ECS 222A: Assignment #6

Due on Thursday,March 5, 2015

Daniel Gusfield TR 4:40pm-6:00pm

Wenhao Wu

Contents

Problem 1

We saw a randomized algorithm that tries to find a global Min cut in an undirected, unweighted graph G with m edges. Now suppose we want to find a cut that has a *large* number of edges, i.e., a partition of the nodes of G into two sets S and T so that the number of edges that have one node in S and one node in T is large. Denote the maximum possible number as $Max(G)$. The problem of finding $Max(G)$ is NP-hard, so we would like a randomized algorithm that finds an S, T cut where the *expected* number of edges that cross the cut is a large fraction of $Max(G)$.
Here is a particularly brainless algorithm that does it. For each vertex, flip a coin: if the coin comes up heads, put the vertex in S, otherwise, put it in T. Assume that the coin is fair, i.e., the probability of heads is 1/2.

Problem 1(a)

Using this randomized algorithm, what is the expected number of edges that have one node in S and one in T? Explain.

Answer:

Lemma 1.1 *Denote the expected number of edges (u, v) from vertex u such that $u \in S$ AND $v \in T$ or $u \in T$ AND $v \in S$ as $E_{cross}(u)$. Then we have*

$$E_{cross}(u) = \frac{d(u)}{2} \tag{1}$$

in which $d(u)$ is the degree of vertex u.

Proof Firstly we have $P\{u \in S\} = P\{u \in T\} = 1/2$. The probability that there among the $d(u)$ neighbors of vertex u, k of them belong to S while $d(u) - k$ of belong to T is

$$\begin{aligned} P_u(|S| = k) &= \left(\begin{array}{c} k \\ d(u) \end{array} \right) \left(\frac{1}{2}\right)^k \left(\frac{1}{2}\right)^{d(u)-k} \\ &= \left(\begin{array}{c} k \\ d(u) \end{array} \right) \left(\frac{1}{2}\right)^{d(u)} \end{aligned} \tag{2}$$

Consequently,

$$\begin{aligned} E_{cross}(u) &= \sum_{k=0}^{d(u)} P\{u \in S\} P_u(|S| = k)(d(u) - k) + \sum_{k=0}^{d(u)} P\{u \in T\} P_u(|S| = k)k \\ &= 2 \cdot \frac{1}{2} \sum_{k=0}^{d(u)} k \left(\begin{array}{c} k \\ d(u) \end{array} \right) \left(\frac{1}{2}\right)^{d(u)} \\ &= \frac{d(u)}{2} \end{aligned} \tag{3}$$

Consequently, the expected number of eges that have one node in S and one in T, according to Lemma 1.1 and the handshake lemma, is

$$\begin{aligned} E_{cross} &= \frac{1}{2} \sum_{u \in V} E_{cross}(u) \\ &= \frac{|E|}{2} \end{aligned} \tag{4}$$

where $|E|$ is the number of edges in graph G.

Problem 1 continued on next page...

Problem 1(b)

Prove that in any undirected graph G, there is a cut that contains at least half of the edges in G.

Answer: We note there are a total number of $2^{|V|-1}$ possible cuts (taking into consideration of the symmetry between S and T), and the random algorithm results in any of them with probability $(1/2)^{|V|-1}$. Denote the number of edges contained in cut C as $n_{cross}(C)$. Assume for contradiction that $n_{cross}(C) < |E|/2$, then we have

$$\begin{aligned} E_{cross} &= \sum_C \left(\frac{1}{2}\right)^{|V|-1} n_{cross}(C) \\ &< \sum_C \left(\frac{1}{2}\right)^{|V|-1} \frac{|E|}{2} \\ &= \frac{|E|}{2} \end{aligned} \tag{5}$$

which is in contradiction with the solution to Problem 1(a). Consequently, there is a cut that contains at least half of the edges in G.

