Advanced Algorithms 10/10, 2017

Lecture 4

Lecturer: Mohsen Ghaffari *Scribe: Markos Karasamanis*

NOTE: This is an initial draft of the scribe notes, and it has not been double checked by the instructor.

1 Previous Lecture

In the previous lecture we saw that a greedy approach—making locally optimal decisions at each step of the algorithm—led us to solutions that were not too far from the optimal in the worst case. We saw: a $(\ln(n) + O(1))$-approximation of set cover all the way down to Graham's $\frac{4}{3}$-approximation of sorted list scheduling. Here we consider trading off running time for accuracy.

2 Approximation Schemes

We define the two ways to formalise the running time of such algorithms that trade running for accuracy.

Definition 1 (PTAS)**.** *A Polynomial Time Approximation Scheme is a family of Algorithms* $\{\mathcal{A}_\epsilon\}_{\epsilon>0}$ *s.t.* $\forall\epsilon > 0$ $\mathcal{A}$ *is a* $(1 \pm \epsilon)$*-approximation algorithm that is poly time in the size of the problem.*

Definition 2 (FPTAS)**.** *A Fully Polynomial Time Approximation Scheme is a PTAS that is poly time in both the size of the problem and* $\frac{1}{\epsilon}$

Plan: Show FPTAS of the Knapsack problem, Bin Packing and Minimum Makespan Scheduling

Genereally, we will restrict our algorithm analysis to a family of instances of the problem and show that an algorithm is efficient on those, then show that we can transition from any instance to one belonging to said family with bounded error concluding a proof. Coarse graining ideas

3 Knapsack

We begin to illustrate the concept of approximation schemes concretely with the aid of the Knapsack problem. You can verify that a greedy approach here is not very good, a 2-approximation is possible (see excercises).

Definition 3 (Knapsack Problem)**.** *Given a list of items* $\{a_i\}$*, functions* $size(a_i), value(a_i) \in \mathbb{Z}^+$ *and a constant* C*, return* $R = arg \max\limits_{I\subset\{a_i\}} \sum_{i\in I} value(a_i) | \sum_{i\in I} size(a_i) \leq C$

Start to look for instances solvable in polynomial time by taking the following assumptions: $P = \max\limits_{i\in\{a_i\}} profit(a_i) \leq poly(n)$ or ,equivalently, that a poly(n,P) runtime suffices. As such the Knapsack problem is solvable in poly(n,P) time using dynamic programming. The algorithm is shown in pseudo-code–closely following the topological sort and dynamic programming recurrence. The running time is $O(n^2P)$:

```
while i ∈ $\{n, n-1\cdots 0\}$ do
    while J ∈ $\{0\cdots S\}$ do
        if $i == n$ then
            $dp[i][j] = 0$
        else
            choices = []
            APPEND(choices, dp[i+1][j])
            if $j \geq s_i$ then
                APPEND(choices, $dp[i+1][j-s_i] + v_i$)
            end if
            dp[i][j]=MAX(choices)
        end if
      end while
    end while
    return dp[0][S]
```

Now since we have a $O(poly(n, P))$ algorithm from the above DP it would suffice to scale down all the values such that P is small and show that this does not mess up the solution too much—this woulld give us the FPTAS:

Theorem 4. *We obtain an FPTAS for the Knapsack problem with the following modification to the dynamic program: Let $\lfloor \frac{n}{\epsilon} \rfloor = k$ for some $\epsilon > 0$, for every a_i $p'_i = \lfloor \frac{p_i}{P} k \rfloor$. Return the output of the dynamic program above after replacing $\{p'_i\}$ for $\{p_i\}$ in the input.*

Proof. LEt OPT be the optimal profit in the original instance then optimal profit in the modified instance is $\sum_{i \in S'} p'_i \geq \sum_{i \in OPT} p'_i$. Also, $\sum_{i \in S'} p_i \geq \frac{P}{k} \sum_{i \in S'} (p'_i = \lfloor \frac{p_i}{P} k \rfloor)$, if you wanted to scale the profit back up. Now we bound the profit of S' w.r.t OPT, dropping the floor function and substituting we get $\frac{P}{k} \sum_{i \in OPT} (p'_i = \lfloor \frac{p_i}{P} k \rfloor) \geq \sum_{i \in OPT} p_i - \epsilon P \geq (1 - \epsilon) OPT$, where we obtain the last inequality from $\epsilon P \leq \epsilon OPT$. Combining, the inequalities concludes the proof for approximation and by definition $P' = \frac{1}{\epsilon}$ resulting in the desired complexity. □

4 Bin Packing

Note: In this example we relax the notion of PTAS to holding only asympotically—for large values of OPT— to avoid the issue created by the hardness of approximation result of bin packing.

Theorem 5. *A $(\frac{3}{2} - \epsilon)$-approximation for $\epsilon > 0$ of Bin Packing implies P=NP*

The proof of this theorem follows if we use such a hypothetical PTAS as an oracle for the Subset Sum problem—one of Karp's original NP-Complete problems.

We tackle the problem with the general approach of searching for easier to solve instances—for example: the greedy solution is good if the items are small (Hint: greedy for any item size can give a 2-approximation. Why?). So small item values are not a problem, we will see now that if the smallest size and the number of different sizes is bouned then we have an exact polynomial algorithm.

Lemma 6. *If there are only k different sizes and the smallest is size ϵ then there exists an exact algorithm for Bin Packing in $O(n^{k^{\frac{1}{\epsilon}}})$*

Proof. Do an exhaustive search of bin packing and observe that there do not exist many configurations to consider. Each bin can take up to $k^{\frac{1}{\epsilon}}$ configurations and there are at most n of them with that configuration, giving $\binom{n+k^{\frac{1}{\epsilon}}}{k^{\frac{1}{\epsilon}}}$ since ordering is irrelevant. □

Now we try to convert any instance to the above without messing up the accuracy too much. And we will also ignore small values for now–as discussed we already have a PTAS for them. Naturally, we will group the items—say in Q groups— so their values are coarse grained to the bounded number of sizes we need. The next two lemmas show that neither grouping nor ignoring small values affect the output too much.

Lemma 7. *For $S = \{a_i | a_i < \epsilon\}$ and $L = \{a_i | a_i \geq \epsilon\}$ any packing of L into l bin can be extended to a packing of $S \cup L$ with size $= \max\{l, (1 + \epsilon)OPT + 1\}$*

Proof. Add elements of S greedily into the bins already used to pack L. If you require extra bins and reached some number k+1 of bins then k bins were at least $(1 - \epsilon)$ full otherwise your item would fit there.Then, k $(1 - \epsilon) \leq$ size$(S \cup L) \rightarrow k + 1 \leq (1 + \epsilon)OPT + 1$, where we used that size$(S \cup L)$ is a trivial lower bound for OPT and that $\frac{1}{1-\epsilon} = 1 + O(\epsilon)$ □

Suppose the instance I' is produced from an instance I when items are ordered— say the item with label 1 has the largest value — and grouped into k equal groups by rounding up to the largest value of the group—the items in Q_1 have the largest value. Lastly, Q_1 is discarded to form I'.

Lemma 8. $OPT(I') \leq OPT(I) \leq OPT(I') + q$

Proof. We can use the packing of I to pack I' because we have dropped the group with largest value, thus each item in I' has value at most that of the item with equal label in I, so $OPT(I') \leq OPT(I)$. But we can also pack the items in the group we dropped by introducing at most q new bins □

Theorem 9. *There exists an asymptotic PTAS for bin packing*

Proof. Follows from the two previous lemmas. □

5 Minimum Makespan Scheduling

Return to the Minimum Makespan Scheduling Problem from last lecture. The procedure is not too different from the bin packing we just showed.

Lemma 10. *Instances of Minimum Makespan Scheduling with bounded different job sizes are easy to solve.*

Proof. The number of possible instances is $\leq n^k$ for k different job sizes. Let $\{s_1 \cdots s_k\}$ be the set of all instances for which the minimum number of machines—$M(s_1 \cdots s_k)$— able to complete them in time T is 1. Where an instance is defined by the number of jobs of each job size.

$$M(a_1 \cdots a_k) = 1 + \min_S M(a_1 - S_1 \cdots a_k - S_k)$$

The Dynamic Program using the recursion above runs in $O((n^k)^2)$, using it in a binary search over the number of machines needed to complete an instance in time T determines the optimal schedule in polynomial time.

□

Lemma 11. *Minimum Makespan Scheduling can be* $(1+\epsilon)$*-approximated to an instance with bounded different job sizes.*

Proof. Let $\epsilon T, \epsilon^2 T, 1/\epsilon, 1/\epsilon^2$ be integers. If all jobs have size $\geq \epsilon T$ we scale them down to the nearest integer multiple of $\epsilon^2 T$. Then DP finds an optimal schedule completing within T. An optimal schedule found after scaling is also optimal before scaling: job sizes are scaled down by at most $\epsilon^2 T$ and there is an integer number $1/\epsilon^2$ of re-scaled job types. Using this schedule on the original input the load on any machine increases by at most ϵT If there are jobs with size $\leq \epsilon T$ then we add them to the schedule we would have found if we ignored them using list scheduling, if we cant we try the next value of T in binary search. □

Theorem 12. *There exists a PTAS for Minimum Makespan Scheduling*

Proof. Follows from the two previous lemmas. □

References