CSC2401 Introduction to Computational Complexity
Student: **Jia Sheng**, jsheng@dgp.toronto.edu
Student #: 992810115
Instructor: Allan Borodin

Answer to Assignment 2

1 ANSWER TO PROBLEM 1

Proof

(a)

First I will show that $A \cup B \in NP$. Because A is $NP - Complete$, thus $A \in NP$, there is a non-deterministic TM M_A that accepts A in polynomial time. $B \in P$ means there is a deterministic TM M_B that accepts it in polynomial time. We can have a non-deterministic TM M that works like following:

1. On input x, M simulates M_B. If $x \in B$, thus M_B accepts, M also accepts and stops. If $x \notin B$, M_B rejects, M will turn to the next step.
2. M simulates M_A on input x and accepts if and only if M_A accepts.

We can see that M accepts all the strings in $A \cup B$ and both steps can be finished in polynomial time. So $A \cup B \in NP$.

Then to show $A \cup B$ is $NP - hard$. Since A is $NP - Complete$, $A \in NP$ and $\forall L \in NP, L \leq_p A$, which is to say $\forall L \in NP, \exists f$, s.t. f is poly time computable and $f : w \in L \Leftrightarrow f(w) \in A$

I will show that $A \leq_p A \cup B$, i.e. $\exists g$, g is poly time computable and
$g : w \in A \Leftrightarrow g(w) \in A \cup B$

So $\forall L \in NP$, there is a poly time computable function $f' = g \circ f$, s.t.

$$\begin{cases} w \in L \Rightarrow f(w) \in A \Rightarrow g(f(w)) = f'(w) \in A \cup B \\ f'(w) = g(f(w)) \in A \cup B \Rightarrow f(w) \in A \Rightarrow w \in L \end{cases}$$

Thus $L \leq_p A \cup B$. Here is the function g.

Assuming $A \cup B \neq \Sigma^*$, there $\exists$ a string $s_0 \notin A \cup B$. g can be defined as follows:

$$g(w) = \begin{cases} s_0 & \text{if } w \in B \\ w & \text{if } w \notin B \end{cases}$$

$B \in P$, thus g is poly time computable. And we have

$$\begin{cases} w \in A \Rightarrow w \notin B \Rightarrow g(w) = w, g(w) \in A \cup B \\ g(w) \in A \cup B \Rightarrow w \notin B \text{ (otherwise } g(w) = s_0 \notin A \cup B) \Rightarrow g(w) = w \\ \Rightarrow w \in A \end{cases}$$

So $A \leq_p A \cup B$ and $A \cup B$ is $NP - complete$.

(b)

The condition $A \cap B = \emptyset$ is necessary. Otherwise, there is possibility that $A \subseteq B$, thus $A \cup B = B \in P$! Here is an example.

Define SSP problem based on Subset Sum (SS) problem: given a finite set S of natural numbers $\{w_1, ..., w_n\}$ and natural number t, $(S, t) \in SSP$ if $\sum_{w_i \in S} w_i \geq t$.

The decision of whether a string (S, t) is in SSP is obviously poly time: just sum up all the numbers in S and see if the sum is larger than t. Moreover, we have $SS \subseteq SSP$.

$\forall (S, t)$, if $(S, t) \in SS$, there exists $S' \subseteq S$ s.t. $\sum_{w_i \in S'} w_i = t$. Since $w_i > 0$, we have $\sum_{w_i \in S} \geq \sum_{w_i \in S'} = t$. Then $(S, t) \in SSP$ and it implies $SS \subseteq SSP$.

So the condition $A \cap B = \emptyset$ is necessary. ■

2 ANSWER TO PROBLEM 2

Proof

First let's define the symbols used in the proof. $G = (V, E)$ is a graph, $V = \{i\}$ is the vertex set, $E = \{(i, j)\}$ is the edge set. GCO problem finds the minimum number of colors (chromatic number) needed to color G such that all adjacent vertices have different colors. GCD is a decision problem, for natural number K, $(G, K) \in GCD$ if G is K-colorable.

