MINERVA SCHOOLS AT K.G.I.
Final Assignment
Quang Tran
CS142 Fall 2019

Part 1. P-Class

Question a. General description of the problem

The 3SUMx3 problem: Given three sets of integers A, B, and C, determine whether there exists a triplet (a, b, c) where $a \in A,\ b \in B,\ c \in C$ and $a + b + c = 0.$ This is an extension of the 3SUM problem, which determines if a set of integers contains three numbers that sum to 0.

Question b. Write the problem as a language.

We define the following language:

3SUMx3 = {<A,B,C> | A, B, C are sets of integers and there exists $a \in A,\ b \in B,\ c \in C$ and $a + b + c = 0\}$

Therefore, solving the 3SUMx3 problem is equivalent to checking whether the given three sets <A,B,C> is in the language 3SUMx3.

Question c. Give a High-level description of the Turing machine that solves the problem in polynomial time.

We first discuss the algorithms for solving this problem and will later build a TM.

The 3SUMx3 problem has an obvious solution: we try all possible triplets (a,b,c) and check whether any of them sums to 0. Suppose |A| is the number of elements in set A, then, because there are |A| ways to choose $a,$ |B| ways to choose $b,$ and |C| ways to choose $c,$ there are |A||B||C| triplets in total and this algorithm takes $O(|A||B||C|)$ time. If $|A| = |B| = |C| = n,$ then the algorithm takes $O(n^3)$ time. This is polynomial, but below we seek another polynomial but more efficient algorithm.

This algorithm would use the solution to 3SUM problem as a subroutine. Therefore, first, we will solve the 3SUM problem. Let us define the following language:

3SUM = {<A> | A is a set of integers and $a + b + c = 0$ with $a \in A,\ b \in A,$ and $c \in A$}

The TM $TM3SUM$ that decides 3SUM is given below with a high-level description. The idea is that we first sort A in ascending order. Then we assign the first element as a . For b and c, we assign the next smallest element to b and the last element in the list to c (two ends of the array if we discard the first, smallest element.) We check if $a+b+c=0.$ If not, depending on whether the sum is larger or smaller than 0, we reassigning c or b respectively. If $a+b+c<0,$ it means the sum needs to be increased if it wants to be equal to 0, so we assign b to the next element in the array. If $a+b+c>0,$ it means the sum needs to be decreased if it wants to be equal to 0, so we assign c to the next largest element. We continue to bridge the distance between b and c this way until they cross each other, at which time we remove the first element from the array, and reassign a to the smallest element in the array, repeating the process. Below is an example:

Iteration	Array	a	b	c	a+b+c
1	[-4,-2,1,4,7]	-4	-2	7	1>0
	[-4,-2,1,4,7]	-4	-2	4	-2<0
	[-4,-2,1,4,7]	-4	1	4	1
2	[-2,1,4,7] (-4 is removed)	-2	1	7	6>0
	[-2,1,4,7]	-2	1	4	3>0
3	[1,4,7] (-2 is removed)	1	4	7	12>0

TM3SUM = “On input <A>:

1. Sort A[0, 1, 2, …, n-1] in ascending order.
2. For $i=0$ to $n-2$ do:
3. Let $a \leftarrow A[i],\ b' \leftarrow i+1,\ c' \leftarrow n-1$
4. While $b' < c'$ do:
5. $b \leftarrow A[b'],\ c \leftarrow A[c']$
6. If $a+b+c=0$, *accept*
7. If $a+b+c<0,\ b' \leftarrow b'+1$
8. If $a+b+c>0,\ c' \leftarrow c'-1$
9. *reject* if this stage is reached.”

We now analyze the complexity of TM3SUM:

- Stage 1 does the sort, which can be done in polynomial time (for example $O(n^2)$ time, by using merge sort which runs in $O(n\ log\ n)$ which means it also runs in $O(n^2)$)

- Stage 3 is done at most $n-1$ time and is just assignments, which can be done in constant time.
- Stages 4 to 8 are done at most $(n-2)(n-1) = O(n^2)$ (the outer *for* loop and the *while* loop) times, and each of them runs in constant time.
- Stage 9 is run at most 1 time (if it is reached), and runs in constant time.

