6.854 **Dan Ports**
Advanced Algorithms drkp@mit.edu
2004/11/10 Collaborators: {sarahl,pramook}

Problem Set 9

Problem 1:

Suppose we are dealing with a graph with n vertices, m edges, and treewidth k. We show that Min-Vertex-Cover is fixed-parameter tractable with respect to k. We use the standard technique of successively selecting a vertex in accordance with the minimum elimination ordering, removing it, and replacing it with edges between each of its neighbors.

For each edge $e(v, w)$, we maintain a "truth table" indicating the assignments (1 of included or 0 if excluded in the vertex cover) of v and w that cover the edge, and their costs (number of vertices required to cover, including eliminated vertices). For edges in the original graph, this is simply the following (though as we eliminate vertices it will become more complex).

v	w	e Covered?	Cost
0	0	$\times$	
0	1	$\surd$	1
1	0	$\surd$	1
1	1	$\surd$	2

At each stage, we take the vertex a that is next in the minimum elimination ordering, and consider all of its neighbors $\{b_1, b_2, \ldots, b_\beta\}$. We build a table over all possible $\{0,1\}$ values of the b_i. This table has 2^β entries, which is less than 2^k since a can have at most k neighbors because k is the treewidth and we are proceeding along the optimal elimination ordering. For each entry in the table, we take the assignments of the b_i and consider each edge $e_i = (a, b_i)$. We examine the truth tables for each e_i and determine which assignment of $a = \{0,1\}$ will cover every edge e_i with the minimum total cost. Assuming such an assignment of a exists, we place the its cost in the entry of the truth table corresponding to this assignment of the b_i, and tag it with the required value of a (and that of any previously eliminated vertices); if no such covering assignment of a exists for these values of the b_i, we note this in the table instead.

We then remove vertex a from the graph, and for each pair $\langle b_i,\ b_j\rangle$ $(0 < i < j \leq \beta)$, we create the edge $e_{ij} = (b_i, b_j)$. To generate the truth table for e_{ij}, we consider the four possible assignments $\langle b_i,\ b_j\rangle = \{0,1\} \times \{0,1\}$. For each such assignment of b_i and b_j, we examine each entry in the truth table generated above that has these values of b_i and b_j; if there exists at least one such entry with a covering assignment, we mark this assignment of b_i and b_j in the e_{ij} truth table as a covering assignment with cost equal to the minimum of the matching entries in the large truth table.

Note that this process eliminates the next vertex in the elimination ordering, and connects all the neighboring vertices with newly created edges, so it is a valid elimination step. This builds a large truth table of size at most 2^k, as shown above; producing each entry in the truth table requires time polynomial in n (independent of k), and constructing the edges requires accessing each entry in the truth table once, so the total time required for each vertex is time $2^k n^{O(1)}$.

We repeat this process of elimination along the elimination ordering until we have reduced the problem to two vertices connected by a single edge. We then examine the truth table for that final edge and select the covering assignment with minimum cost; this gives a minimum-cost vertex covering. The cost of the vertex covering is the cost of the selected assignment; to read out the actual vertices involved in the covering, we follow the associated set of tags indicating which assignments for the eliminated vertices were required to achieve this minimum cost. At most n

elimination steps are required, and the time for each is bounded as shown above, so we have shown that MIN-VERTEX-COVER is fixed-parameter tractable with respect to the treewidth of the graph.

Problem 2:

a) Suppose DET is a deterministic algorithm for dating. Then it has some deterministic sequence of people it will choose to date. Fate as an adversary can assign ranks based on this sequence so that DET's input sequence will consist of partners in order of increasing rank (increasing badness). Since the only information DET has at any time is the relative ordering of the ranks of the previous partners, all such input sequences appear identical to it. Since it is deterministic, it must have a fixed response to this sequence: it must terminate and marry the $\varkappa$th partner it sees, for some $\varkappa \leq k$. But then we can feed it an input sequence consisting of partners $k - \varkappa + 1$ to k in that order. Then it will choose the $\varkappa$th partner, which is the kth, i.e. the worst possible choice.

b) Our randomized algorithm is as follows: we choose $\frac{k}{2}$ people in random order to date and then break up with. We then repeatedly date people selected randomly and marry them if they rank highest of all the people previously dated. Since the selections are made randomly, the adversary cannot control the input sequence of ranks; the ordering of ranks is uniformly distributed over the set of all permutations. Consider the probability of ending up with the best companion (call him/her/it x_1, and x_2 the next best, etc.). Clearly this requires that x_1 be in the second $\frac{k}{2}$ choices; this happens with probability $\frac{1}{2}$. It will definitely occur if the x_2 is also in the first half, since then x_1 is the only person that ranks higher. This also occurs with probability $\frac{1}{2}$, and since the probabilities are independent, the combined case occurs $\frac{1}{4}$ of the time. (This condition is sufficient but not necessary; it is also possible, for example, for x_3 to be in the first half and x_2 to come after x_1). So this algorithm will choose the best person with probability at least $\frac{1}{4}$.

