.e. negative result) is 1.54037

If we are allowed to first sort the items so that $x_1 \geq x_2 \ldots \geq x_n$, then for FirstFit-Decreasing and BestFit-Decreasing, it has been shown that these algorithms satisfy $ALG \leq \frac{11}{9} OPT + 1$.
Since we have presented the results for the online bin packing algorithms in the text, we shall just sketch those results on the board. (We also presented the ski rental and Graham's makespan results in the text but wanted to use those to introduce the topic. The analysis for FirstFit and BestFIt is a little more involved than for NextFIt but still these analyses are all "combinatorial".

Formalizing Request-Answer games and the Competitive Ratio for Deterministic Minimization Problems

Having presented three example of online problems and corresponding online algorithms, we present the precise definitions in Chapter 2.

The problems and online algorithms we consider in Part I of the text (with the exception of the Line Search Problem) can be abstracted by *request-answer games* and the Online Algorithm Template. Request-answer games abstract both mininization and maximization problems. We first define the competitive ratio for minimization problems and algorithms. Following that we will consider two maximization problems and the corresponding definition for the competitive ratio.

We define the competitve ratio as follows:
$\rho(ALG) = limsup_{OPT(\mathcal{I} \rightarrow \infty} \frac{ALG(\mathcal{I})}{OPT(\mathcal{I})}$