Overall, the complexity of the algorithm is $O(n^2)$.

We now use the solution to 3SUM to solve 3SUMx3:

TM3SUMx3 = "On input <A,B,C>:

1. For each element a of A, transform a into $a' = 10a + 1$
2. For each element b of B, transform b into $b' = 10b + 2$
3. For each element c of C, transform c into $c' = 10c - 3$
4. Concatenate A, B, and C into one array D.
5. Run TM3SUM on <D>
6. If TM3SUM accepts, *accept.* Otherwise, *reject*."

Proof of correctness: We will prove that $a' + b' + c' = 0$ iff $a' \in A,\ b' \in B,\ c' \in C$ and $a + b + c = 0.$

We prove the first direction: suppose we have $a' + b' + c' = 0.$ Assume for the sake of contradiction that a', b', c' are not from three distinct arrays A, B, and C. We have the following cases:

Case #	Number of integers from A	Number of integers from B	Number of integers from C	$a' + b' + c'$
1	2	1	0	$20a + 10b + 4 = 10K + 4 \neq 0$
2	2	0	1	$20a + 10c - 1 = 10K + 9 \neq 0$
3	1	2	0	$10a + 20b + 5 = 10K + 5 \neq 0$
4	0	2	1	$20b + 10c + 1 = 10K + 1 \neq 0$
5	1	0	2	$10a + 20c - 5 = 10K + 5 \neq 0$
6	0	1	2	$10b + 20c - 4 = 10K + 6 \neq 0$
7	3	0	0	$30a + 3 = 10K + 3 \neq 0$
8	0	3	0	$30b + 6 = 10K + 6 \neq 0$
9	0	0	3	$30c - 9 = 10K + 1 \neq 0$

We see that it's impossible for $a' + b' + c' = 0,$ so the assumption that a', b', c' are not from three distinct arrays A, B, and C is wrong and a', b', c' must be from three arrays.
We also have $a + b + c = \frac{a'-1}{10} + \frac{b'-2}{10} + \frac{c'+3}{10} = \frac{a'+b'+c'}{10} = 0.$

Now, we prove the reverse direction: suppose we have $a' \in A,\ b' \in B,\ c' \in C$ and $a + b + c = 0.$ Then $a' + b' + c' = (10a + 1) + (10b + 2) + (10c - 3) = 10(a + b + c) + (1 + 2 - 3) = 10(a + b + c) = 0.$ Our proof is complete. Thus, the algorithm dictated by TM3SUMx3 is correct.

Question d. Complexity analysis.

Consider TM TM3SUMx3:

- Stage 1 to 4 each take $O(n)$ because they perform only one multiplication, one addition, and/or concatenation.
- Stage 5 runs TM3SUM, the running time of which was proven to be quadratic, so the running time of this stage is $O((|A| + |B| + |C|)^2)$. If $|A| = |B| = |C| = n,$ then it's $O(9n^2) = O(n^2).$
- Stage 6 takes constant time.

Overall, it takes $O(n^2),$ which is polynomial. Because we have shown a high-level description of a decider for SUMx3 that runs in polynomial time, it means that SUMx3 is in class P (proof by construction.)

Part 2. NP Class and NP-Complete Problems

Question a. Write a general description of the NP problem.

Vertex cover problem:

To describe this problem, we first define vertex cover:

- **Incidence:** An edge in a graph is said to be incident to a vertex if the vertex is one of the two vertices that the edge connects.
- **Vertex cover:** Given a graph G(V,E) where V is the set of vertices and E is the set of the edges, a vertex cover is a subset V' of V such that every edge in E is incident to at least one vertex in V'.