c) To achieve an expected rank of $O(\log k)$, we use the following algorithm: as before, choose $\frac{k}{2}$ people in random order to date and break up with. Then, repeatedly date randomly-selected people until finding one who ranks preferable to all people previously dated except at most $\log k$.

I am reasonably convinced that this algorithm achieves the correct expected result, but I do not have nearly enough time to write a proof of correctness, particularly if I want to have enough time to ever go on a date.

d) We see that successful dating requires a randomized algorithm. For those of us without access to a perfect random number generator, this suggests that the dating problem is hopeless. Empirical evidence supports this conclusion.

Problem 3:

a) Suppose the online algorithm makes a mistake. This means that the (yes/no) answer of maximum weight is incorrect; i.e. at least $1/2$ of the total weight was associated with the wrong answer. So this weight will be halved, and $1/4$ of the total weight is removed. Thus, the total weight decreases by a factor of $\frac{1}{1-1/4} = 4/3$.

Suppose the wisest faculty member makes m mistakes. Then the algorithm cannot make less than m mistakes, because this would mean that there is at least one faculty member that makes less than m mistakes, contradicting the optimality of the wisest faculty member. Since the total weight begins as n, and decreases by a factor of $4/3$ after each mistake, the weight after m mistakes is $n\,(4/3)^m$.

b) The wisest faculty member makes m mistakes. The wisest faculty member began with weight 1, and the weight was halved for each of the m mistakes: the weight is $(1/2)^m$. The wisest faculty member has this weight, so the total weight of the faculty is at least this.

c) Let W be the total weight of the faculty, and x be the number of mistakes made by the algorithm. We show that $x \leq 2.41(m + \lg n)$.

Since the algorithm makes at least as many mistakes at the wisest faculty member, the upper bound shown above indicates that

$$W \leq n\,(3/4)^x$$
$$\frac{W}{n} \leq (3/4)^x$$

Since the weight is bounded below by $(1/2)^m$

$$\frac{(1/2)^m}{n} \leq (3/4)^x$$
$$\lg \frac{(1/2)^m}{n} \leq \lg\,(3/4)^x$$
$$m \lg \frac{1}{2} - \lg n \leq x \lg \frac{3}{4}$$
$$x \leq \frac{m \lg 1/2 - \lg n}{\lg 3/4}$$
$$x \leq \frac{-m - \lg n}{\lg 3/4}$$
$$x \leq \frac{m + \lg n}{\lg 4/3} \approx 2.41(m + \lg n)$$

d) After each question, if F is the fraction of the weight of the faculty with the wrong answer, and W is the total weight, the new weight W' is $\beta FW + (1 - F)W = W - WF(1 - \beta)$. So the multiplicative change in weight is

$$\frac{W'}{W} = 1 - F(1 - \beta)$$

e) Note that as before, the total weight of the faculty is bounded below by the weight of the wisest faculty member, which is β^m.

Let X be the random variable representing the total number of errors made using this algorithm, and define $X = \sum_i X_i$, where X_i is the indicator variable indicating that the algorithm made a mistake on the ith question. Then $\mathrm{E}[X] = \sum_i \mathrm{E}[X_i] = \sum_i F_i$ since the probability that a mistake is made on question i is the fraction of the weight of the faculty with the wrong answer.

The total weight of the faculty after k questions will be

$$W_k = n \prod_{i=1}^{k} (1 - (1 - \beta)F_i) \geq \beta^m$$

applying the lower bound above. Note that since $(1 - \beta)F_i$ is less than 1,

$$e^{-(1-\beta)F_i} = 1 - (1 - \beta)F_i + \frac{((1 - \beta)F_i)^2}{2!} + \ldots \geq 1 - (1 - \beta)F_i$$

since the magnitude of the terms is strictly decreasing. Applying this to the upper bound,

$$\begin{aligned}
n\prod_{i=1}^{k} e^{-(1-\beta)F_i} &\geq \beta^m \\
e^{-(1-\beta)\sum_{i=1}^{k} F_i} &\geq \frac{\beta^m}{n} \\
-(1-\beta)\sum_{i=1}^{k} F_i &\geq m\ln\beta - \ln n \\
\sum_{i=1}^{k} F_i &\leq \frac{-m\ln\beta + \ln n}{1-\beta} \\
\mathrm{E}\left[X_k\right] = \sum_{i=1}^{k} F_i &\leq \frac{m\ln\frac{1}{\beta} + \ln n}{1-\beta}
\end{aligned}$$

which is the desired bound.

