Assignment #3

Due: Augest 14, 2021, 11:59 pm (EDT)

SIWEI TANG(tangsiw3)
Yuewei Wang (wangyuew)

Q1:

a) Let the problem be denoted as D. For inputs G and k, WTS:$D \in NP$ by definition. To summarize the certificates for D, we have to ensure:

- $\{G_1,\dots, G_m\}$ where$m \leq k$
- for each $i \in \{1,\dots, m\}$, $G_i \subseteq V$
- for each pair of$i, j \in \{1,\dots, m\}$, $G_i \cap G_j = \emptyset$
- $G_1 \cup\dots\cup G_m = V$
- for each$G_i \subseteq \{G_1,\dots, G_m\}$, for each pair of $u, v \in G_i$, $(u, v) \in E$

D is a decision problem that gives "yes" when such a collection $\{G_1,\dots, G_m\}$ exist. For checking these certificates, the algorithm needs to take nested-loops go over with$E$ and$V$, which takes polynomial time. By the definition of class NP, $D\in NP$.

b) From the certificates in a), D is a partitioning problem that determines whether there exists a partition to divide Vinto at most k groups according to$E$. Assume $\{G_1,\dots, G_m\}$ exists, since$m \leq k$ and $G_1 \cup\dots\cup G_m = V$, $m \geq 1$and $k \geq 1$ must be true.

Consider $k = 1$: If any independent set exists, the solution will be "no" since at least two groups of G_i exist. Finding independent sets takes polynomial time and is decidable.

Consider $k = 2$: We are checking if$\{G_1, G_2\}$exists, BFS can be used to grouping all vertices into$G_1$and G_2. For vertex $u \in V$, there are two cases for other vertices (i.e. $v \in V\backslash\{u\}$):

- if v is not grouped: then group$v$ by the result of checking $(u, v) \in E$
- if v is grouped: then check if$(u, v) \in E$. If the grouping of v against the certificates, the solution will be "no". Otherwise, the algorithm continues until all connected vertices are found.

By the grouping result from BFS, we could know if all vertices can be grouped as $k = 2$ within polynomial time (linear time), which is decidable as well.

Therefore, when $k \leq 2$, D is decidable in polynomial time.

c) part 1: From a), it follows that $D\in NP$.

part 2: WTS$D \in NP - hard$

Consider the known $NP - Complete$ problem GRAPHCOLORING, it determines whether the vertices from a graph can be colored at most k colors, where the same color

vertices are from one independent set. GRAPHCOLORING is also $NP - hard$ by the relationship between $NP - hard$and NP. We now show that GRAPHCOLORING$\leq_p D$:

D is similar as GRAPHCOLORING on the limitation of group numbers (at most$k$), but different on grouping criteria. In a group G_i, every pair of vertices has an edge. We can consider G_i as an independent set for the complementary edges in G. Then,G_i will be an independent set in a graph $G' = (V, E')$ where all edges from E' are complimentary from E (i.e. if $(u, v) \in E$, then $(u, v) \notin E'$, vice versa).

Define a function: f as:

$$f(G) = G'$$

Construct $f(G)$ takes polynomial time (linear time) to iterate the edges in E.

If GRAPHCOLORING can partition$V$in G' into at most$k$ independent sets, then we must can partition$V$in G into at most$k$ groups for D since the edges in E' is complementary to$E$. In another hand, if Vcan be grouped into at most$k$ groups for D, then GRAPHCOLORING $(f(G),\ k)$must be true for coloring at most k independent sets. Thus, GRAPHCOLORING $\Leftrightarrow D$.

Then, GRAPHCOLORING$\leq_p$D.

Since GRAPHCOLORING is $NP - hard$ when $k \geq 3$, then$D \in NP - hard$ when $k \geq 3$.

Therefore, $D \in NP - Complete$ by definition.