The following is a figure from Wikipedia in which the red nodes are members of a vertex cover of the graph:

Vertex cover problem is a **decision problem**: given an undirected graph G(V,E) and a positive integer k, determine whether there exists a vertex cover of the graph that has size k. In the figure above, the vertex cover has size 3. Note that every graph has at least one vertex cover, which is V and has size |V|. Therefore, there is a related, optimization problem, which is to find a vertex cover that has the smallest possible size. However, we only investigate the decision version in this assignment.

Question b. Write the problem as a language.

We define the following language:

VERTEXCOVER =
$\{< G(V, E),\ k\ > \mid G \text{ is an undirected graph and there exists } V' \subseteq V,\ |V| = k \text{ such that } uv \in E \Rightarrow u \in V' \vee v \in V'\}$

Solving the vertex cover problem is equivalent to determining whether $(G(E, V), k)$ is in the language.

Question c. Show that the problem is NP.

We will do this in two ways:

Constructing a non-deterministic polynomial time Turing machine for VERTEXCOVER:

TMVC = "On input <G(V,E), k>:

1. Nondeterministically select k distinct vertices to put in set $V' = \{v_1, v_2, ..., v_k\}$ from V.
2. For each edge in E, check if it has at least one vertex in V' as one of the vertices it connects. If any of the checks fails, *reject*. *accept* otherwise."

Construct a polynomial time verifier VER with the subset V' as a certificate:

VER = "On input <G(V,E), k, V'>:

1. Test whether |V'|=k. If not, *reject.*

2. For each edge in E, check if it has at least one vertex in V' as one of the vertices it connects. If any of the checks fails, *reject*. Otherwise, *accept.*"

We now analyze the complexity of **TMVC:**

- Stage 1 is selecting k vertices from a pre-defined set, so it is polynomial in time.
- Stage 2 runs in time proportional to $|E||V'| = k|E| \leq |V||E|$, which is polynomial in the size of the graph.

As such, **TMVC** is polynomial ($O(k|E|)$)

Because each stage in TMVC is guaranteed to halt, it's the decider. Because we are able to construct a non-deterministic polynomial time Turing machine decider for VERTEXCOVER, VERTEXCOVER is in NP class.

Now we analyze the complexity of **VER:**

- Stage one traverses through V' to count the number of elements. However, the check can stop once the counter reaches $k+1,$ so it runs in $O(k).$
- Stage 2 runs in time proportional to $|E||V'| = k|E| \leq |V||E|$, which is polynomial in the size of the graph.

So the verifier runs, in worst-case, in $O(k|E|),$ which is also polynomial.

Because we can construct a polynomial time verifier for VERTEXCOVER, it's in NP class.

Question d. Identify a second computational problem that is NP-Complete, and define a polynomial-time reduction function to solve your problem by a reduction to the NP-Complete problem.

The NP-Complete problem: Set cover problem.

Problem description: This is one of Karp's 21 NP-complete problems. The set cover problem is also a decision problem. Given:

1. A finite set S
2. A set U of *some* (not necessarily all) subsets of S
3. An integer *k*

Determine whether there is a set of size k $U' \subseteq U$ such that the union of all the sets in U' is S.

An example of an instance of the problem:

- Inputs:
  - $S = \{1, 2, 3, 4, 5\}$
  - $U = \{\{1\}, \{1, 2\}, \{4, 5\}, \{2, 3, 4, 5\}\}$

- $k = 2$
- Output:
  - Yes. $U' = \{\{1\}, \{2,3,4,5\}\}, \ |U'| = k = 2,$ and

$$\bigcup_{u \in U'} u = \{1\} \cup \{2,3,4,5\} = \{1,2,3,4,5\} = S$$

We express this in a language:
SETCOVER =

$$\{< S, U, k > \mid U \text{ is a set of some subsets of } S \text{ and there exists } U' \subseteq U \text{ such that } |U'| = k \text{ and } \bigcup_{u' \in U'} u' = S\}$$

Reduction from VERTEXCOVER to **SETCOVER:**

Because **SETCOVER** is in NP-complete class, any language in NP can be polynomial time reducible to it. Because we proved that VERTEXCOVER is in NP, VERTEXCOVER must be polynomial time reducible to SETCOVER.

