CSC373S Lecture 13

- Recall that as a final randomized DC application (very related to Quicksort) we are considering an approximation algorithm for what is called the (weighted) *feedback arc set problem on tournaments* (FAS-Tournaments) and its application to aggregate (aka universal) ranking and aggregate clustering.

  A weighted tournament is a complete directed graph $G = (V.E)$ with a weight function $w(e) = w_{ij}$. Here complete means that for every $i \neq j$, the directed edge (i, j) exists. We will only be interested in weighted tournaments where $w_{ij} + w_{ji} = 1$ in which case we can think of the weights as probabilities. An unweighted tournament is such a weighted tournament where all edge weights are in $\{0, 1\}$; that is, we can think of an unweighted tournament as one where for every $i \neq j$, exactly one of (i, j) or (j, i) is in E. The weighted feedback arc set problem in a weighted graph is the problem of choosing a total ordering $<_\pi$ on the vertices (i.e. choosing a permutation π of the vertices) so as to minimize $\sum_{(i,j):i<_\pi j} w_{ji}$. That is, we are trying to minimize the combined weight of all edges that are inconsistent with the total ordering given. In the unweighted case this is simply $|(i, j) : (j, i) \in E|$.

- Weighted tournaments give rise to unweighted (majority) tournaments by defining a graph (V, E_w) where (i, j) in E_w if $w(i, j) > w(j, i)$. If $w_{ij} = w_{ji}$ then arbitrarily put either (i, j) or (j, i) in E_w.

  Now what does all this have to do with randomized DC algorithms? Ailon, Charikar, and Newman solve the unweighted FAS on tournaments by an algorithm resembling quicksort. Their algorithm is a 3-approximation algorithm (and is the first constant time algorithm for this problem). Then the weighted problem is solved by reducing to the unweighted majority tournament. The randomized DC algorithm (called FAS-PIVOT) for the unweighted case is as follows:

  FAS-PIVOT(G)
  $V_L := \emptyset; V_R := \emptyset$
  Pick random pivot $i \in V$
  For all vertices $j \in V - \{i\}$
  If $(j, i) \in E$ then $V_L := V_L \cup \{j\}$
  Else $V_R := V_R \cup \{j\}$
  Let $G_L = (V_L, E_L)$ be tournament induced by V_l
  Let $G_R = (V_L, E_L)$ be tournament induced by V_R
  Return [FAS-PIVOT(G_L), i,FAS-PIVOT(G_R)]

  Theorem: FAS-PIVOT provides an expected 3-approximation; that is, $\mathcal{E}[cost(\text{FAS-PIVOT}]] \leq 3 \cdot cost(\text{OPT})$.
  Note: To avoid confusion with E as the set of graph edges, I am using $\mathcal{E}$ for expectation.

  Furthermore, when applied to the majority tournament induced by a weight function which satisfies the triangle inequality (as well as $w_{ij} + w_{ji} = 1$) this becomes (in

expectation) a 2-approximation. Therefore, given a set of rankings and the Kemeny cost criteria, let PIVOT be the algorithm for the ranking produced by the FAS-PIVOT on the majority tournament and cost(OPT) denote the optimal Kemeny meta ranking. Then $\mathcal{E}[cost(\text{FAS-PIVOT})] \leq 2 \cdot \text{cost(OPT)}$

Now this sounds good except that it turns out that a "randomized" or brute force search also works this well; that is, randomly choose one of the k given rankings as the aggregate ranking (or take the best of the k rankings). This also provides a 2-approximation. But then one can show that taking the best of the two random methods yields $\mathcal{E}[\text{cost(Best)}] \leq (11/7) \cdot cost(\text{OPT})$.

- Once the expected value of a solution is good, we can insure that with some arbitrarily good probability that we will be at least close to that expected value. Namely, we can use Markov's inequality:

  For a non negative random variable X with $\mathcal{E}[X] = \mu$, $Prob[X \geq (1+\epsilon)\mathcal{E}[X]] \leq (\frac{1}{1+\epsilon})$

  Now suppose we run the same randomized algorithm k times (using new random bits) viewed as k independent trials of the random variable $X = \text{cost[ALG]}$. Then the probability that each trial will have cost $\geq (1+\epsilon)\mathcal{E}[X]$ is at most $(\frac{1}{1+\epsilon})^k$. For example, suppose $\epsilon = 1/10$ so that $(\frac{1}{1+\epsilon})^k \leq (\frac{10}{11})^k$. When say $k = 10$, this probability is $\approx \frac{10}{26}$.