Q2:

a) part 1**:** DISJOINTHAMILTONIANPATHS$\in NP$

Given inputs $G = (V, E)$, we will have certificates to check every combination of two sets vertices in all possible permutations (i.e. $\{u_1,..., u_n\}, \{u'_1,..., u'_n\}$):

- check if the permutation could be a path by checking the edge between every pair of vertices in order (i.e.$(u_i, u_{i+1}) \in E$for $i \in \{1,..., n-1\}$), for both $\{u_1,..., u_n\}\ and\ \{u'_1,..., u'_n\}$
- check if no edge exists twice in both permutations (i.e $(u_i, u_{i+1}) \neq (u'_j, u'_{j+1})$ for $i, j \in \{1,..., n-1\}$)

If any two sets of vertices satisfy for both certificates, then$G$ has at least two edge-disjoint Hamiltonian paths and gives the decision "yes". Otherwise, it returns "no " until the algorithm checks all permutations. The algorithm needs to construct all permutations and take nested-loops go over with$E$ and$V$for checking the criteria, which takes polynomial time. By the definition of class NP, DISJOINTHAMILTONIANPATHS $\in NP$.

part 2: DISJOINTHAMILTONIANPATHS$\in NP - hard$

Consider the known $NP - Complete$ problem UHAMPATH, it determines whether an undirected graph has a Hamiltonian path. UHAMPATH is also $NP - hard$ by the relationship between $NP - hard$and NP. We now show that UHAMPATH$\leq_p$ DISJOINTHAMILTONIANPATHS:

Let $G = (V, E)$ is undirected, we can construct it into directed graph$G' = (V, E')$ where $E \subseteq E'$ by providing two directions for each edge in E(i.e. if $(u, v) \in E$, then $(u, v),\ (v, u) \in E'$).

Define a function: f as:

$$f(G) = G'$$

Construct $f(G)$ takes polynomial time (linear time) by duplicate the edges in E.

If UHAMPATH(G)gives "yes", then exists two paths in opposite directions inG'(i.e. $\{u_1,... , u_n\}, \{u_n,... , u_1\}$). Then DISJOINTHAMILTONIANPATHS$(f(G))$gives "yes" as well. In another hand, if DISJOINTHAMILTONIANPATHS$(f(G))$returns true, then at least two edge-disjoint Hamiltonian paths exist inG'. UHAMPATH(G)must be true due to either path from inG'. Thus, UHAMPATH $\Leftrightarrow$DISJOINTHAMILTONIANPATHS.

Thus, UHAMPATH$\leq_p$DISJOINTHAMILTONIANPATHS.

Since UHAMPATH is $NP - hard$, DISJOINTHAMILTONIANPATHS must also be $NP - hard$.

Overall, DISJOINTHAMILTONIANPATHS is $NP - Complete$ by the definition.

b)

The definition of *DISJOINTHAMILTONIANPATHS-SEARCH* problem:

Given a graph $G = (V, E)$ in either undirected or directed, return the two edge-disjoint Hamiltonian paths if the graph has. Otherwise, return "NIL".

WTS: *DISJOINTHAMILTONIANPATHS-SEARCH* problem is polynomial-time self-reducible

We will solve the problem by using DISJOINTHAMILTONIANPATHS to select all edges belonging to the two edge-disjoint Hamiltonian paths. Also using HAMPATH-SEARCH to separate the two paths:

```
1:procedure DISJOINTHAMILTONIANPATHS-SEARCH(G)
2:      if not DISJOINTHAMILTONIANPATHS(G) then
3:              return NIL
4:      E' := {}
5:      for each edge e ∈ E do
6:              E' = E\{e}
7:              G' = (V, E')
8:              if DISJOINTHAMILTONIANPATHS(G') then:
9:                      E = E'
10:     e := E\ an arbitrary edge in E
11:     G' = E\{e}
12:     E1: =HAMPATH-SEARCH(G')
13:     E2: = E\E1
14:     return E1 ∪ E2
```

Correctness: DISJOINTHAMILTONIANPATHS*(G')* remains true at every step to select all edges belonging to any path of two edge-disjoint Hamiltonian paths in$G$. Then other edges are removed from$E$. G'contains all edges in$E$ except e, HAMPATH-SEARCH*(G')* gives the Hamiltonian path of without e , by which two paths can be separated. The algorithm uses the decision problem, which is self-reducible by the definition.

Runtime: Let $t(n, m)$be the runtime of DISJOINTHAMILTONIANPATHS on a graph of n vertices and m edges. In line 5 to 9, DISJOINTHAMILTONIANPATHS executes for m-round. In line 12, HAMPATH-SEARCH(G') is polynomial (from lecture). Let m' be the edge number ofG', then$m' \leq m$. The total runtime is $O(m \times t(n, m) + (m' \times t(n, m'))$, which is polynomial if $t(n, m)$is.

