HW11

Due: December 6, 2018

Attach a fully filled-in cover sheet to the front of your printed homework. Your name should not appear anywhere; the cover sheet and each individual page of the homework should include your Banner ID only.

While collaboration is encouraged in this class, please remember not to take away notes from any labs or collaboration sessions.

Please monitor Piazza, as we will post clarifications of questions there. Hand in your solutions by 10:20am to the CSCI 1010 bin on the second floor of the CIT. Late homeworks are not accepted.

Problem 1

In order to construct additional pylons, probes must deliver minerals to the appropriate nexus. The nexuses are arranged in a circle, and the probes each have a starting nexus and a destination nexus. Each probe starts at its starting nexus and must choose to go around the circle either clockwise or counterclockwise to get to its destination.

Unfortunately, there is known to be an infestation of zerglings in the area, and they will position themselves between a pair of consecutive nexuses in order to attack the largest number of probes. Help the probes by directing each one clockwise or counterclockwise so that the maximum over all edges of the number of probes that traverse that edge is minimized.

An instance of this problem is given by a cycle graph on \( n \) nodes where \( V = \{0, \ldots, n-1\} \) and \( E = \{(0,1),(1,2), \ldots,(n-2,n-1),(n-1,0)\} \), and a list of \( m \) routing jobs which are pairs of vertices denoting the start and destination of each probe (these are of the form \((s_1,d_1),(s_2,d_2), \ldots,(s_m,d_m)\) where \( s_i, d_i \in V \)).

A solution specifies the direction each probe is routed: clockwise or counterclockwise. The cost of a solution is the number of probes that go through the solution’s most overloaded edge (in other words, the maximum over all edges of the number of probes that traverse that edge); this cost function is called “Zerg Feast.” An optimal solution is one that minimizes this cost function.
Algorithm \textbf{ProbeRouting} = “On input a cycle graph of size $n$ and list of routing jobs:

1. While not all jobs have been chosen to be routed clockwise or counterclockwise:
   
   • pick an unrouted job, and route it in whichever direction is the shorter path (ties broken arbitrarily)”

Prove that this algorithm is a 2-approximation for Zerg Feast.

\textbf{Hint:} Think about the edge with the largest number of passing probes in the routing scheme produced by this algorithm, and think about the edge directly opposite this edge in the graph. What can you say about the number of probes that pass through these edges?

\textbf{Problem 2}

Leonidas is trying to find a clique. With only polynomial strength he cries out “This is madness!” Actually, this is Sparta, so only a rough approximation is needed.

a. Leonidas is given a graph $G$ with $3n$ vertices. He is told that the largest clique in $G$ is of size $2n$. Help him construct a polynomial time algorithm that uses this fact to find a 2-approximation of the largest clique. Specifically, the algorithm should return a clique of size at least $n$.

b. In class you learned that in general finding 2-approximations for maximum cliques is NP-hard. Why does this fact not contradict your answer to the previous part?

\textbf{The following questions are lab problems.}

\textbf{Lab Problem 1}

A \textbf{probabilistic Turing machine} is a Turing machine that, in addition to the usual tape, has a second tape filled with random bits. This machine may sometimes accept and sometimes reject the same input $x$, depending on the content of the random tape.
The concept of a probabilistic Turing machine gives rise to a whole new set of complexity classes, namely probabilistic polynomial time complexity classes. Some of the main ones are BPP, RP, coRP, and ZPP\(^1\), and they are defined as follows:

- A language \( L \) is in **BPP** if there exists a probabilistic Turing machine \( M \) that runs in polynomial time such that the following two conditions hold:
  
  - **Completeness**: For every \( x \in L \), \( \Pr[M \text{ accepts } x] \geq 2/3 \).
  - **Soundness**: For every \( x \notin L \), \( \Pr[M \text{ accepts } x] \leq 1/3 \).

  This Turing machine \( M \) is called a **Monte Carlo** algorithm for \( L \).

- A language \( L \) is in **RP** if there exists a probabilistic Turing machine \( M \) that runs in polynomial time such that the following two conditions hold:

  - **Completeness**: For every \( x \in L \), \( \Pr[M \text{ accepts } x] \geq 2/3 \).
  - **Perfect soundness**: For every \( x \notin L \), \( \Pr[M \text{ accepts } x] = 0 \).

- A language \( L \) is in **coRP** if its complement is in **RP**. Equivalently, a language \( L \) is in **coRP** if there exists a Turing machine \( M \) that runs in polynomial time such that the following two conditions hold:

  - **Perfect completeness**: For every \( x \in L \), \( \Pr[M \text{ accepts } x] = 1 \).
  - **Soundness**: For every \( x \notin L \), \( \Pr[M \text{ accepts } x] \leq 1/3 \).

