6.854 Advanced Algorithms **Petar Maymounkov**

Problem Set 8 (November 2, 2005)

With: Benjamin Rossman, Oren Weimann, and Pouya Kheradpour

Problem 1. We want $\pi P = \pi$, where $\pi \geqq 0, \pi \neq 0$. If such a π exists, it is an eigenvector and hence there also exists a π for which $\sum \pi_i = 1$. Hence we want $\pi P = \pi$, where $\pi \geqq 0, \sum \pi_i = 1$. The LP for this is $\min \left\{ 0x \middle| Ax = b, x \geqq 0 \right\}$, where $A^T = \left(P - I_n \middle| 1^n \right)$, $b^T = \left(0^n \middle| 1 \right)$.

The dual is $\max \left\{ \lambda \middle| (P - I)y + \lambda \leqq 0 \right\}$. We prove that the original LP is feasible and bounded, by showing that the dual is feasible and bounded. In particular we prove that $\lambda \leqq 0$.

Pick (y, λ) such that $(P - I)y + \lambda \leqq 0$. Let y_i be the smallest entry in y, then we have $p_i y - y_i + \lambda \leqq 0$ where p_i is the i-th row of P. We also have that $y_i \leqq p_i y$, because $p_i y$ is a convex combination of the entries in y. And therefore $\lambda \leqq 0$.

Problem 2a. When Alice's strategy is known x is fixed, hence Bob is optimizing $\max_y (xA)y$. In other words Bob is trying to pick a maximum convex combination ($\sum y = 1$) of the entries of (xA). This is achieved by assigning probability 1 to any maximum entry of (xA).

Problem 2b. Part 2a implies:

$$\min_x \max_y xAy = \min_x \max_y \{xA_1, \ldots, xA_n\}$$

Where A_i is the i-th column of A. And so this is equivalent to:

$$\min \left\{ z \middle| \sum^n x_i = 1, xA_1 \leqq z, \cdots, xA_n \leqq z, x \geqq 0 \right\}$$

Alternatively, for Bob:

$$\max_y \min_x xAy = \max_y \min_x \{a_1 y, \dots, a_m y\}$$

Where a_i is the i-th row of A. And so this is equivalent to:

$$\max \left\{ w \middle| \sum^n y_i = 1, w \leqq a_1 y, \cdots, w \leqq a_m y, y \geqq 0 \right\}$$

Problem 2c. Alice's goal is to find an assignment for x that minimzes the largest of $xA_1, \dots, xA_n$. We identify the largest value by introducing the variable z and minimizing it.

Intuitively, the largest term xA_i represents Bob's fixed strategy that maximizes Bob's expected outcome regardless of which strategy Alice chooses according to her distribution x.

Problem 2d. The dual of Alice's LP after a straightforward application of Dual-taking rules is:

$$\max \left\{ w \middle| \sum^m y_i = 1, -a_1 y \geqq w, \cdots, -a_m y \geqq w, y \leqq 0 \right\}$$

Without changing the dual LP, replace y with $-y$:

$$\max \left\{ w \middle| \sum^m y_i \geqq 1, a_1 y \geqq w, \cdots, a_m y \geqq w, y \geqq 0 \right\}$$

Finally, we can replace $\sum^m y_i \geqq 1$ with $\sum^m y_i = 1$ without modifying the optimal solution, because $a_1 \geqq 0, \dots, a_m \geqq 0$, so we get the required form:

$$\max \left\{ w \middle| \sum^m y_i = 1, a_1 y \geqq w, \cdots, a_m y \geqq w, y \geqq 0 \right\}$$

Problem 3a. Algorithms: exhaustively pick a point x_i and remove all vertices H_i that are at distance $\leqq d$ from it (including itself). Do until no vertices left. Here $i < j$ if x_i was picked before x_j.

Let the optimal clustering be $C_1, \dots, C_l$. And let $x_i \in C_p$ and $x_j \in C_q$ and $i < j$. Claim that $p \neq q$. Assume otherwise, then x_i and x_j in C_r ($r = p$ or $r = q$), hence $d(x_i, x_j) \leqq d$, but then by construction $x_j \in H_i$ – contradiction. Therefore, by pigeon-hole there can be at most l centers x_i, where $l \leqq k$ is the number of optimal clusters, and after relabeling $x_i \in C_i$ for all $1 \leqq i \leqq l$.

Show that $C_i \subset H_i$ by induction. Base case: for any $y \in C_1$, $d(y, x_1) \leqq d$ (since $x_1 \in C_1$) hence by algorithm definition $y \in H_1$.

Inductive step: pick any $y \in C_i$, then $d(x_i, y) \leqq d$, therefore $y \in H_i$ or $y \in H_j$ for some $j < i$. If $y \in H_j$, then $d(y, x_j) \leqq d$, and therefore $y \in C_j$ (since $x_j \in C_j$). So $y \in C_i \cap C_j$, but $C_i \cap C_j = \emptyset$, since $i \neq j$ – contradiction.

Therefore, we've shown that the H_i's cover all the C_i's and hence they cover all vertices. Next, pick any $v, w \in H_i$, then $d(v, w) \leqq d(v, x_i) + d(w, x_i) \leqq d + d \leqq 2d$. Hence the diameter of each H_i is $\leqq 2d$, hence we get a 2-approximation.

Problem 3b. Let $x_1, \ldots, x_k$ be the center points chosen by the given algorithm. We claim that they are a valid super-sequence of the points that the algorithm from part a could have chosen (knowing d).

Indeed, inductively after choosing x_{i-1} the algorithm from part either ends, or chooses x_i such that $d(x_i, x_j) > d$ for all $1 \leqq j \leqq i - 1$. In particular, a point whose minimum distance to all previously chosen centers is largest will do.

From part 3a, we know that there are $l \leqq k$ clusters $H_1, \ldots, H_l$ and every point lies within distance $\leqq d$ of some $x_1, \ldots, x_l$. Hence if we obtain clusters $G_1, \ldots, G_k$ by assigning every vertex to the center closest to it, we have that if $y \in G_i$, then $d(y, x_i) \leqq d$. And therefore, by the triangular inequality (as in part 3a), the diamter of each G_i is at most $2d$. Hence we obtain a 2-approximation.

Problem 4a. Assume given any schedule A where jobs j and i are both feasible, j is scheduled before i and $d_j \geqq d_i$. Let $c_A(i)$ denote the completion time of job i under schedule A.

Consider schedule B produced from schedule A by removing job j and placing it right on top of job i. Observe, the completion times of job i and all jobs between i and j in schedule A only decrease (by p_j) in schedule B, also $c_B(j) = c_A(i)$, and no other jobs' completion times change.

Since by assumption $d_j \geqq c_A(i)$, in schedule B job j is still feasible but now it is ordered after i.

Repeat exhaustively until all feasible jobs are ordered in ascending order of their deadlines.

Problem 4b. Define $B(i, w)$ to be the fastest-completing feasible subset of $\{1, \dots, i\}$ such that the total weight of this subset is at least w. WLOG assume $i < j \Rightarrow d_i \leqq d_j$ which will imply (using part 4a) that for any schedule using jobs up to $i - 1$, we only need to try to incorporate job i on top of that schedule. Compute $B(i, w)$ dynamically via:

$$B(i,w) = \begin{cases} 0, & \text{if } w = 0 \\ \infty, & \text{if } i = 0 \\ \min\Big\{B(i-1,w), p_i + B(i-1, w-w_i)\Big\}, & \text{if } d_i \geqq p_i + B(i-1, w-w_i) \\ B(i-1,w), & \text{otherwise} \end{cases}$$

Runtime. The dynamic program computes a table of $B(i, w)$'s where i ranges from 0 to n and $w \leqq nw_{\max} = n\text{poly}(n)$. Total runtime is poly$(n)$.

Problem 4c. Guess optimum penalty OPT. Divide jobs into large and small, respectively, with $w_j > \text{OPT}$ and $w_j \leqq \text{OPT}$. Consider only large jobs. They have to be scheduled without any penalty. Try to schedule them. As shown in part 4a, a best schedule is simply to order them in ascending order of their deadlines. If this schedule is infeasible, your initial guess for OPT was bad.

Otherwise, "subtract" schedule for large jobs from initial problem: Pick a large scheduled job j and collapse time period $[d_j - p_j, d_j]$ to the single point $d_j - p_j$. I.e. for all $d_i \in [d_j - p_j, d_j]$ set $d_i = d_j - p_j$, and for all $d_i > d_j$ set $d_i = d_i - p_j$. Repeat until you run out of large jobs.

Call this the residual problem. The residual problem accommodates all schedules for the small jobs that don't violate the schedule for the large jobs in the original problem.

Find an approximate minimum penalty schedule for the small jobs in the residual problem by scaling. Set:

$$w_i' = \frac{n}{\epsilon \mathrm{OPT}} w_i \leqq n/\epsilon$$

New optimum is n/ϵ. Round weights up $w_i'' = \lceil w_i' \rceil$. Run DP on integer weights now. Guaranteed that DP schedule will have penalty at most $n/\epsilon + n = (1+\epsilon)n/\epsilon$, which is an ϵ-approximation of new optimum and hence correponds to an ϵ-approximation of the original optimum on the small weights.

When we insert back the large jobs we get an ϵ-approximation schedule for the original problem.

Finally, we need to guess OPT. First guess OPT $= 0$, if not, then do binary search on OPT, where $w_{\min} \leqq \mathrm{OPT} \leqq n w_{\max}$, and we only care to fall ϵ-factor within OPT.

Running times. Computing the large jobs involves sorting and checking so it takes $O(n \log n)$ assuming that we can compare deadlines in unit time. (But even without this assumption it is $O(n \log n \ \mathrm{poly}(n))$ which is still OK.) The computation of the residual problem takes $O(n^2)$. And the dynamic program takes $O(n^3/\epsilon)$ (because $w_{\max} = n/\epsilon$). So the dominant term so far is $O(n^3/\epsilon)$.

We have to perform a binary search on OPT, which searches over $m = n w_{\max}/(\epsilon \mathrm{OPT})$ values. Since if OPT $\neq 0$, OPT $\geqq w_{\min}$, we have $m \leqq (n/\epsilon)(w_{\max}/w_{\min})$. Since $w_{\max}$ and $w_{\min}$ have at most l bits (where l is the numbers of bits in the w with largest number of bits), $w_{\max}/w_{\min}$ has $O(l)$ bits. Hence, $\log m \leqq \log n + \log(1/\epsilon) + O(l)$.

Total runtime is: $O\big((n^3/\epsilon)(\log n + \log(1/\epsilon) + l)\big)$ (which is polynomial in the size $O(nl)$ of the input problem), so we have an FPAS.