CSC2420 Spring 2015: Lecture 2

Allan Borodin

January 15,2015

Announcements and todays agenda

- First part of assignment 1 was posted last weekend. I plan to assign one or two more questions.
- Today will be a short lecture without a break. If possible I will try to find extra time later in the term.
- I try to post the slides within a day or so of the lecture and usually post what was discussed. This week, I have posted all the intended slides although we only discussed slides 2-9, and 17-21. The ither slides will be discussed next week.
- Todays agenda
  1. Some more comments on set packing. Obtaining an $O(\sqrt{m})$-approximation.
  2. $(k+1)$-claw free graphs.
  3. Priority algorithms with revocable acceptance (for packing problems). The “greedy” algorithm for weighted job interval scheduling.
  4. Priority stcak algorithms. Not discussed
  5. Chordal graphs. Not discussed
  6. The random order model (ROM)
     i
  7. Start dynmaic programming Not discussed

Greedy algorithms for the set packing problem

The set packing problem

We are given n subsets $S_1, \ldots, S_n$ from a universe U of size m. In the weighted case, each subset S_i has a weight w_i. The goal is to choose a disjoint subcollection $\mathcal{S}$ of the subsets so as to maximize $\sum_{S_i \in \mathcal{S}} w_i$. In the s-set packing problem we have $|S_i| \leq s$ for all i.

- This is a well studied problem and by reduction from the max clique problem, there is an $m^{\frac{1}{2}-\epsilon}$ hardness of approximation assuming $NP \neq ZPP$. For s-set packing, there is an $\Omega(s/\log s)$ hardness of approximation assuming $P \neq NP$.
- Set packing is the underlying allocation problem in what are called combinatorial auctions as studied in mechanism design.
- We will consider two "natural" greedy algorithms for the s-set packing problem and a somewhat less obvious greedy algorithm for the set packing problem. These greedy algorithms are all fixed order priority algorithms.

The first natural greedy algorithm for set packing

Greedy-by-weight ($Greedy_{wt}$)

```
Sort the sets so that $w_1 \geq w_2 \ldots \geq w_n$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If $S_I$ does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For
```

- In the unweighted case (i.e. $\forall i, w_i = 1$), this is an online algorithm.
- In the weighted (and hence also unweighted) case, greedy-by-weight provides an s-approximation for the s-set packing problem.
- The approximation bound can be shown by a charging argument.

Two types of approximation arguments

- Recall the argument for makespan on identical machines.
  1. We identify some intrinsic limiting bounds for any solution including an OPT solution; in this case average load/machine and processing time for any job.
  2. Then we relate the algorithmic solution (in this case the natural greedy solution) to those bounding factors.
  3. We will see something similar when consider "LP rounding".
- We now consider a different type of argument. Namely a charging argument.
- We will consider this in the context of a maximization problem, namely the charging argument for $Greedy_{wt}$ for s-set packing.
  1. We will charge the weight of every set in an OPT solution to the first set in the greedy solution with which it intersects.
  2. How many sets in OPT can be charged to the same set in $Greedy_{wt}$?
  3. If say set $S_i \in OPT$ is being charged to $S_j \in Greedy_{wt}$, then we know $w_i \leq w_j$.

The second natural greedy algorithm for set packing

Greedy-by-weight-per-size

```
Sort the sets so that $w_1/|S_1| \geq w_2/|S_2| \ldots \geq w_n/|S_n|$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If $S_I$ does not intersect any set in $\mathcal{S}$ then
       $\mathcal{S} := \mathcal{S} \cup S_i$.
End For
```

- In the weighted case, greedy-by-weight provides an s-approximation for the s-set packing problem.
- For both greedy algorithms, the approximation ratio is tight; that is, there are examples where this is essentially the approximation. In particular, greedy-by-weight-per-size is only an m-approximation where $m = |U|$.
- We usually assume $n >> m$ and note that by just selecting the set of largest weight, we obtain an n-approximation.

Improving the approximation for greedy set packing

- In the unweighted case, greedy-by-weight-per-size can be restated as sorting so that $|S_1| \leq |S_2| \ldots \leq |S_n|$ and it can be shown to provide an $\sqrt{m}$-approximation for set packing.
- On the other hand, greedy-by-weight-per-size does not improve the approximation for weighted set packing.

Greedy-by-weight-per-squareroot-size

```
Sort the sets so that $w_1/\sqrt{|S_1|} \geq w_2/\sqrt{|S_2|} \ldots \geq w_n/\sqrt{|S_n|}$.
$\mathcal{S} := \varnothing$
For $i : 1 \ldots n$
    If $S_I$ does not intersect any set in $\mathcal{S}$ then
        $\mathcal{S} := \mathcal{S} \cup S_i$.
End For
```

Theorem: Greedy-by-weight-per-squareroot-size provide a $\sqrt{m}$-approximation for the set packing problem. And as noted earlier, this is essentially the best possible approximation assuming $NP \neq ZPP$.