Given an instance <G(E,V), m> of the vertex cover problem, we will reduce it to an instance of set cover problem by mapping <G(E,V), m> to <S,U,k> as follows:

1. $S = E$. Let the set of edges be S.
2. $U = \{\{e \mid e \in E \text{ and } e \text{ is incident to } v\} \mid v \in V\}.$ U is a set of some subset of S. Each set in U_i corresponds to a vertex i in G and contains all the edges incident to that vertex. For example, suppose the graph is $1 - 2 - 3$ (node 1 connects node 2 which connects node 3) then $S = \{(1,2),(2,3)\}$ and $U = \{\{(1,2)\}, \ \{(1,2),(2,3)\}, \ \{(2,3)\}\}$
3. $k = m.$

We will prove that this reduction works by proving a two-direction statement:

$$< G(E,V), \ m > \ \in VERTEXCOVER \text{ iff } < S, U, k > \ \in SETCOVER"$$

- Prove $< G(E,V), \ m > \ \in VERTEXCOVER \Rightarrow \ < S, U, k > \ \in SETCOVER$

$< G(E,V), \ m > \ \in VERTEXCOVER$ means that there is $V' \subseteq V$ that is a vertex cover and has size $m.$ Let K be the set of all edges that is incident to some vertex in V'. By definition of a vertex cover, $K = E.$

We can also build K by looping through every vertex v in V' and adding the edge that is incident to vertex v to K, which means we will take the union of all the sets in U that corresponds to a vertex in V':

$$E = K = \bigcup_{u \in U'} u$$

where $U' = \{\{e \mid e \in E \text{ and } e \text{ is incident to } v\} \mid v \in V'\}$

Because U' has size $|V'| = m = k,$ we have found k subsets of S whose union is S, which means:

$$< S, U, k > \ \in SETCOVER$$

- Prove $< S, U, k > \in SETCOVER \Rightarrow < G(E, V),\ m > \in VERTEXCOVER$

$< S, U, k > \in SETCOVER$ means that there are k sets in U whose union is $S = E.$ Suppose without loss of generality those k sets are $U_1, U_2, ..., U_k$.

Now, if we choose $V' = \{1, 2, 3, ...k\}$ then V' is the vertex cover of G. To show this we show any edge $e \in E$ is incident to some vertex in V. Because $S = E,\ e \in S.$ Because S is the union of $U_1, U_2, ..., U_k,\ e \in U_r$ for some $1 \le r \le k.$ Because U_r is the set of all the edges incident to vertex $r,\ e$ is incident to $r.$

We have just shown that V' is the vertex cover of G, which means $< G(E, V),\ m > \in VERTEXCOVER$

Our proof is complete.

Now that we have shown it is indeed a reduction, we just need one more step: showing it runs in polynomial time. We repeat the procedure for the mapping below for convenience:

1. $S = E$. Let the set of edges be S.
2. $U = \{\{e \mid e \in E \text{ and } e \text{ is incident to } v\} \mid v \in V\}.$ U is a set of some subset of S. Each set in U_i corresponds to a vertex i in G and contains all the edges incident to that vertex. For example, suppose the graph is $1 - 2 - 3$ (node 1 connects node 2 which connects node 3) then $S = \{(1,2), (2,3)\}$ and $U = \{\{(1,2)\},\ \{(1,2), (2,3)\},\ \{(2,3)\}\}$
3. $k = m.$

- Step 1 takes time $O(|E|),$ if we view the operation $S = E$ as copying elements from E to S.
- Step 2 takes $O(|V|^2)$ because we need to loop through all $v \in V$ and each vertex has at most $|V| - 1$ edges connecting it with another vertex.
- Step 3 takes constant time.

