CSC2420: Algorithm Design, Analysis and Theory **Fall 2017**

Lecture 3: Introductions

Lecturer: Allan Borodin, Nisarg Shah *Scribe: Lily Li*

3.1 Continuation of Greedy, Greedy-Like Algorithms

3.1.1 Priority Stack Algorithm

Recall: you look at the input item and make an immediate solution to put it on a stack or throw it away. The resulting stack might not be feasible, so you pop the stack to ensure feasibility. For covering problems, the popping operation needs to insure minimality.

Example 3.1 *Chordal graphs and perfect elimination ordering. Note an interval graph is an example of a chordal graph (which are example of perfect graph — chromatic number of induced subgraph equals size of largest clique in subgraph).*

Start with empty stack, sort nodes as in PEO (perfect elimination ordering), for all starting at node v_1... they he moved on.

Basic idea: put on stack if the value is better (not necessarily that much better, just the residual is better).

Example 3.2 ***Interval Coloring Problem*** *Given set of intervals, color all intervals so that interval having same color does not intersect. Want to use as few colors as possible where number of colors is m.*

One possible solution is to iterate the problem for increasing m. Stopping when the scheduling is possible. Unfortunately this is not very efficient.

Here is a better way (best actually): consider the earliest starting time for each interval. Use a color you used if possible. Otherwise given the interval a new color. To see why this works, think about the associated graph. You only add new colors when there is a clique using all previous colors.

3.1.2 m-machine Interval Scheduling

3.2 Dynamic Programming

Example 3.3 *In the makespan problem let n be the number of jobs and m be the number of machines and $1/\epsilon$ where $1+\epsilon$ is your approximation.*

Recall the essence of DP algorithm is optimum substructure. Note that we don't know that if: everything that can be done in polynomial time can be done using dynamic programming (this is true if you swap in greedy for DP). Maybe think about how to do a perfect matching in a bipartite graph using DP in polynomial time. (Trivia: Bellman came up with dynamic programming! He says: don't formalize it.)

Example 3.4 ***Weighted Interval Scheduling Problem (WISP):*** *here you want to work on the head subset of the intervals. Consider what is the optimum value you could get from the first i intervals, $V[i]$. We want $V[n]$ (n being the total number of items). Borodin makes a distinction between the semantic array $V[\cdot]$ and $\tilde{V}[\cdot]$ which he calls a* computation or recursive array*.*

With m machines the lower bound on the problem is $\Omega(n^m)$.

Note that the distinction between DP and divide-and-conquer is the difference between non-repeating and repeating sub-problems.

Example 3.5 *DP algorithm for knapsack. Yay! Recall if C is the capacity (an integer) running time is $O(nC)$. That was a PTAS. Now lets try for a FPTAS. $S[j, v]$ it he minimum size s needed to achieve a certain value v from items $v_1, ..., v_j$. Running time is $O(nV)$. To achieve the FPTAS you just scale the values down decreasing the total value.*

Lets try something similar for the makespan problem:

Example 3.6 *Let T be a candidate achievable makespan. Depending on T and ϵ, scale down large jobs ($p_i \geq T/s = T \cdot \epsilon$) to largest multiple of T/s^2. Eventually only s^2 large values. Binary search on T.*

Even though Bellman said "don't do it": there are models for dynamic programming. Consider priority branching tree (pBT) model: leveled tree where each path is a priority algorithm. The tree branches according to different decisions and continue down the path or terminate. Three cases:

1. Fixed order pBT (with online as a special case): make order initially. Each item corresponds to a level.
2. Adaptive order pBT: at each level node can choose what to look at. Every node on the same level sees the same things.
3. Strong adaptive: same as the above but nodes in the same level see different things.