  NOTE: This is one of the most important aspects of randomized algorithms. If there is some (say) constant probability of something good happening in execution of an algorithm, then we can execute the algorithm many times to improve the probability to any desired level (at the cost of more time and more random bits).

- Since we have begun to consider randomized algorithms, let us continue this discussion for another week. Here we have been using randomization for solving function computation (eg median) and optimization problems (eg FAS-Tournaments) which do not have any probabilistic aspect (as part of the problem definition). It is not known theoretically speaking if randomization changes the concept of what can and can't be computed in polynomial time. But even when randomization is not needed, it is often simper to think in terms of randomized algorithms and/or can improve upon the best known deterministic algorithm.

  NOTE: There are problem domains (e.g. simulation of a stochastic process, cryptography, sub linear time and space algorithms) where randomization is necessary for correctness (e.g. security in cryptography) or getting any reasonable estimate of the correct answer (as in sublinear algorithms).

  As already noted, randomization can be used with any algorithmic technique, as well as in more naive randomization (eg simple sampling) where the probability of a "successful trial" is sufficiently high.

- As our next example, lets consider two similar algebraic problems:
  1) Given $n \times n$ matrices A, B, C, determine if $C = A * B$. It could be that you have developed a very complicated matrix multiplication algorithm that beats the currently best algorithm (say having time complexity $O(n^{2.3})$. The algorithm is so complicated to analyze its correctness that you decide that whenever you use the algorithm you had better check the output. You could, of course, use a classical or other known method which will then dominate the complexity and take away any reason to use the complicated algorithm? So what should you do?

  2) We are given an $n \times n$ matrix A whose entries a_{ij} are say linear polynomials (over m variables $X = \{x_1, \ldots, x_m\}$. The determinant of $A = det(A) = \sum_{\sigma \in S_n} (-1)^{sgn\sigma} \Pi_i a_{i,\sigma(i)}$ is then a degree n multi-variate polynomial over the variables in X. We want to determine if $det(A) \equiv \bar{0}$ where $\bar{0}$ is the identically zero polynomial. How can we check whether or not $det(A) \equiv \bar{0}$. We could use an algorithm for computing determinants but even ignoring the computation steps for adding and multiplying polynomials, $det(A)$ generally has $n!$ terms so that just writing out $det(A)$ could be computationally infeasible.

- Checking if $C = A * B$.

  Let $\vec{x} = (x_1, \ldots, x_n)$ be a vector. If $C = A*B$, then $A*(B*\vec{x}) = (A*B)*\vec{x} = C*\vec{x}$. A Matrix vector product can be computed in $O(n^2)$ operations. So to check if $C = A * B$ we can simply choose random vectors $\vec{x}$ and test if $A * (B * \vec{x}) = C * \vec{x}$. Equivalently we are testing if $(C - A * B)\vec{x} = \vec{0}$ where $\vec{0}$ is the zero vector. Each such trial costs $O(n^2)$. So we are left with the question as to what if the probability that we will detect $C \neq A * B$ when using this randomized test. If we have some reasonable probability of a trial detecting $C \neq A * B$ then we can run enough trials to obtain high probability that we will detect $C \neq A * B$. Of course, if $C = A * B$ then we will never have $A * (B * \vec{x}) \neq C\vec{x}$.

- Checking if $det(A) \equiv \bar{0}$. We basically use the same rather naive idea of checking for the desired result by choosing random values for the variables in X and then running a determinant algorithm on the resulting scalar matrix. Again, if we have a reasonable probability that a random trial will detect $det(A) \neq \bar{0}$ (if $det(A)$ is not the zero polynomial), we can raise the probability by running sufficiently many trials.

- In the next lecture we will prove that we obtain the necessary probabilty bounds in these two applications of polynomial identity testing.