Problem 1(c)

Often in the analysis of social media, people build graphs representing who knows or likes (or hates or dates) whomever else. Then they analyze the graphs for particular features, such as large cliques, large independent sets, small cuts, nodes with high degree, the number of nodes of degree 1, number of triangles, etc. and they ascribe a "meaning" to each of these features. For example, a node with high degree is a "hub" or "kingpin"; and a large clique represents a "socially cohesive unit"; and a small cut is a "bottleneck", etc. This same approach to studying interactions is used in a huge variety of other systems. For example, graphs (call them "biological networks") are also to represent interactions between molecules, or between animals in a biological system, and biological, behavioural or chemical meanings are ascribed to features in these graphs. In general, such networks are called "interaction networks".
Now consider a "large cut" as a feature of an interaction network. Make up (use your imagination) as many meanings you can for a large cut in a social or biological network, or any other interaction network you can describe.
Given the fact stated in problem 1b, how large must a cut be before it could plausibly say anything meaningful about the interactions represented in an interaction graph? If you were a "network analyst" trying to find meaningful information from an interaction network, and you didn't know the fact stated in problem 1b, would that be a problem?

Answer: As an example of a large cut, consider a conflict graph in which each node represents "a student" and each edge represents "hatred" between two students. The teacher is having a big headache since the students in his class hates each other so much, that he decides to divide the class into 2 so that the total conflicts is minimized in each subclass. To do so he will need to find a large cut, so that a large number of conflict across the cut can be removed with this division.
A cut must be greater than $|E|/2$ to be meaningful, since according to Problem 1b, there is going to be a cut of size of at least $|E|/2$ anyway in any graph G.

Problem 2

Extend the analysis done for 3-SAT (in class and also in Section 13.4) to 4-SAT, i.e., the assumption that every clause has 4 literals. Then generalize to t-SAT for any fixed integer t. That is, generalize statments 13.14, 13.15, and 13.16, and justify your answers.

Problem 2 continued on next page...

Answer:

Lemma 2.1 *(Generalization of statement 13.14) Consider a t-SAT formula, where each clause has t different variables. The expected number of clauses satisfied by a random assignment is within an approximation factor* $1-(1/2)^t$ *of optimal.*

Proof Denote $Z_i = 1$ if the i-th clause is true and $Z_i = 0$ otherwise. The expectation of the random variable Z_i is

$$\begin{aligned} E[Z_i] &= P\{Z_i = 1\} \cdot 1 + P\{Z_i = 0\} \cdot 0 \\ &= P\{Z_i = 1\} \\ &= 1 - P\{\text{All } t \text{ literals in the } i\text{-th clauses are false}\} \\ &= 1 - \left(\frac{1}{2}\right)^t \end{aligned} \tag{6}$$

Consequently, the expected number of clauses satisfied by the random assignment is

$$\begin{aligned} E[Z] &= \sum_{i=1}^{k} E[Z_i] \\ &= \left[1 - \left(\frac{1}{2}\right)^t\right] k \end{aligned} \tag{7}$$

where k is the total number of clauses.

Lemma 2.2 *(Generalization of statement 13.15) For every instance of t-SAT, there is a truth assignment that satisfies at least a* $1-(1/2)^t$ *fraction of all clauses.*

Proof Denote the probability that the random algorithm results in assignment A_l as $p(A_l)$ and the number of clauses satisfied by A_l as $Z(A_l)$. Assume for contradiction that $Z(A_l) < (1-(1/2)^t)k$ for all assignment A_l. Then we have

$$\begin{aligned} E[Z] &= \sum_{l} p(A_l) Z(A_l) \\ &< \sum_{l} p(A_l) \left[1 - \left(\frac{1}{2}\right)^t\right] k \\ &= \left[1 - \left(\frac{1}{2}\right)^t\right] k \end{aligned} \tag{8}$$

which is in contradiction with Lemma 2.1. Consequently, there is a truth assignment that satisfies at least a $1-(1/2)^t$ fraction of all clauses.