Q3:

a) The decision problem *3-PCNF-DECISION*:

Given a propositional formula φ in 3-PCNF and a non-negative integer k, is it possible to make φ is satisfiable by setting at most$k$ variables to TRUE?

WTS: the 3-PCNF optimization problem is polynomial-time self-reducible

We will have the 3-PCNF optimization algorithm by using *3-PCNF-DECISION* to check if each variable can be replaced and satisfy φ:

1:**procedure 3-PCNF(φ)**
2: k: =the number of variables in φ
3: **while** *3-PCNF-DECISION(φ,* $k-1$*)* **do**
4: $k = k - 1$
now$k$ is the minimum number of variables that makes φ be satisfiable
5: $S: = \{\}$
6: n: =the number of variables in φ
7: **for** $i = 1$ *to* n **do**:
8: φ': = φ by relacing $x_i$by using other variables to represent
9: **if** *3-PCNF-DECISION(φ', k)* **and** (x_i) is not a clause in φ **then**
10: x_i: =FALSE
11: φ = φ'
12: **else**
13: x_i: =TRUE
14: $k = k - 1$
15: φ = remove the clause with x_i since x_i is TRUE
16: $S = S \cup \{x_i\}$
17: **return** *S*

Correctness: both *3-PCNF-DECISION(φ', k)* remains true at every step if φ' has unvisited variables only after replacingx_i. If (x_i) is a clause in φ, then x_i must be TRUE to make φ be satisfiable. Then each x_i is whether TRUE or FALSE is determined and collected into S. The algorithm uses this decision problem to check in each iteration, which is self-reducible by the definition.

Runtime: Let $t(n,k)$be the runtime of *3-PCNF-DECISION* for φ in n variables and non-negative integer k. Then line 3 while loop takes $O(t(n,k))$. In the line 7 for loop, *3-PCNF-DECISION(φ', k)* executes for all n variables in φ. Then the total runtime is $O(t(n,k) + n \times t(n,k))$, which is polynomial if $t(n,k)$is.

b) Let OPT be the minimum value of solution in the problem, n be the number of variables and k be the number of clauses in φ, we will have the integers to represent TRUE or FALSE, with each $x_i$as variables:

minimize: $\sum_{i=1}^{n} x_i$

subject to: $x_i \in \{0, 1\}$ for $i \in \{1,..., n\}$, where 0 is FALSE and 1 is TRUE

$$(x_{i_a} \vee x_{i_b} \vee x_{i_c}) = (x_{i_a} + x_{i_b} + x_{i_c}) \geq 1 \text{ for } i \in \{1,..., k\}$$

Convert the IPP to an LPP, we have constraint that:

$$0 \leq x_i \leq 1 \text{ for } i \in \{1,..., n\}$$

Now, compute an approximation solution to this LPP as $\{x_1^*, \dots, x_n^*\}$. Each x_i^* for $i \in \{1,\dots, n\}$ will have two cases:

- $x_i^* \geq \frac{1}{3}$: which means that x'_i is TRUE, thus make the clause be TRUE. Then $x_i = 1$
- $x_i^* < \frac{1}{3}$: then x'_i is FALSE and $x_i = 0$

Thus, if a set of $\{x_1, \dots, x_n\}$ satisfies that $x_i^* \geq \frac{1}{3}$ for $i \in \{1,..., n\}$ in $\{x_1^*, \dots, x_n^*\}$, then that set is the OPT.

Correctness:

In the above program, we can compute the solution in the LPP as $\{x_1^*, \dots, x_n^*\}$. $\frac{1}{3}$ is the boundary value to distinguish TRUE and FALSE since the constraint $(x_{i_a} \vee x_{i_b} \vee x_{i_c}) = (x_{i_a} + x_{i_b} + x_{i_c})$. For x_i^*, if the clause $(x_{i_a}^* \vee x_{i_b}^* \vee x_{i_c}^*)$ is TRUE, then $(x_{i_a}^* + x_{i_b}^* + x_{i_c}^*) \geq 1$, which requires at least one of $x_{i_a}^*, x_{i_b}^*, x_{i_c}^*$ be greater than or equal to $\frac{1}{3}$ for $i_a, i_b, i_c \in \{1,\dots, k\}$.

Thus, if a set of $\{x_1, \dots, x_n\}$ satisfies that $x_i^* \geq \frac{1}{3}$ for $i \in \{1,..., n\}$ in $\{x_1^*, \dots, x_n^*\}$, then that set is the OPT.