(a)

By using GCD as an *oracle*, we can get the chromatic number K of a certain graph G within K time (ask the GCD oracle questions like "Is G 1-colorable, 2-colorable, 3-colorable,...?" until we get a negative answer and stop at K).

We can use this oracle to solve GCO in poly time. Given a graph G, we first get the chromatic number K. Then we repetitively add edges to G: 1) choose $(i', j') \notin E$; 2) ask GCD oracle whether $G' = (V, E \cup \{(i', j')\})$ is still K-colorable; 3) if G' is still K-colorable, let $G = G'$, go on 1) to add new edges; 4) if not, i.e. G' is now $K+1$-colorable, leave G as it is and check a new edge in 1).

Continue this process until we cannot add any new edge to our current G to make it still K-colorable. We call current graph G_{full}. Any new edge to G_{full} will increase the chromatic number.

Then we can color G_{full} like this: 1) start from one vertex i, assign it color 1, search for all the vertices that are not adjacent to i and color them with the same color 1; 2) choose any un-colored vertex j, assign it color 2, search all the un-colored vertices that are not adjacent to j and color them 2, ..., continue until all vertices are assigned colors.

Take an example, G is shown on the left and G_{full} is in the middle ($K = 2$). Note there might be multiple possible G_{full}. The colored G_{full} is shown on the right.

We can see that this process will end up with a good color assignment to G_{full}, for no adjacent vertices are assigned with the same color. And this assignment is also good for the original graph G, for non-adjacent vertices in G_{full} are also non-adjacent in G.

One note about the coloring of $G_{full} = (V', E')$ is that: in step 1) when we choose one vertex i and color all its non-adjacent vertices, there cannot be edges between those non-adjacent vertices themselves (thus those vertices form an Independent Set in G_{full}). Otherwise, suppose $(i, j) \notin E'$, $(i, k) \notin E'$, while $(j, k) \in E'$. For a K-color assignment to G_{full}, j and k are assigned a and b color respectively. Then if i is assigned color different than a and b, we can connect i, j or i, k without conflicting the K-colorability. If i is assigned color a or b, we also can connect i, k or i, j without changing the chromatic number. Those both are contradictory to our assumption that G_{full} is "*full*" of edges wrt K-colorability. Thus our assignment can indeed be carried on.

To wrap up, we now have a polynomial time color assignment algorithm to G using GCD as an oracle. This completes the proof of $GCO \leq_{poly}^{T} GCD$

(b)

From [1], we can see that there are no polynomial time 4/3 approximation algorithm for the graph coloring problem, given the NP-hardness of 3-COLORABILITY problem. We can extend the result to any constant c by "*augmenting*" graph $G = (V, E)$ to $G' = (V', E')$ as following.

For graph G, we replace every vertex by G itself: vertex i becomes G_i. For $(i, j) \in E$, G_i and G_j is "jointed" by connecting all vertices in G_i with all vertices in G_j. Note that if G is K-colorable, then G' is K^2-colorable. (That is because when we take G_i as a single vertex, we need K colors to color G'. Then within each G_i, we need another K colors, which cannot be the same as any color in G_j "joining with" G_i. Thus making K^2 colors). Also is true that if G' is K^2-colorable, G is K-colorable.

The "*augmentation*" process is polynomial time bounded. So the fact that there is no poly time 4/3 approximation algorithm for G colorability problem implies that there is no poly time $(4/3)^2 = 16/9 (> 4/3)$ approximation algorithm for G' colorability problem. Continuing the procedure, we can see that for any constant c, there is no c-approximation algorithm for colorability problem.

(the idea of [1] is also augmenting G with another graph topology. The detail is omitted here) ■

3 ANSWER TO PROBLEM 3

Proof

(a) $BPP \subseteq BPP^A$

This direction is trivial, because $\forall L \in BPP$, there exists a polynomial-time PTM M that decides $x \in L$ with error probability bounded by a constant, say $1/4$. We can use the same poly-time PTM M, together with the oracle A to make oracle PTM M^A, to decide L. Here M^A does exactly the same thing as M and makes no query to A. So L is decided by M^A with the same error probability $1/4$ as well.