Over all, the reduction takes $O(|E| + |V|^2),$ which is polynomial in the size of the graph.

We conclude that the mapping is **polynomial time reduction** from **VERTEXCOVER** to **SETCOVER**

Question e. Show that your problem is NP-Complete. If the problem is not NP-Complete, justify your conclusion. If the problem is NP-Complete, use a third problem from either the P class or the NP class that reduces to your original NP problem.

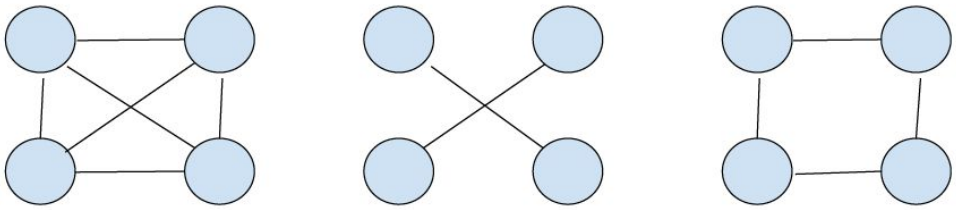
The third problem we choose to prove that VERTEXCOVER is NP-complete is the CLIQUE problem, introduced in Sipser. CLIQUE problem is proven, in Sipser, to be NP-complete, and thus, by definition, it is also in the NP-class.

Below is the proof sketch to prove that VERTEXCOVER is NP-complete:

1. Prove that VERTEXCOVER is in NP class
2. Prove that CLIQUE is polynomial time reducible to VERTEXCOVER.

We have proven (1) in question c. What is left is to prove (2). Like in the last question, we will build a polynomial time reduction from CLIQUE to VERTEXCOVER by mapping an instance $< G(E, V), k >$ of CLIQUE to $< G'(E', V'), k' >$. The mapping is as follows:

1. E' is all edges not in E in a complete graph with vertices V, and $V' = V$. The figure below shows an example



Complete graph G G'

2. $k' = |V| - k$

We will prove that this reduction works by proving a two-direction statement:

$< G(E, V), k > \in CLIQUE$ iff $< G'(E', V'), k' > \in VERTEXCOVER$"

- Prove that $< G(E, V), k > \in CLIQUE \Rightarrow < G'(E', V'), k' > \in VERTEXCOVER$

$< G(E, V), k > \in CLIQUE$ means that there is a clique of size k. Let V_{clique} be the set of vertices in the clique. We have $|V_{clique}| = k$. We will prove that $V'' = V \setminus V_{clique}$ is a vertex cover

of size k' of G' because that would mean $< G'(E', V'), k' > \in VERTEXCOVER$ (the thing we have to prove).

First, because $V_{clique} \subseteq V$, $|V''| + |V_{clique}| = |V| \Rightarrow |V''| = |V| - |V_{clique}| = |V| - k = k'$. Therefore, the size of V'' is indeed k'.

Next, to prove that V'' is a vertex cover of G', we need to show that every edge $(u, v) \in E'$, we must have $u \in V'' \vee v \in V''$. Suppose for the sake of contradiction that $u \notin V'' \wedge v \notin V''$:

$\Rightarrow u \in V_{clique} \wedge v \in V_{clique}$
This means the edge (u, v) is in the clique of graph G which also means $(u, v) \in E$. However, because $(u, v) \in E'$, (u, v) is an edge in both E and E'. This is not possible, because the way we construct E', $E \cap E' = \varnothing$. We have obtained a contradiction. Thus, $u \in V'' \vee v \in V''$ and V'' is indeed a vertex cover of G' which means $< G'(E', V'), k' > \in VERTEXCOVER$

- Prove that $< G'(E', V'), k' > \in VERTEXCOVER \Rightarrow < G(E, V), k > \in CLIQUE$

$< G'(E', V'), k' > \in VERTEXCOVER$ means that there is a set of vertices V_{cover} that is a vertex cover of G' and $|V_{cover}| = k'$. We will prove that $V'' = V \setminus V_{cover} = V \setminus V_{cover}$ forms a k-clique for G.