- A language \( L \) is in **ZPP** if there exists a probabilistic Turing machine \( M \) that, in addition to the usual accepting and rejecting states, has an “unsure” state in which the machine can halt. \( M \) will run in polynomial time and at termination time will be in the accept, reject, or unsure state, such that the following two conditions hold:

  - **Completeness**: For every \( x \in L \), \( \Pr[M \text{ accepts } x] \geq 2/3 \), \( \Pr[M \text{ rejects } x] = 0 \), and \( \Pr[M \text{ is unsure on } x] \leq 1/3 \).
  - **Soundness**: For every \( x \notin L \), \( \Pr[M \text{ accepts } x] = 0 \), \( \Pr[M \text{ rejects } x] \geq 2/3 \), and \( \Pr[M \text{ is unsure on } x] \leq 1/3 \).

---

**a.** Show that \( P \subseteq \text{RP} \subseteq \text{BPP} \) and \( P \subseteq \text{coRP} \subseteq \text{BPP} \).

---

\(^{1}\)For the curious: BPP stands for bounded-error probabilistic polynomial time, RP stands for randomized polynomial time, and ZPP stands for zero-error probabilistic polynomial time.
b. Show that $\text{ZPP} = \text{RP} \cap \text{coRP}$.

c. Using either the given definition for ZPP or the one you just derived in part (b), demonstrate that the following definition is also equivalent: A language $L$ is in ZPP if there exists a probabilistic Turing machine $M$ (which does not have “unsure” states) and some polynomial $p$ such that:

- **Runtime:** For all inputs $x$ of length $n$, the expected runtime $E[\text{runtime of } M \text{ on } x]$ is $O(p(n))$.
- **Completeness:** For every $x \in L$, $\Pr[M \text{ accepts } x] = 1$.
- **Soundness:** For every $x \notin L$, $\Pr[M \text{ accepts } x] = 0$.

**Hint:** You may find Markov’s inequality to be helpful. If $X$ is a nonnegative random variable and $a > 0$, then

$$\Pr[X \geq a] \leq \frac{E[X]}{a}$$

**Lab Problem 2**

Recall that in the decision problem 3SAT, we wanted to determine whether there was an assignment of truth values that satisfied every clause of a 3CNF Boolean formula. In the optimization version Max-3SAT, the goal is to find some assignment of truth values that satisfies the maximum possible number of clauses, even if it is not possible to satisfy every single clause.

Design a (deterministic) polynomial-time algorithm that, given as input a Boolean formula in 3CNF, finds an assignment of values that satisfies at least $7/8$’s as many clauses as the maximum possible number of satisfiable clauses. You should assume that each clause contains exactly 3 literals of distinct variables. Prove that your algorithm is correct and runs in polynomial time.

**Hint:** What happens if you assign each variable randomly? (This hint may be useful to consider, but remember that your solution has to be a deterministic algorithm.)

**Lab Problem 3**

The **PCP Theorem** is one of the most important and surprising results in computational complexity, which gives an alternative characterization of the class NP and has many implications for approximability and algorithm design. One statement of the PCP Theorem is the following:
PCP Theorem. There exists a constant $\rho < 1$ such that the following holds. For every language $L$ in NP, there is a polynomial-time computable function $f$ mapping a string $x$ to a 3CNF formula $f(x)$ such that:

- If $x \in L$, then there is an assignment of variables that satisfies every clause in the formula $f(x)$.
- If $x \not\in L$, then every assignment of variables satisfies less than a $\rho$-fraction of the clauses in $f(x)$.

Note that if we replace $\rho$ with 1, then the two bullet points above are just the definition of a mapping reduction, so for any language $L$ in NP, the Cook-Levin reduction for $L$ would satisfy them. The PCP Theorem says something stronger: not only can we map strings in $L$ to satisfiable formulas and map strings not in $L$ to unsatisfiable formulas, but we can also create a gap between the maximum fraction of satisfiable clauses in the two cases.

a. Consider the types of clauses produced by the Cook-Levin reduction and explain why if $L \in$ NP, then even if $x \not\in L$, almost all of the clauses can be satisfied. In other words, the Cook-Levin reduction does not satisfy the conditions of the PCP Theorem. Thus, while mapping reductions may help in proving results about approximability, they don’t always work and have to be analyzed more carefully.

b. Using the PCP Theorem, show that if there is a polynomial-time $(1/\rho)$-approximation for Max-3SAT (where $\rho$ is the constant defined in the PCP Theorem), then P = NP.

c. Using the PCP Theorem, show that for every language $L$ in NP, there is a polynomial-time computable function $g$ mapping a string $x$ to a graph $g(x)$ such that:

- If $x \in L$, then there is a clique in $g(x)$ of size at least $(|V(g(x))|/3)$ (i.e., a clique containing at least 1/3 of the vertices in $g(x)$).
- If $x \not\in L$, then every clique in $g(x)$ has size less than $\rho(|V(g(x))|/3)$.

Together with what you have shown in class, this result implies that if Max-Clique can be approximated in polynomial time to within any constant factor, then P = NP. (For example, if there is a polynomial-time algorithm that is guaranteed to find a clique that is even $1/1000$ the size of the maximum clique, then P = NP.)