(b) $BPP^A \subseteq BPP$

idea Consider a language $L \in BPP^A$ with $A \in BPP$. The language L is defined by two poly-time PTMs, one for the base computation and the other, the oracle, serves for the query. What we want to show is that the two PTMs can be combined into a single PTM without creating much error. This can be done by using *majority voting* when simulating the oracle A.

proof First we will show that for any strictly increasing polynomial r, there exists a poly-time PTM N such that for every $x \in \Sigma^*$, N will accept x with at least probability $1 - 2^{-r(|x|)}$ if $x \in A$ and will accept x with at most probability $2^{-r(|x|)}$ if $x \notin A$, i.e. the error probability of N is bounded by $2^{-r(|x|)}$.

Let M is the PTM corresponding to $A \in BPP$ that decides $x \in A$ with bounded error probability $1/4$. Let $q(n) = 6r(n)+1$ and N be the PTM that simulates M on input x exactly $q(|x|)$ times. N accepts x iff M accepts in a majority of the simulations. Suppose $x \in A$, let α be the probability that M accepts x and $e = \alpha - 1/2$. Note that $e \geq 1/4$. The probability that N on input x rejects is at most

$$
\begin{aligned}
& \sum_{0\le i\le \lfloor q(n)/2\rfloor} C^i_{q(n)}(1/2+e)^i(1/2-e)^{q(n)-i} \\
\le & \sum_{0\le i\le \lfloor q(n)/2\rfloor} C^i_{q(n)}(1/2+e)^{q(n)/2}(1/2-e)^{q(n)/2} \\
= & \sum_{0\le i\le \lfloor q(n)/2\rfloor} C^i_{q(n)}(1/4-e^2)^{q(n)/2} \\
\le & \sum_{0\le i\le q(n)} C^i_{q(n)}(1/4-e^2)^{q(n)/2} \\
= & \ 2^{q(n)}(1/4-e^2)^{q(n)/2} \\
= & \ (1-4e^2)^{q(n)/2}
\end{aligned}
$$

This is at most $(3/4)^{q(n)/2} \le (3/4)^{3r(n)} < 2^{-r(n)}$. Similarly if $x \notin A$, the probability N accepts x is at most $2^{-r(|x|)}$. So we can see that the error bound of BPP can be lowered to some arbitrarily small constant without loss of polynomial time computability.

Now, for L there also exists an oracle PTM D^A with error probability bounded by $2^{-r(|x|)}$. We may assume that there exists a polynomial p such that for every $x \in \Sigma^*$, D^A makes at most $p(|x|)$ queries to A (p exists because the total time used by D^A is polynomial). Also we may assume that each query of D to A is at least as long as the input x (otherwise, just fill with dummy symbols to make the length). So we can construct our single PTM S now.

On input $x \in \Sigma^*$, S simulates D and when D makes a query ,say u, to A, S simulates N on u to decide the oracle answer. For every x, D makes at most $p(|x|)$ queries, and for every query, N makes an error with probability at most $2^{-r(|u|)} \le 2^{-r(|x|)}$ since $|u| \ge |x|$ and r is increasing polynomial. So for every x, the probability of the paths on which S computes differently from D^A is at most $p(|x|)*2^{-r(|x|)}$. Since D^A itself has the error probability $2^{-r(|x|)}$, the total probability that S makes an error is at most $(p(|x|)+1)*2^{-r(|x|)}$. Because r is increasing polynomial, for $|x|$ sufficiently large, the error probability of S is smaller than $1/4$. Thus $L \in BPP$ and $BPP^A \subseteq BPP$. ■

References

[1] Michael R. Garey, David S. Johnson: *Computers and Intractability: A Guide to the Theory of NP-Completeness*, p143, San Francisco : W. H. Freeman, 1979