f) Let

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

Note that

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

and

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

Then the expected number of errors is

$$\begin{aligned}
\mathrm{E}\left[X_k\right] &\leq \frac{m\ln\frac{1}{\beta} + \ln n}{1-\beta} \\
&= \left(m\ln\left(\sqrt{\frac{m}{\ln n}}+1\right) + \ln n\right)\left(1+\sqrt{\frac{\ln n}{m}}\right)
\end{aligned}$$

Since $x \geq \ln(1+x)$ (the proof is by a standard monotonicity argument),

$$\begin{aligned}
&\leq \left(m\sqrt{\frac{\ln n}{m}} + \ln n\right)\left(1+\sqrt{\frac{m}{\ln n}}\right) \\
&= \left(\sqrt{m\ln n} + \ln n\right)\left(1+\sqrt{\frac{m}{\ln n}}\right) \\
&= \sqrt{m\ln n} + \ln n + m + \sqrt{m\ln n} \\
&= m + \ln n + 2\sqrt{m\ln n} = m + \ln n + O\left(\sqrt{m\ln n}\right)
\end{aligned}$$

which is the desired bound.

Problem 4:

a) Define the potential function $\Phi = kM_{\min} = \sum_{DC}$. As with the DC-Line algorithm, we show that the potential function increases by at most kd when Opt moves by d, and decreases by at most d when DC-Tree moves by distance d.

Suppose Opt moves a server by distance d. $M_{\min}$ increases by at most d since this server may be moving away from its matched server. $S = \sum_{DC}$ does not change. So the potential function increases by at most kd.

Consider a phase in which the request has x neighbors. Then there are x servers moving, and $k-x$ stationary. Suppose the DC-Tree algorithm moves these x servers by distance d. $M_{\min}$ increases by $(x-2)d$: it decreases by d because one server is moving closer to the server it is matched with at the destination, and increases by at most $(x-1)d$ because the other moving servers may be moving away from their matched servers.

S decreases by $\frac{x(x-1)}{2}2d$ because the x servers that are moving move closer to each other by $2d$. It also decreases by $(k-x)(x-2)d$ since each stationary server has one server moving away from it and the others moving towards it (if there were a second server moving away from it, it could not be a neighbor of the request). So the total change in S is an decrease of $d\,(x(x-1)+(k-x)(x-2))$.

Thus, the total change of Φ during this phase is

$$d\,(k(x-2) - x(x-1) - (k-x)(x-2)) = -dx$$

Since there are x servers moving a distance d during this phase, this means the potential function decreases by the total cost of the solution by DC-Tree.

Since the potential function Φ is always positive, the decrease in Φ cannot be larger than the increase in Φ. But the increase in Φ is less than k times the cost of Opt, and the decrease in Φ is the cost of DC-Tree. So DC-Tree is k-competitive.

b) We can represent the paging problem with n pages and a cache of size k as a case of the k-server on a tree problem. We introduce a tree consisting of a root node connected by edges of equal length to n nodes, one per page. We place k servers on it, where k is the size of the cache. Each request for a page corresponds to a request on the tree, and moving a server from one leaf node to another corresponds to evicting the old page from memory and loading the requested page.

Since the k-server algorithm can place servers midway along edges, which makes no sense for paging, we change the interpretation so that a page is in the cache whenever a server is on the edge that connects the root to that page's node. This is still reasonable for paging; there is never more than one server on the same edge because as soon as one server moves onto the edge, it will block any other server from being the neighboring server to a request for that page.

c) The DC-Tree algorithm, applied to this reduction, is equivalent to the paging algorithm of filling the cache while empty pages remain, then evicts all pages from the cache once a page fault occurs and no empty pages remain. To see this, note that the algorithm begins with servers placed at the nodes for each of the pages in the cache. When a new request arrives, all pages are moved to the root node, and they all arrive simultaneously; this corresponds to emptying the cache. One of them, selected arbitrarily, moves to service the request, i.e. loading that page into the cache. The remaining $k-1$ servers will be available for future requests until the cache fills up; they will then move back to the center.

Problem 5:

About 15 hours, much of it spent staring blankly at problem 2 before realizing that I was misinterpreting the problem.