Lemma 2.3 *(Generalization of statement 13.16) There is a randomized algorithm with polynomial expected running time that is guaranteed to produce a truth assignment satisfying at least a* $1-(1/2)^t$ *fraction of all clauses.*

Proof We propose the random algorithm that generate random assignment (with 1/2 probability of truth) independently and continuously until a truth assignment satisfying at least a $(1-(1/2)^t)$ is generated. Next we show that this algorithm has polynomial expected running time.
Denote p_j as the probability that a single round of random truth assignment results in exactly j clauses being satisfied. From Lemma 2.1, we have

$$\sum_{j=1}^{k} j p_j = \left[1 - \left(\frac{1}{2}\right)^t\right] k \tag{9}$$

Problem 2 continued on next page...

Denote the probability that a single round of random truth assignment results in at least $1-(1/2)^t$ fraction of all clauses as p:

$$p = \sum_{j \geq [1-(1/2)^t]k} p_j \tag{10}$$

Define $k' = \lceil (1-(1/2)^t)k \rceil - 1$, then from Eq. (9) and Eq. (10) we have

$$\begin{aligned} \left[1 - \left(\frac{1}{2}\right)^t\right] k &\leq \sum_{j < [1-(1/2)^t]k} k' p_j + \sum_{j \geq [1-(1/2)^t]k} k p_j \\ &= k'(1-p) + kp \\ &\leq k' + kp \end{aligned} \tag{11}$$

Therefore

$$\begin{aligned} p &\geq \left[1 - \left(\frac{1}{2}\right)^t\right] - \frac{k'}{k} \\ &\geq \frac{1}{k2^t} \end{aligned} \tag{12}$$

since both k and k' are integers. Consequently, according to Bernoulli distribution, the expected running time of this random algorithm is

$$E_{run} = \frac{1}{p} = k2^t \tag{13}$$

which is linear w.r.t the number of clauses.

Problem 3

Do problem 7 in chapter 13 of the book. How does the answer to this problem relate to the answer of problem 2?
In Section 13.4, we designed an approximation algorithm to within a factor of 7/8 for the MAX 3-SAT Problem, where we assumed that each clause has terms associated with three different variables. In this problem, we will consider the analogous MAX SAT Problem: Given a set of clauses $C_1, \ldots, C_k$ over a set of variables $X = \{x_1, \ldots, x_n\}$, find a truth assignment satisfying as many of the clauses as possible. Each clause has at least one term in it, and all the variables in a single clause are distinct, but otherwise we do not make any assumptions on the length of the clauses: There may be clauses that have a lot of variables, and others may have just a single variable.

Problem 3(a)

First consider the randomized approximation algorithm we used for MAX 3-SAT, setting each variable independently to *true* or *false* with probability 1/2 each. Show that the expected number of clauses satisfied by this random assignment is at least $k/2$, that is, at least half of the clauses are satisfied in expectation. Give an example to show that there are MAX SAT instances such that no assignment satisfies more than half of the clauses.

Answer: As in Problem 2, denote $Z_i = 1$ if clause C_i is satisfied and $Z_i = 0$ otherwise. The expectation

of the random variable Z_i is

$$\begin{aligned} E[Z_i] &= P\{Z_i = 1\} \cdot 1 + P\{Z_i = 0\} \cdot 0 \\ &= P\{Z_i = 1\} \\ &= 1 - P\{\text{All } t_i \text{ literals in clause } C_i \text{ are false}\} \\ &= 1 - \left(\frac{1}{2}\right)^{t_i} \\ &\geq \frac{1}{2} \end{aligned} \tag{14}$$

where $t_i \geq 1$ is the number of literals in C_i. Consequently, the expected number of clauses satisfied by the random assignment is

$$\begin{aligned} E[Z] &= \sum_{i=1}^{k} E[Z_i] \\ &\geq \frac{k}{2} \end{aligned} \tag{15}$$

An example of MAX SAT instances where no assignment satisfies half of the clauses is

$$C_1 = x_1,\ C_2 = \bar{x_1} \tag{16}$$

apparently any assignment can only satisfy half of the clauses.

