CS 3510 Homework 4

Wenqi He, whe47

December 4, 2017

1

(a)

```
ALGORITHM Verifier {
  counter = 0
  FOR (i = 1...m) {
    IF (clause i evaluates to TRUE) {
      counter++
    }
  }
  IF (counter == m-1) {
    RETURN YES
  } ELSE {
    RETURN NO
  }
}
```

The algorithm runs in $\mathcal{O}(m)$ time, assuming each Boolean evaluation is $\mathcal{O}(1)$.

(b)

We can simply pick

$$clause_{m+1} = x_1$$

$$clause_{m+2} = \neg x_1$$

(c)

If F has an assignment that satisfies SAT, then only $clause_{m+1}$ or $clause_{m+2}$ in F' evaluates to FALSE under such assginment, which means that F' satisfies ALMOST-SAT. On the other hand, if F does not have such assignment, then in addition to one of $clause_{m+1}$ and $clause_{m+2}$ being FALSE, there always exists other clauses that are also FALSE, which means that F' cannot satisfy ALMOST-SAT. Lastly, the construction of F' in (b) only takes constant time, which is of course polynomial.

2

(a)

MAX-2-SAT(F, p): Given a 2-SAT formula F, output whether it's possible to find an assignment that satisfies more than p clauses in F.

(b)

Use x_i to denote whether vertex $v_i \in S$, then we can construct a 2-SAT formula as follows:

$$F = \bigwedge_{1 \le i,j \le |V(G)|} \big((x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)\big)$$

For each edge that crosses cut S, x_i and x_j have different values, therefore both $x_i \vee x_j$ and $\neg x_i \vee \neg x_j$ evaluate to TRUE.

However, if both v_i and v_j are in S, then $x_i \vee x_j = TRUE$, but $\neg x_i \vee \neg x_j = FALSE$. Similarly, if both v_i and v_j are in $G \setminus S$, then $x_i \vee x_j = FALSE$, $\neg x_i \vee \neg x_j = TRUE$.

Thus, the statement that there are k edges leaving S is equivalent to the statement that the number of clauses that are satified in F is:

$$N(\text{edges not on the cu}t) + 2 \times N(\text{edges on the cut})$$

$$= (N - k) + 2k = N + k$$

Therefore, the reduction $\textsc{MaxCut} \to \textsc{Max2Sat}$ is

$$f_{\textsc{MaxCut} \to \textsc{Max2Sat}}(G, k) = (F_G, |E(G)| + k)$$

where

$$F_G = \bigwedge_{1 \le i,j \le |V(G)|} \big((x_i \vee x_j) \wedge (\neg x_i \vee \neg x_j)\big),$$

(c)

Since MAX-CUT is reducible to MAX-2-SAT, and MAX-CUT is already NP-hard (by definition of NP-complete), MAX-2-SAT must be NP-hard. It's also in NP because given the desired assignment, it only takes polynomial time to evaluate all m clauses to verify that the assginment indeed satisfies more than p clauses in F.

Because MAX-2-SAT is both NP-hard and NP, by definition, it is NP-complete.

(d)

First, we can reduce MAX-2-SAT to MAX-CUT using the construction provided here: `https://courses.engr.illinois.edu/cs579/sp2009/assignments/hw-1.pdf` (See Assignment 1, Problem 8). This algorithm runs in $\mathcal{O}(m)$ time and the resulting graph has $\mathcal{O}(m)$ edges. Then we can run the 2-approximation for MAXCUT on this constructed graph, which will give us the 2-approximation of MAX-2-SAT.

3

(a)

We can simply perform BFS on each vertex to find out all $DIST(u, v)$ and then pick the largest one. Since there are n vertices, this will take $\mathcal{O}(mn)$ time.

(b)

Suppose a and b are the most distant pair of vertices in this graph, then according to the triangular inequality,

$$\begin{aligned} DIST(a, b) &\le DIST(a, s) + DIST(s, b) \\ &\le \max_u DIST(s, u) + \max_u DIST(s, u) \\ &= 2 \max_u DIST(s, u) \end{aligned}$$

(c)

Randomly pick one vertex as the starting vertex s, perform BFS to compute $DIST(s, u)$ for all u, and then search for the vertex u^* with largest distance. From the result of (b),

$$DIST(s, u^*) = \max_u DIST(s, u) \ge \frac{1}{2} D,$$

which gives us the desired output s and u^*. BFS runs in $\mathcal{O}(m)$ time, and searching runs in at most $\mathcal{O}(m)$ time (in the case of linear search), so the algorithm runs in $\mathcal{O}(m)$ time.