Another way to obtain an $O(\sqrt{m})$ approximation

There is another way to obtain the same aysmptototic improvement for the weighted set packing problem. Namely, we can use the idea of partial enumeration greedy; that is somehow combining some kind of brute force (or naive) approach with a greedy algorithm.

Partial Enumeration with Greedy-by-weight ($PGreedy_k$)

Let Max_k be the best solution possible restricting solutions to those of cardinality at most k. Let G be the solution obtained by $Greedy_{wt}$ applied to sets of cardianlity at most $\sqrt{m/k}$. Set $PGreedy_k$ to be the best of Max_k and G.

- Theorem: $PGreedy_k$ achieves a $2\sqrt{m/k}$-approximation for the weighted set packing problem (on a universe of size m)
- In particular, for $k = 1$, we obtain a $2\sqrt{m}$ approximation and this can be improved by an arbitrary constant factor $\sqrt{k}$ at the cost of the brute force search for the best solution of cardinality k; that is, at the cost of say n^k.

$(k+1)$-claw free graphs

A graph $G = (V, E)$ is $(k+1)$-claw free if for all $v \in V$, the induced subgraph of $Nbhd$(v) has at most k independent vertices (i.e. does not have a $k+1$ claw as an induced subgraph).

$(k+1)$-claw free graphs abstract a number of interesting applications.

- In particular, we are interested in the (weighted) maximum independent set problem (W)MIS for $(k+1)$-claw free graphs. Note that it is hard to approximate the MIS for an arbiitrary n node graph to within a factor $n^{1-\epsilon}$ for any $\epsilon > 0$.
- We can (greedily) k-approximate WMIS for $(k+1)$-claw free graphs.
- The (weighted) k-set packing problem is an instance of (W)MIS on $k+1$-claw free graphs. What algorithms generalize?
- There are many types of graphs that are $k+1$ claw free for small k; in particular, the intersection graph of translates of a convex object in the two dimensional plane is a 6-claw free graph. For rectangles, the intersection graph is 5-claw free.

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 allows 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:

```
$S$ := $\emptyset$          % $S$ 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.t. $(S/C_i) \cup \{I_i\}$ 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.

Priority Stack Algorithms

- For packing problems, instead of immediate permanent acceptances, in the first phase of a priority stack algorithm, items (that have not been immediately rejected) can be placed on a stack. After all items have been considered (in the first phase), a second phase consists of popping the stack so as to insure feasibility. That is, while popping the stack, the item becomes permanently accepted if it can be feasibly added to the current set of permanently accepted items; otherwise it is rejected. Within this priority stack model (which models a class of primal dual with reverse delete algorithms and a class of local ratio algorithms), the weighted interval selection problem can be computed optimally.
- For covering problems (such as min weight set cover and min weight Steiner tree), the popping stage is insure the minimality of the solution; that is, while popping item I from the stack, if the current set of permanently accepted items plus the items still on the stack already consitute a solution then I is deleted and otherwise it becomes a permanently accepted item.

Chordal graphs and perfect elimination orderings

An interval graph is an example of a chordal graph. There are a number of equivalent definitions for chordal graphs, the standard one being that there are no induced cycles of length greater than 3.

We shall use the characterization that a graph $G = (V, E)$ is chordal iff there is an ordering of the vertices $v_1, \ldots, v_n$ such that for all i, $Nbdh(v_i) \cap \{v_{i+1}, \ldots, v_n\}$ is a clique. Such an ordering is called a perfect elimination ordering (PEO).

It is easy to see that the interval graph induced by interval intersection has a PEO (and hence is chordal) by ordering the intervals such that $f_1 \leq f_2 \ldots \leq f_n$. Using this ordering we know that there is a greedy (i.e. priority) algorithm that optimally selects a maximum size set of non intersecting intervals. The same algorithm (and proof by charging argument) using a PEO for any chordal graph optimally solves the unweighted MIS problem. The following priority stack algorithm provides an optimal solution for the WMIS problem on chordal graphs.

The optimal priority stack algorithm for the weighted max independent set problem (WMIS) in chordal graphs