We have that $|V''| = |V'| - |V_{cover}| = |V| - k' = k$. The size of V'' is indeed k.

Next, we must show that for any $u, v \in V''$, $(u, v) \in E$. Suppose, for the sake of contradiction, that $(u, v) \notin E$. Then $(u, v) \in E'$ (because $E + E'$ is forms a complete graph, which has an edge connecting any pair of nodes). Because $(u, v) \in E'$, by definition of a vertex cover, $u \in V_{cover} \vee v \in V_{cover}$. This contradicts with the fact that $u, v \in V''$which also means $u \notin V_{cover} \wedge v \notin V_{cover}$. We have obtained a contradiction. Ergo, $V''$is a k-clique for G.

We have just proven that the above mapping is a mapping reduction from CLIQUE to VERTEXCOVER. What is left to prove is that the reduction is polynomial time. Let us repeat the mapping steps below for convenience:

1. E' is all edges not in E in a complete graph with vertices V , and $V' = V$.
2. $k' = |V| - k$

- Step one runs in time proportional to the number of edges in a complete graph with vertices V. That number is $|V|(|V| - 1)/2 = O(|V|^2)$.
- Step two runs in constant time.

The total time of mapping is $O(|V|^2)$, polynomial in the size of graph. Therefore, it is a polynomial time reduction.

Because:

1. VERTEXCOVER is in NP
2. CLIQUE is NP-complete
3. $CLIQUE \leq_P VERTEXCOVER$

according to Theorem 7.36 (Sipser), VERTEXCOVER is NP-complete.

Part 3. Polynomial Simulators

We will simulate TMVC, a non-deterministic polynomial time Turing machine for VERTEXCOVER. Please see the code in the notebook attached.

Let us repeat the TM's description here for convenience:

TMVC = “On input <G(V,E), k>:

1. Nondeterministically select k distinct vertices to put in set $V' = \{v_1, v_2, ..., v_k\}$ from V.
2. For each edge in E, check if it has at least one vertex in V' as one of the vertices it connects. If any of the checks fails, *reject*. *accept* otherwise.”

Part 4. Summary

In this assignment I have studied the computability and complexity of various problems. The concept of computability, defined as Turing-decidable, is helpful in terms of the structural method of proving that a problem is solvable (we only need to create a decider for it.) Once a problem has been the determined to be solvable, one is still left with the question of the complexity of the solutions. Here where the theory of complexity plays both theoretical and practical roles by classifying problems according to their complexity. If a problem is said to be in P class, then generally we consider it being solved in practically reasonable time. The NP-complete class lays a theoretical, guiding foundation for the answering the question of whether P=NP. The highlight of my work, is the use of **reducibility** concept in classifying problems by making use of previously known type of another problem.

The 3SUM problem provides an analogy to how we define NP-complete. In the field of theory of computation and complexity, 3SUM-hard has been defined for a problem whose a sub-quadratic

solution implies a sub-quadratic solution for 3SUM. This is similar to how we classify NP-complete: any problem in this class solvable in polynomial time implies polynomial time solutions for *all* problems in NP-class.

In this assignment, the clique problem reduces to the vertex cover problem, and the vertex cover problem is used to reduce to set cover problem. The set cover problem has been shown to be reducible to **the hitting set problem.** In the hitting set problem, given a set A of some subsets of S, we determine the smallest subset of S that intersects every subset in A. This hitting set problem This hitting set problem in turn is proven an important problem in electrical engineering with multithreaded or distributed softwares (Wikipedia). It is fascinating how the observations/ discoveries on the **complexity of one problem builds up to another in a chain until it can get to a practical point of application**. I guess good things take time.