CSC2420 Fall 2012: Algorithm Design, Analysis and Theory

Allan Borodin

September 20, 2012

Lecture 2

We continue where we left off last lecture, namely we are considering a PTAS for the 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 S_H that maximizes the 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 (i.e. a PTAS) running in time $\frac{1}{\epsilon}n^{\frac{1}{\epsilon}}$.
- To obtain a 2-approximation it suffices to maximize between the largest value item and the value of the profit density greedy algorithm.
- 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.

The priority algorithm model and variants

Before temporarily leaving 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 mentioned last class)? With the exception of Huffman coding (which we can also deal with) all these algorithms consider one input item 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.

- Note: that a problem is only fully specified when we say how input items are represented.
- 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. 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 more general than saying it must place the item on the least loaded machine.

- 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 then is to indicate how the algorithm chooses the order in which input items are considered. We cannot allow the algorithm to use any ordering (as there may very well be an optimal ordering that allows the algorithm to achieve optimality). 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 polynomial time. Instead, we take an information theoretic viewpoint in defining the orderings we allow.

Informal definition of a priority algorithm

Lets first consider fixed 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. Finally, 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.

The fixed (order) priority algorithm template

$\mathcal{J}$ is the set of all possible input items
$\mathcal{I} \subset \mathcal{J}$ is the input instance
decide on a total ordering π of $\mathcal{J}$
$S := \varnothing$ % 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 index 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.
- 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 proofs 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

Inapproximations with respect to the priority model

Once we have a precise model, we can then argue that certain approximation bounds are not possible within this model. Such inapproximation results have been estalished with respect to (adaptive) priority algorithms for a number of problems but in many cases much better results can be established using extensions of the model.

1. For the weighted interval selection (a *packing problem*) with arbitrary weighted values (resp. for proportinal weights $v_j = |f_j - s_j|$), no priority algorithm can achieve a constant approximation (respectively, better than a 3-approximation).
2. For the knapsack problem, no priority algorithm can achieve a constant approximation. (We have already noted how partial enumeration greedy can achieve a PTAS.)
3. For the set cover problem, the natural greedy algorithm is essentially the best priority algorithm.
4. As previously mentioned, for fixed order priority algorithms, there is an $\Omega(\log m \log \log m)$ inapproximation bound for the makespan problem in the restricted machines model.

Extensions of the priority model: priority with revocable acceptances

- For packing problems, we can have priority algorithms with revocable acceptances. That is, in each iteration the algorithm can now eject previously accepted items in order to accept the current item. However, at all times, the set of currently accepted items must be a feasible set and all rejections are permanent.
- Within this model, there is a 4-approximation algorithm for the weighted interval selection problem WISP (Bar-Noy et al [2001], and Erlebach and Spieksma [2003]) and a ≈ 1.17 inapproximation bound (Horn [2004]). More generally, the algorithm applies to the weighted job interval selection problem WJISP resulting in an 8-approximation.
- The model has also been studied with respect to the proportional profit knapsack problem/subset sum problem (Ye and B [2008]) improving the constant approximation. And for the general knapack problem, the model immediately yields a 2-approximation.

The $Greedy_\alpha$ algorithm for WJISP

The algorithm as stated by Erlebach and Spieksma (and called ADMISSION by Bar Noy et al) is as follows:

$S := \varnothing$ % S is the set of currently accepted intervals
Sort input intervals so that $f_1 \leq f_2 \ldots \leq f_n$
for $i = 1..n$
    C_i := min weight subset of $S : (S/C_i) \cup \{I_i\}$ feasible
    if $v(C_i) \leq \alpha \cdot v(I_i)$ **then**
        $S := (S/C_i) \cup \{I_i\}$
    end if
END FOR

Figure: Priority algorithm with revocable acceptances for WJISP

The $Greedy_\alpha$ algorithm (which is not greedy by my definition) has a tight approximation ratio of $\frac{1}{\alpha(1-\alpha)}$ for WISP and $\frac{2}{\alpha(1-\alpha)}$ for WJISP.