```
$Stack$ := $\varnothing$          % $Stack$ is the set of items on stack
Sort input intervals so that $f_1 \leq f_2 \ldots \leq f_n$
**For** $i = 1..n$
    $C_i$ := nodes on stack that are adjacent to $v_i$
    **If** $w(v_i) > w(C_i)$ **then** push $v_i$ onto stack, **else** reject
**End For**
$S$ := $\varnothing$              % $S$ will be the set of accepted nodes
**While** $Stack \neq \varnothing$
    Pop next node $v$ from $Stack$
    **If** $v$ is not adjacent to any node in $S$, **then** $S$ :=$S \cup \{v\}$
**End While**
```

Figure : Priority stack algorithm for chordal WMIS

A k-PEO and inductive k-independent graphs

- An alternative way to describe a PEO is to say that $Nbhd(v_i) \cap v_{i+1}, \ldots, v_n\}$ has independence number 1.
- We can generalize this to a k-PEO by saying that $Nbhd(v_i) \cap v_{i+1}, \ldots, v_n\}$ has independence number at most k.
- We will say that a graph is an inductive k-independent graph is it has a k-PEO.
- Inductive k-independent graphs generalize both chordal graphs and $k + 1$-claw free graphs.
- The intersection graph induced by the JISP problem is an inductive 2-independent graph.
- The intersection graph induced by axis parallel rectangles in the plane are inductive 4-independent.
- The priority stack algorithm is a k-approximation algorithm for WMIS wrt inductive k-independent graphs.

More extensions of the priority model

- So far we have been implicitly assuming deterministic priority algorithms. We can allow the ordering and/or the decisions to be randomized.
- A special case of fixed priority with randomized orderings is when the input set is ordered randomly without any dependence on the set of inputs. In the online setting this is called the random order model.
- The revocable acceptances model is an example of priority algorithms that allow reassignments (of previous decisions) to some extent or at some cost.
- The partial enumeration greedy is an example of taking the best of some small set of priority algorithms.
- Priority stack algorithms are an example of 2-pass (or multi-pass) priority algorithms where in each pass we apply a priority algorithm. Of course, it has to be well specified as to what information can be made available to the next pass.

The random order model (ROM)

Motivating the random order model

The random order model provides a nice compromise between the often unrealistic negative results for worst case (even randomized) online algorithms and the often unrealistic setting of inputs being generated by simple distributions.

- In many online scenarios, we do not have realistic assumptions as to the distributional nature of inputs (so we default to worst case analysis). But in many appliications we can believe that inputs do arrive randomly or more precisely uniformly at random.
- The ROM can be (at least) traced back to what is called the (classical) secretary (aka marriage or dowry) problem, popularized in a Martin Gardner Scientific American article.
- As Fiat et al (SODA 2015) note, perhaps Johannes Kepler (1571-1630) used some secretary algorithm when interviewing 11 potential brides over two years.

The secretary problem

The classical problem (which has now been extended and studied in many different variations is as follows:

- The classic problem (as in the Gardiner article) assumes an adversarially chosen set of distinct values for (say N) items that arrive in random order (e.g. candidates for a position, offers for a car, etc.). N is assumed to be known.
- Once an item (e.g. secretary) is chosen, that decision is irrevocable. Hence, this boils down to finding an *optimal stopping rule*, a subject that can be considered part of stochastic optimization.
- The goal is to select one item so as to maximize the probability that the item chosen is the one of maximum value.
- For any set of N values, maximizing the probability of choosing the best item immediately yields a bound for the expected (over the random orderings) value of the chosen item. For an "ordinal algorithm", these two measures are essentially the same. Why?

The secretary problem continued

- It is not difficult to show that any deterministic or randomized (adversarial order) online algorithm has competitive ratio [1] at most $O(\frac{1}{N})$. Hence the need to consider the ROM model to obtain more interesting (and hopefully more meaningful) results.
- We note (and this holds more generally) that "positive results" for the ROM model subsume the stochastic optimization scenario where inputs are generated by an unknown (and hence known) i.i.d. process. Why?
- There are many variations and extensions of the secretary problem some of which we will consider.
- In general, any online problem can be studied with respect to the ROM model.

[1] Recall that for maximization problems, competitive and approximation ratios can sometimes presented as fractions $\alpha = \frac{ALG}{OPT} \leq 1$ and sometimes as ratios $c = \frac{OPT}{ALG} \geq 1$. I will try to follow the convention mainly used in each application.

The optimal stopping rule for the classical secretary problem

The amusing history of the secretary problem and the following result is taken up by Ferguson in a 1989 article.

Theorem: For N and r, there is an exact formula for the probability of selecting the maximum value item after observing the first r items, and then selecting the first item (if any) that exceeds the value of the items seen thus far. In the limit as $N \to \infty$, the optimal stopping rule is to observe (i.e. not take) the first $r = N/e$ items. The probability of obtaining the best item is then $1/e$ and hence the expected value of the item chosen is at least $\frac{1}{e} v_{max}$.