Problem 3(b)

If we have a clause that consists only of a single term (e.g., a clause consisting just of x_1, or just of $\bar{x_2}$), then there is only a single way to satisfy it: We need to set the corresponding variable in the appropriate way. If we have two clauses such that one consists of just the term x_i, and the other consists of just the negated term $\bar{x_i}$, then this is a pretty direct contradiction.
Assume that our instance has no such pair of "contacting clauses"; that is, for no variable x_i do we have both a clause $C = \{x_i\}$ and a clause $C' = \{\bar{x_i}\}$. Modify the randomized procedure above to improve the approximation factor from 1/2 to at least 0.6. That is, change the algorithm so that the expected number of clauses satisfied by the process is at least $0.6k$.

Answer: We modify the random algorithm as follows: for all literals that appears in a single-literal clause, let the probability of setting the literal so that the clause is satisfied to be 0.6; For all other literals, let the probability of setting them to *true* to be 0.5.

Lemma 3.1 *This algorithm results in the expected number of satisfied clauses to be* $0.6k$

Proof Here we use the same notation as in the proof to Lemma 2.1. Consider clause C_i

- If C_i is a single-literal clause, we have $E[Z_i] = P\{Z_i = 1\} = 0.6$.
- If C_i contains at least two literals, i.e. $t_i \geq 2$, we can divide the literals in this clause into 3 classes:
  1. The literal corresponds to one of the single-literal clause. We assume there are a_i such literals.
  2. The inverse of the literal corresponds to one of the single-literal clause. We assume there are b_i such literals.
  3. The literal or its inverse don't correspond to any single-literal clause. We assume there are c_i such literals.

Based on this classification, we have $a_i + b_i + c_i = t_i \geq 2$. Then

$$\begin{aligned} E[Z_i] &= P\{Z_i = 1\} \\ &= 1 - \left(\frac{1}{2}\right)^{c_i} \cdot 0.4^{a_i} \cdot 0.6^{b_i} \\ &\geq 1 - 0.6^2 \\ &= 0.64 \\ &> 0.6 \end{aligned} \tag{17}$$

Consequently, $E[Z] = \sum_{i=1}^{k} E[Z_i] \geq 0.6k$.

Problem 3(c)

Give a randomized polynomial-time algorithm for the general MAX SAT Problem, so that the expected number of clauses satisfied by the algorithm is at least a 0.6 fraction of the maximum possible.
(Note that, by the example in part (a), there are instances where one cannot satisfy more than $k/2$ clauses; the point here is that we'd still like an efficient algorithm that, in expectation, can satisfy a 0.6 fraction of the maximum that can be satisfied by an optimal assignment.)

Answer: Similar to the proof of Lemma 2.3, we propose the random algorithm that generate random assignmentas in Problem 3(b) independently between different rounds until a truth assignment satisfying at least a $0.6k$ clauses are generated. To show that this algorithm has polynomial expected running time, denote p_j as the probability that a single round of random truth assignment results in exactly j clauses being satisfied, and p as

$$p = \sum_{j \geq 0.6k} p_j \tag{18}$$

Also define $k' = \lceil 0.6k \rceil - 1$. From Problem 3(b), we have

$$\begin{aligned} 0.6k &\leq \sum_{j < 0.6k} k' p_j + \sum_{j \geq 0.6k} k p_j \\ &= k'(1-p) + kp \\ &\leq k' + kp \end{aligned} \tag{19}$$

Therefore

$$\begin{aligned} p &\geq 0.6 - \frac{k'}{k} \\ &\geq \frac{1}{5k} \end{aligned} \tag{20}$$

since both k and k' are integers. Consequently, according to Bernoulli distribution, the expected running time of this random algorithm is

$$E_{run} = \frac{1}{p} = 5k \tag{21}$$

which is linear w.r.t the number of clauses.

Problem 4

Assume that SAT is NP-complete. Define twice-SAT as the problem of determing whether a given Boolean formula can be satisfied in at least two *different* ways. Two ways to satisfy a Boolean formula are different if at least one variable is set differently (i.e., true in one and false in the other).
Show how to reduce the SAT problem to the twice-SAT problem in polynomial time.
Assuming SAT is NP-complete, show that twice-SAT is NP complete.