Complexity:

Solving the optimization problem in LPP takes linear time with additional constraint, which is $OPT \geq \sum_{i=1}^{n} x_i^*$. Minimizing $\sum_{i=1}^{n} x_i^*$ can evaluate $\sum_{i=1}^{n} x_i$. Since $\frac{1}{3}$ is the boundary to differentiate x_i^* is whether TRUE or FALSE, we have $3x_i^*$ for evaluating x_i.

Then, $\sum_{i=1}^{n} 3x_i^* \geq \sum_{i=1}^{n} x_i$

$$3\sum_{i=1}^{n} x_i^* \geq \sum_{i=1}^{n} x_i$$

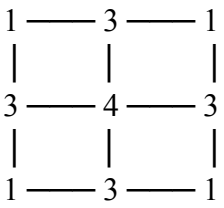
$$3OPT \geq 3\sum_{i=1}^{n} x_i^* \geq \sum_{i=1}^{n} x_i$$

Therefore, if the minimum number of variables needed to be set to TRUE to make φ satisfiable is k^*, the algorithm takes $3k^*$.

Q4:

a)

An counter-example would as the map below:



The greedy algorithm will return $S = \{(3, 1), (3, 3), (2, 2), (1, 1), (1, 3)\}$. Corner (2, 2) has the maximum profit of 4 initially, thus $S = \{(2, 2)\}$after this first round of checking. C removes all adjacent corners and remains $\{(3, 1), (3, 3), (1, 1), (1, 3)\}$as the second round options. The same procedure repeats to choose the maximum profit corner until no options left in$C$, The solution S eventually has a total profit of 8.

However, the optimal solution regarding the map is $S = \{(3, 2), (2, 1), (2, 3), (1, 2)\}$ with a total profit of 12.

b)

Consider the input map as an undirected graph, each corner coordinates be vertices and the neighbourhood relationships between corners be edges. The criteria of no choosing from $\{(i - 1, j), (i + 1, j), (i, j - 1), (i, j + 1)\}$when selected $(i, j)$is equivalent to choosing the vertex from an independent set in a graph.

Define MIN be the minimum total profit from any independent sets from the greedy algorithm, MAX be the maximum total profit from the algorithm. WTS: the greedy algorithm has an approximation ratio of 4 (i.e. $MIN \geq \frac{1}{4}MAX$ for all input graphs)

Let S be the selection (independent set) returned by the greedy algorithm, and let$T$ be any other valid selection (independent sets) that has the maximum total profit. Then, there are two cases for a corner (i, j) :

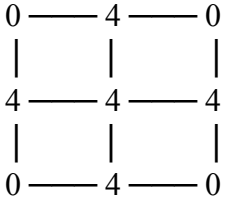
- if $(i, j) \in S$: then $p_{i,j}$is the max profit over itself and all adjacent corners. S selects (i, j)
- if $(i, j) \notin S$: which means $(i, j) \in T$, then exists an adjacent corner $(i', j') \in S$ satisfy that $p_{i',j'} \geq p_{i,j}$. C removes (i, j)since it is adjacent to a chosen corner in S

Note that each corner will have at most 4 neighbours, thus for a corner $(i, j) \in S$ we have the profit relationship that: $p_{i,j} \geq p_{i',j'}$ where $(i', j') \in \{(i - 1, j), (i + 1, j), (i, j - 1), (i, j + 1)\}$.

Then, $4p_{i,j} \geq p_{i-1,j} + p_{i+1,j} + p_{i,j-1} + p_{i,j+1}$ where $(i, j) \in S$ and $(i - 1, j), (i + 1, j), (i, j - 1), (i, j + 1) \in T$

$$p_{i,j} \geq \frac{1}{4}(p_{i-1,j} + p_{i+1,j} + p_{i,j-1} + p_{i,j+1})$$

There could be a special case that: there are multiple options for the maximum profits initially, but only one with the optimal solution. It is possible that the greedy algorithm chose the corner in any non-optimal solution and returned the results. For instance:



$S = \{((3,\ 2),\ (2,\ 1),\ (2,\ 3),\ (1,\ 2))\}$should be the optimal solution by observation, but it is possible that the algorithm will select (2, 2) initially since the profit is also maximum in the first round checking. Then the approximation ratio = 4 exactly.

By the definition of $MIN$and MAX, we have $MIN \geq \frac{1}{4}MAX$.