Variations and extensions of the secretary problem

- Instead of maximizing the probability of choosing the best item, we can maximize the expected rank of the chosen item.
- Perhaps the most immediate extension is to be choosing k elements.
- This has been generalized to the matroid secretary problem by Babaioff. For arbitrary matroids, the approximation ratio remains an open problem.
- Another natural extension is to generalize the selection of one item to the online (and ROM) edge weighted bipartite matching problem, where say $N = |L|$ items arrive online to be matched with items in R. In online matching the goal is usually to maximize the size (for the unweighted case) or weight of a maximum matching.
- I plan to discuss online matching and the extension to the adwords problem where the online nodes L represent advertisers/bidders with budgets and preferences/values for the R nodes representing keywords/queries.

Dynamic programming and scaling

We now move on to one of the main objects of study in an undergraduate algorithms course.

- We have previously seen that with some use of brute force and greediness, we can achieve PTAS algorithms for the identical machines makespan which is polynomial in the number n of jobs but exponential in the number m of machines and $\frac{1}{\epsilon}$ where $1+\epsilon$ is the approximation guarantee.
- For the knapsack problem we had a PTAS that was polynomial in n and exponential in $\frac{1}{\epsilon}$. .
- We now consider how dynamic programming (DP) can be used to acheive a PTAS for the makespan problem which is polynomial in m and n, and how to achieve an FPTAS for the knapsack problem. To achieve these improved bounds we will combine dynamic programming with the idea of scaling inputs to improve the results for knapsack and identical machine makespan.
- NOTE: Defining “useful” precise models of DP algorithms is challenging.

An FPTAS for the knapsack problem

Let the input items be $I_1, \ldots, I_n$ (in any order) with $I_k = (v_k, s_k)$. The idea for the knapsack FPTAS begins with a "pseudo polynomial" time DP for the problem, namely an algorithm that is polynomial in the numeric value v (rather than the encoded length $|v|$) of the input values.

Define $S[j, v]$ = the minimum size s needed to achieve a profit of at least v using only inputs $I_1, \ldots I_j$; this is defined to ∞ if there is no way to achieve this profit using only these inputs.

This is the essense of DP algorithms; namely, defining an approriate generalization of the problem (which we give in the form of an array) such that

1. the desired result can be easily obtained from this array
2. each entry of the array can be easily computed given "previous entries"

How to compute the array $S[j, v]$ and why is this sufficient

1. The value of an optimal solution is $max\{v | S[n, v] \text{ is finite}\}$.
2. We have the following equivalent recursive definition that shows how to compute the entries of $S[j, v]$ for $0 \leq j \leq n$ and $v \leq \sum_{j=1}^{n} v_j$.
   - Basis: $S[0, v] = \infty$ for all v
   - Induction: $S[j, v] = \min\{A, B\}$ where $A = S[j-1, v]$ and $B = S[j-1, \max\{v - v_j, 0\}] + s_j$.
3. It should be clear that while we are computing these values that we can at the same time be computing a solution corresponding to each entry in the array.
4. For efficiency one usually computes these entries iteratively but one could use a recursive program with *memoization*.
5. The running time is $O(n, V)$ where $V = \sum_{j=1}^{n} v_j$.
6. Finally, to obtain the FPTAS the idea (due to Ibarra and Kim [1975]) is simply that the high order bits/digits of the item values give a good approximation to the true value of any solution and scaling these values down (or up) to the high order bits does not change feasibility.

The better PTAS for makespan

- We can think of m as being a parameter of the input instance and now we want an algorithm whose run time is poly in m, n for any fixed $\epsilon = 1/s$.
- The algorithm's run time is exponential in $\frac{1}{\epsilon^2}$.
- We will need a combinaton of paradigms and techniques to achieve this PTAS; namely, DP and scaling (but less obvious than for the knapsack scaling) and binary search.

The high level idea of the makespan PTAS

- Let T be a candidate for an achievable makespan value. Depending on T and the ϵ required, we will scale down "large" (i.e. if $p_i \geq T/s = T \cdot \epsilon$) to the largest multiple of T/s^2 so that there are only $d = s^2$ values for scaled values of the large jobs.
- When there are only a fixed number d of job sizes, we can use DP to test (and find) in time $O(n^{2d})$ if there is a soluton that achieves makespan T.
- If there is such a solution then small jobs can be greedily scheduled without increasing the makespan too much.
- We use binary search to find a good T.

The optimal DP for makespan on identical machines when there is a fixed number of job values

- Let $z_1, \dots, z_d$ be the d different job sizes and let $n = \sum n_i$ be the total number of jobs with n_i being the number of jobs of szie z_i.
- The array we will use to obtain the desired optimal makespan is as follows:
  $M[x_1, \dots, x_d]$ = the minimum number of machines needed to schedule ix_i jobs having size z_i within makespan T. (Here we can assume $T \geq \max p_i \geq \max z_i$ so that this minimum is finite.)
- The n jobs can be scheduled within makespan T iff $M[n_1, , n_d]$ is at most m.