CSC2420: Algorithm Design, Analysis and Theory Fall 2012

Allan Borodin

September 13, 2012

Lecture 1

Course Organization:

1. **Sources**: No one text; lots of sources including specialized graduate textbooks, my posted lecture notes (beware typos), lecture notes from other Universities, and papers. Very active field and we will discuss some recent work.
2. **Lectures and Tutorials**: One two hour lecture per week with tutorials as needed and requested; TA is
3. **Grading**: Will depend on how many students are taking this course for credit. Most likely, like last term there will be three assignments with an occasional opportunity for some research oriented questions.
4. **Office hours**: I will be posting regular office hours for my CSC 200 course (with Craig Boutilier) but mainly when I am in (which is most of the time), my door is open and I welcome questions (unless I am preoccupied). So feel free to drop by and/or email me to schedule a time. My office is SF 2303B and my email is bor@cs.toronto.edu. The course web page is www.cs.toronto.edu/~bor/2420f12

What is appropriate background?

- In short, a course like our undergraduate CSC 373 is essentially the prerequisite. Upon request, I will make available (hard copy) last years CSC 373 final exam as one way to test yourself.
- Any of the popular undergraduate texts.
- It certainly helps to have a good math background and in particular understand basic probability concepts.

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

Reviewing some basic algorithmic paradigms

The conceptually simplest algorithms

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

- brute force
- greedy
- local search

Comment

- We usually dismiss brute force search as it really isn't much of an algorithm approach but for small enough problems it might be the way to go.
- Moreover, sometimes we can combine some aspect of brute force search with another approach as we will see by combining brute force and greedy.

Greedy algorithms in CSC373

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

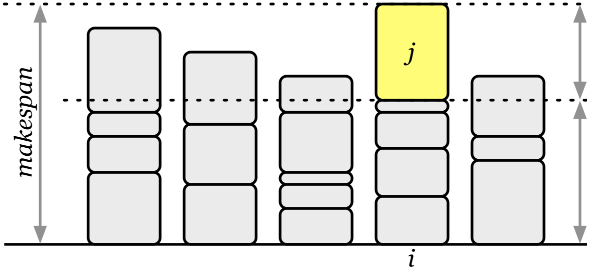
- The optimal algorithm for the fractional knapsack problem and the approximate algorithm for the proportional profit knapsack problem.
- The optimal unit profit interval scheduling algorithm and 3-approximation algorithm for proportional profit interval scheduling.
- The 2-approximate algorithm for 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 approximation algorithm for unweighted vertex cover.
- The approximation algorithms for $(k+1)$-claw free graphs.
- The approximation algorithm for set cover.

Greedy algorithms: Graham's online and LPT makespan algorithms

- Let's start with these two greedy algorithms that date back to 1966 and 1969 technical reports.
- 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 the start of approximation algorithms. Moreover, there are some general concepts to be observed in this work and even after some 45 years still some open questions concerning such makespan problems.

The makespan problem

- 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 $\max_k \left(\sum_{\ell:\sigma(\ell)=k} p_\ell\right)$.

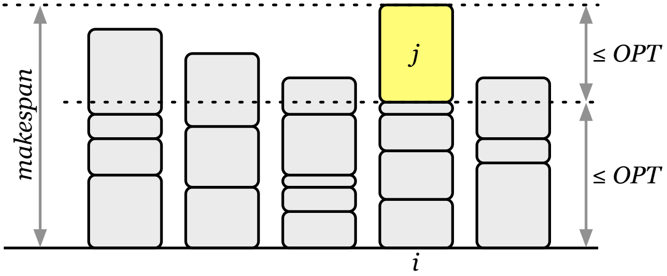


[picture taken from Jeff Erickson's lecture notes]

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:**



[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.

- Recall that the **approximation ratio** for this algorithm is $2 - \frac{1}{m}$; that is, for any set of jobs $\mathcal{J}$, $C_{Greedy}(\mathcal{J}) \leq (2 - \frac{1}{m}) C_{OPT}(\mathcal{J})$.
- 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: I will usually not provide proofs in the lecture notes but rather will do most proofs in class.

- 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 a greedy algorithm is. We will present the priority model for greedy (and greedy-like) algorithms. I claim that all the algorithms mentioned on slide 5 can be formulated within the priority model.
- Asssuming we maintain a priority queue for the least loaded machine,
  - the online greedy algorithm would have time complexity $O(n \log m)$
  - the LPT algorithm would have time complexity $O(n \log n)$ since we can assume $n \geq m$.

Brute force plus greedy

- Combining the LPT greedy algorithm with a brute force approach improves the approximation ratio but at a significant increase in time complexity.

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) for any fixed m. 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 makespan problem.
- The online greedy gives the best online ratio for m = 2,3 but better bounds are known for $m \geq 4$.

  Basic idea: leave some room for a possible large and this forces the online algorithm to be non-greedy in some sense but still within the priority model which subsumes online algorithms.
- Randomization can provide somewhat better competitive ratios.
- 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
- Recall that each 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., and recently Ebenlendr and Sgall establish an online inapproximation of 2.564 following the 2.428 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 recent progress in the offline restricted machines case.
- Even for the case of two allowable machines per job, this is an interesting problem and we will probably look at recent work later in the term.
- 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 that 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 (i.e. the graph orientation problem).

The unrelated machines model

- The most general of the 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.
- The LST algorithm is still the best known algorithm even for the restricted machines model although there has been some progress made in this regard (which we will discuss later).

The knapsack problem

Input: A 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 $S \subseteq \{1, \ldots, n\}$ satsifying $\sum_{j \in S} s_j \leq C$ so as to maximize $V(S) = \sum_{j \in S} v_j$.
Note: I will usually 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. $ALG \geq \rho OPT$.
It is easy to see 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 partial enumeration greedy PTAS for knapsack

```
$PGreedy_k$ Algorithm:
Sort $\mathcal{I}$ so that $\frac{v_1}{s_1} \geq \frac{v_2}{s_2} \ldots \geq \frac{v_n}{s_n}$
For every feasible subset $H \subseteq \mathcal{I}$ with $|H| \leq k$
    Let $R = \mathcal{I} - H$ and let $S_H := H$
    Consider items in $R$ (in the order of profit densities)
    and greedily add items to $S_H$ that do not exceed knapsack capacity $C$.
        % It is sufficient to stop as soon as the first item is too large to fit
End For
Choose the maximum profit $S_H$.
```

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 (i.e. a PTAS) 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 this algorithm was quickly subsumed but still the partial enumeration technique is important.
- In particular, more recently this technique (for $k = 3$) was used 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 and hence this is also the best possible ratio for submodular maximization subject to a knapsack constraint.