Answer: Firsly we show how to reduce the SAT problem to the twice-SAT problem in polynomial time. Given a conjunctive normal form (CNF) of n Boolean variables $\phi(x_1, \ldots, x_n)$, we convert it into another CNF by adding a new variable and a 2-literal clause:

$$\phi'(x_1, \ldots, x_n, x_{n+1}) = \phi(x_1, \ldots, x_n) \wedge (x_{n+1} \vee \neg x_{n+1}) \tag{22}$$

This conversion takes constant thus polynomial amount of time. Suppose ϕ has a satisfying assignment $x_1, \ldots, x_n$, then ϕ' has at least 2 satisfying assignment $x_1, \ldots, x_n, 0$ and $x_1, \ldots, x_n, 1$. Conversely, if ϕ' has at least 2 different satisfying assignments and one of them is $x_1, \ldots, x_n, x_{n+1}$, then $x_1, \ldots, x_n$ is a satisfying assignment for ϕ.
Next we show that twice-SAT is NP complete. Since we assume SAT is NP-complete, from the first step we know that twice-SAT is NP hard. All that is left to prove is that twice-SAT is NP. Given a satisfiable twice-SAT formula ϕ of n boolean variables, the certificate we choose for twice-SAT is simply 2 different n-bit Boolean assignments. To verify each of the two assignments takes $O(\sum_{l=1}^{L} t_l)$ time where L is the total number of clauses and t_l is the number of literals in the l-th clause. Consequently, the time of verification is polynomial. On the other hand, if ϕ is not satisfiable, we can find no assignment that will pass the verification. As a result, twice-SAT is also NP.
In summary, since twice-SAT is both NP hard and NP, we conclude that twice-SAT is NP complete.

Problem 5

Approximation Algorithm for Node Cover
Recall the node cover problem:
Let G be an undirected graph with each node i given weight $w(i) > 0$. A set of nodes S is a *node cover* of G if every edge of G is incident to at least one node of S. The *weight* of a node cover S is the summation of the weights, denoted $w(S)$, of the nodes in S; the weighted node cover problem is to select a node cover with minimum weight.
The node cover problem (even when all weights are one) is known to be NP-hard, and hence we do not expect to find a polynomial-time (in terms of worst case) algorithm that is always correct. Therefore, we relax somewhat the insistence that the method be both correct and efficient for all problem instances. There are many types of relaxations that have been developed for NP-hard problems. The most common is the constant-factor, polynomial-time approximation algorithm.
For a graph G with node weights, let $S(G)$ denote the minimum weight node cover. Let A be a polynomial time algorithm that always finds a node cover, but one that is not necessarily minimum; let $S(G)$ denote the node cover of G that A finds. Then A is called a *constant-error polynomial-time approximation algorithm* (or approximation algorithm for short) if for any graph G, $S(G)/S(G) \leq c$ for some fixed constant c.
For the node cover problem we will give an approximation algorithm, based on network flow, with $c = 2$. First, recall that the node cover problem has a nice solution when the graph G is bipartite. This was done on a previous homework. You may take it as a black box at this point.
The Approximation Algorithm for General Graphs
Given G (not necessarilly bipartite), create bipartite graph $B = (N, N', E)$ as follows: for each node i in G, create two nodes i and i', placing i on the N side, and i' on the N' side of B; give both of these nodes the

Problem 5 continued on next page...

weight $w(i)$ of the original node i in G. If (i, j) is an edge in G, create an edge in B from i to j' and one from j to i'. Now find a minimum cost node cover $S(B)$ of graph B. From $S(B)$, create a node cover $S(G)$ in G as follows: for any node i in G, if either i or i' is in $S(B)$, then put i in $S(G)$.

Problem 5(a)

It is easy to find examples where $S(G)$ is not a minimum node cover of G, and where $S(G)/S(G) = 2$. Do it.

Answer:

Problem 5(b)

However, no worse error ever happens.

Theorem 5.1 *$S(G)/S(G) \leq 2$ for any G and any choice of node weights for G.*

Prove the theorem.

Answer: