6.854 Advanced Algorithms **Petar Maymounkov**

Problem Set 11 (November 23, 2005)

With: Benjamin Rossman, Oren Weimann, and Pouya Kheradpour

Problem 1. We reduce vertex cover to MAX-SAT with weights, such that the corresponding MAX-SAT instance has the same dependancy graph as the vertex cover problem (and hence the same tree-width). Then we apply the fact (proven in the previous problem set) that MAX-SAT with weigths runs in time $2^T O(n)$, where T is the tree width.

For each vertex v in the vertex set instance, introduce a variable x_v in the MAX-SAT instance. For each edge vw in vertex cover, introduce "an edge clause" $(x_v \vee x_w)$ in MAX-SAT with weight n. Lastly, for each vertex v, introduce "a vertex clause" $(\overline{x_v})$ with weight 1 in MAX-SAT. Observe that the dependancy graph of the MAX-SAT problem (where an edge corresponds to the relation "share a clause") is identical to the original vertex cover graph, and hence they have the same tree-width.

By construction, every vertex cover corresponds to an assignment that satisfies all edge clauses and some of the vertex clauses. We now prove that a MAX-SAT assignment corresponds to a smallest vertex cover.

First we show that a MAX-SAT assignment is a valid vertex cover. Assume not. Then there is at least one edge clause that is not satisfied, and hence the total satisfied weight is $\leqq mn$ (where m is the number of edges in the original graph). On the other hand, any valid vertex cover assignment that satisfies all edge clauses satisfies $> mn$ weight, because it must also satisfy at least one vertex clause. So we get a contradiction.

Finally, argue that the MAX-SAT assignment is indeed the minimal vertex cover. If it isn't, then the minimal vertex cover induces an assignment with higher satisfied weight (because all vertex clauses have the same weight), and hence we get a contradiction.

Problem 2a. We use the same potential function as in the linear case: $\Phi = kM + \sum_{i<j} d\big(s_i^{\mathrm{DC}}, s_j^{\mathrm{DC}}\big)$. Notice it is always non-negative since all terms are individually non-negative.

Much like in the linear case, when OPT moves distance d, it increases the current min-cost matching by at most d, and hence it increases the optimal (after the move) min-cost matching by at most d. The overall change in Φ is then $\leqq kd$, since $\sum_{\text{DC}}$ doesn't change.

Consider the case when DC moves. Subdivide it into phases, where within a phase the set of neighbors moving towards the request does not change. Let the number of neighbors within a phase be n.

WLOG one of the neighbors is matched with an OPT server at the request point (otherwise we could uncross the min-cost matching without increasing its cost). If the total distance traveled by DC in this phase is d, each neighbor individually travels d/n.

In the worst case (when there is only one OPT server between the moving neighbors) the current matching is (a) decreased by d/n due to the neighbor matched with the OPT server, and (b) increased by $(n-1)d/n$ by the remaining neighbors moving away from their matches. In total $\Delta M \leqq (n-2)d/n$.

Let's consider the change to $\sum_{\text{DC}}$. Let $b_i = |B_i|$, where B_i is the set of DC servers behind neighbor i. Then:

$$\Delta \sum_{j \in B_i} d\big(s_i^{\text{DC}}, s_j^{\text{DC}}\big) = \frac{d}{n}\left(b_i - \sum_{k \neq i} b_k\right)$$

Further, the distance change among all pairs $i \neq j$ of neighbors is:

$$\Delta \sum_{i \neq j} d\big(s_i^{\text{DC}}, s_j^{\text{DC}}\big) = -2\frac{d}{n}\binom{n}{2} = -\frac{d}{n}n(n-1)$$

Summing over all changes in distances among DC's servers, we get:

$$\Delta \sum_{i<j} \big(s_i^{\text{DC}}, s_j^{\text{DC}}\big) = -(n-2)(k-n)\frac{d}{n} - n(n-1)\frac{d}{n}$$

And combining this with ΔM, we get:

$$\Delta\Phi \leqq -d$$

In conclusion, every d move of OPT makes room for at most kd move of DC, hence DC is k-competitive.

Problem 2b. In the paging problem, recalling a new page into cache costs 1 unit. We model the paging algorithm by a star-shaped tree. With leaves corresponding to the pages, and k servers corresponding to the cache entries. Each edge in the tree has length 1/2.

Assume that the k-server problem starts off with all servers on the leaves (if not, the extra incurred cost is absorbed by the asymptotics). The star model establishes a minimum cost of 1 unit for a server to move from one page to another. (Of course, the k-server algorithm is allowed to be more wasteful, if it so desires.) This shows that the model is correct, because the minimum cost for page switch is the same in both problem statements.

We convert any given k-server algorithm into a paging algorithm as follows. If a server leaves a page, the corresponding paging algorithm does no evict that page until that server arrives at a different page. It is obvious that under this construction the paging algorithm incurs no more cost than the underlying k-server algorithm.

Problem 2c. We get Fiat's marking algorithm. To argue this, we need to disambiguate the case when the DC algorithm reaches a phase when multiple neighbors (of the request) reach the same tree vertex, because we need to specify which single one continues towards the request. It is clear that it doesn't matter which one does that, but for the sake of this problem, we will assume that a random one does.

The equivalence between Fiat and the DC algorithm is then given in the following simple terms. A cache entry is marked, if the corresponding server is at a leaf, and it is unmarked if it is at the center of the star.

This equivalence illuminates an interesting fact. In particular, it shows that the random choice of an unmarked page in Fiat's algorithm is unncessary. And that choosing an arbitrary unmakred page is sufficient to achieve k-competitiveness.

Problem 2d. We are going to model the weighted paging algorithm as a star as well. Where the length of the edge to page p will be equal to $w_p/2$, where w_p is the cost of recalling p into cache.

To prove equivalence, we have to show that switching pages incurs the same cost in both the paging and the server models. We make the following observation. In any sequence of events

each recall of a page is paired with an eventual eviction. We spread the cost of the recall equally over the recall and the eviction. This gives us direct correspondance to the server model, where the recall corresponds to traversing the edge towards the leaf, and the eviction corresponds to traversing the edge away from the leaf. In summary, a pair of recall/evict operations for a page p in the server model is forced to incur cost at least w_p.

We use the same interpretation of the k-server algorithm as a paging algorithm as the one described in part 2b.

Problem 3a. At step i, the online algorithm makes a mistake when a fraction $F_i > 1/2$ of the total weight w_i is in mistake. The updated total weight w_{i+1} is then:

$$\begin{aligned} w_{i+1} &= (1/2)Fw_i + (1-F)w_i \\ &= (1 - F/2)w_i \\ &\leqq (3/4)w_i \end{aligned}$$

Let x be the total number of mistakes, the online algorithm makes. Then after the completion of the experiment for the final total faculty weight w, we have that:

$$w \leqq (3/4)^x n$$

Problem 3b. On the other hand, if we examine the weight of the wisest faculty member, it begins at 1, and is shrinked by a factor of 2 exactly m times. Hence at the end of the algorithm it is equal to 2^{-m}.

Problem 3c. The weight of the wisest faculty member is a lower bound on the total final weight. Combining this with the upper bound from 3a, we get:

$$2^{-m} \leqq w \leqq (3/4)^x n$$

Solving the inequality for x gives:

$$\begin{aligned} x &\leqq \frac{1}{\log(4/3)}(m + \log n) \\ &\leqq 2.41(m + \log n) \end{aligned}$$

Problem 3d. At question i, regardless of whether the online algorithm guesses (probabilistically) the answer correctly, the total weight decreses as follows:

$$\begin{aligned} w_{i+1} &= F_i w_i \beta + (1 - F_i) w_i \\ &= \big(1 - F_i(1-\beta)\big) w_i \end{aligned}$$

Further, if X_i is the indicator random variable that the online algorithm errs on the i'th question, and $X = \sum_i X_i$, then $E[X] = \sum_i E[X_i] = \sum_i F_i$. And hence our goal will be to bound $\sum_i F_i$.

Problem 3e. Much like in the deterministic case, the total final weight w is lower-bounded by the weight of the wisest faculty β^m:

$$\beta^m \leqq w = n \prod_i \big(1 - F_i(1-\beta)\big)$$

Solve this inequality, using that $\ln(1-y) \leqq -y$ for $y \geqq 0$:

$$\begin{aligned} \beta^m &\leqq n \prod_i \big(1 - F_i(1-\beta)\big) \\ m \ln \beta &\leqq \ln n + \sum_i \ln\big(1 - F_i(1-\beta)\big) \\ &\leqq \ln n - (1-\beta) \sum_i F_i \\ \sum_i F_i &\leqq \frac{m \ln(1/\beta) + \ln n}{1-\beta} \qquad (\dagger) \end{aligned}$$

And since $E[X] = \sum_i F_i$, we get the needed result.

Problem 3f. Set:

$$\beta = \frac{1}{1 + \sqrt{\ln n / m}}$$

And directly substitute into (†) to get:

$$E[X] \leqq m + \ln n + 2\sqrt{m \ln n}$$

Problem 4a. Transform each axes-aligned rectangle in $\mathbb{R}^2$ defined by its diagonal corners (a, b) and (a', b') into the point (a, b, a', b') in $\mathbb{R}^4$. Also, transform each query rectange $(x, y), (x', y')$ in $\mathbb{R}^2$ into the query rectangle $(x, y, x, y), (x', y', x', y')$ in $\mathbb{R}^4$.

By definition $a < a'$ and $b < b'$. In $\mathbb{R}^2$, the rectangle $(a, b), (a', b')$ is contained in the query $(x, y), (x', y')$ iff $x \leqq a < a' \leqq x'$ and $y \leqq b < b' \leqq y'$ (Condition †).

In $\mathbb{R}^4$, the point (a, b, a', b') is contained in the query rectangle $(x, y, x, y), (x', y', x', y')$ iff $x \leqq a \leqq x'$, $x \leqq a' \leqq x'$, $y \leqq b \leqq y'$, and $y \leqq b' \leqq y'$ (Condition ‡).

Conditions (†) and (‡) are equivalent.

We prove the desired space and time requirements by induction. For space, inductive hypothesis is that d-dimensional structure requires $O(n \log^{d-1} n)$ space. Base case is trivial. A $d+1$-dimensional structure requires $O(n)$ space for the highest level BST as well some space for the d-dimensional sub-structures for each internal tree node of the highest level BST (by inductive hypothesis):

$$\sum_{i=1}^{\log n} 2^i \frac{n}{2^i} \log^{d-1} \frac{n}{2^i} \leqq n \log^d n$$

For time, again by induction. Hypothesis is that in d dimensions, identifying the last dimension's BST subtrees that contain the query points takes $O(\log^d n)$ time. Base step is trivial. In $d + 1$ dimensions, in $O(\log n)$ time we identify $O(\log n)$ subtrees of the highest level subtree, and for each of them, we need to recursively run the algorithm, so total time is $O(\log n \log^d n) = O(\log^{d+1} n)$. Once we've identified all subtrees ($O(\log^{d+1}$ in count) in the lowest dimension that contain the query answers, we enumerate the answers in another k steps in total.

Problem 4b. Observe that if a polygon is contained in an axes-aligned query rectange, so is its bounding axes-aligned query rectangle. Therefore, preprocess (in linear time) all polygons and replace them by their bounding rectangles. (Simply find the smallest x, largest x, smallest y and largest y coordinates for each polygon across all its vertices). And apply the algorithm from 4a to the bounding rectangles.