CPS 430/590.06 **Prof. John Reif**
Design and Analysis of Algorithms **Fall 2013**
Homework 6 **Matt Dickenson**

Question 1

Given a vector of probabilities p of length n, we wish to compute the probability of k successes in n trials, with probability of success i in the i^{th} trial. Obviously if $k > n$, the probability is zero.

One (inefficient) way to go about this is to iterate over all of the possible subsets S of n that could offer k successes and $n - k$ failures. For each of those subsets $s \in S$, we can multiply the probabilities of the successes (s_a), $\prod_{i \in s_a} p_i$, and the complements of the probabilities for the failures (s_b), $\prod_{j \in s_b}(1 - p_j)$. This gives us the formula

$$Pr(k \text{ successes}) = \sum_{s \in S} \prod_{i \in s_a} p_i \prod_{j \in s_b} (1 - p_j)$$

However, there are $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ elements in S, which is quite large and makes this algorithm inefficient.

A more efficient approach uses recursion, alternating between addition and subtraction to avoid double-counting. Again, if $k > n$, we return zero. The base case of the recursion is $k = 0$, in which case we return the product of all failure probabilities, $\prod_j = 1^n(1 - p_j$. For $0 < k <= n$,

$$Pr(k \text{ successes}) = \frac{1}{k} \sum_{i=1}^{k} \sum_{j=1}^{n} (\frac{p_j}{1 - p_j})^i Pr(k - i \text{ successes})(-1)^{i-1}$$

Notice that the inner term $\sum_{j=1}^{n}(\frac{p_j}{1-p_j}) = C$ is constant for a given problem, so we can compute it once at the beginning:

$$Pr(k \text{ successes}) = \frac{1}{k} \sum_{i=1}^{k} C^i Pr(k - i \text{ successes})(-1)^{i-1}$$

We are then left with a sum of size k and $k + 1$ recursive calls. Using n as an upper bound on k, this is $O(n \times n + n) = O(n^2)$, as desired.

Question 2

1. Assume $P = NP$.
2. The factorization of large numbers, $F(n)$, can be verified in polynomial time, that is $F(n) \in NP$. $P = NP$ implies $F(n) \in P$.
3. Suppose an adversary initially has the public key function $P(M) = M^e mod n$ for the message M, but not the secret key $S(C) = C^d mod n$ or the value of n. $S(P(M)) = M = (P(S(M))$.

4. If $F(n) \in P$ then n can be factorized in polynomial time. Because e and d are small primes, the adversary can find $S(C)$ by trial and error (iteration) in polynomial time.

5. Because $S(C)$ can be computed in polynomial time, the adversary can use the encrypted transmission $P(M)$ to compute $M = S(P(M))$.

6. Thus, if $P = NP$, an adversary can break RSA encryption.

Question 3

Recall from class that $CLIQUE$ is NP-complete. For $KITE$, we wish to find a clique of size g connected to a tail, also of size g.

If we are given a graph $G = \{V, E\}$ and asked to perform $CLIQUE$, we could reduce this problem to $KITE$. We could do this by adding tails of size g to all $v \in V$. Then, the kite we find would consist of a clique plus a tail.

Finding a polynomial-time solution to $KITE$ would imply a polynomial-time solution to $CLIQUE$.Because $CLIQUE$ is NP-complete and can be reduced to $KITE$, $KITE$ must also be NP-complete.

Question 4

A When $k = 2$, there are exactly two terminal nodes. Notice that if there is more than one edge in E containing s_i, then s_i is not a terminal node. Thus, any cut of the graph is also a multiway cut for $k = 2$. We can reduce minimum cut to maximum flow by seeking the maximum flow from s_1 and s_2. Maximum flow has an exact, polynomial solution when $k = 2$, so we can exactly solve the multiway cut problem in polynomial time when $k = 2$.

B Let C^* denote the optimal solution to the multiway cut problem. For each $s_1, ..., s_3$ there is a subset of C^* that contains the cut removing s_i from the rest of G; denote this C_i^*. Notice that $C^* = C_1^* \cup C_2^* \cup C_3^*$.

Next, notice that the weight of C_i^* must be less than or equal to the weight of all of the edges containing s_i, because the set of all edges containing s_i is also a cut.

Each edge is connected to exactly two components, at most two of which can be in $\{s_1, s_2, s_3\}$. Thus, if we remove all of the edges connected to each of the terminal nodes, this will be at most twice the weight of the optimal solution C^*. This gives us a two-approximation for the multiway cut problem when $k = 3$.

C We can use a greedy algorithm for multiway cut.

1. Find the minimum cut that splits G into two components by running minimum flow from each $s_i \in V$ to each $s_{-i} \in V$. Denote the two resulting components by G_1', G_2'.

2. Find the minimum cut dividing G_1' or G_2', yielding G_3' (and a modified G_1' or G_2').

3. Continue this process until all k terminal nodes are in their own component.

Notice that this process is similar to the approximation in (B) above, in which we treat the subgraph at each step of the greedy search as its own terminal (temporarily). Similarly, this is a two-approximation of the optimal multiway cut.