# Assignment 2

***Part II***

## Problem 1

Problem summary:

- Call problem *SUS-COAL*
- User, $u$
- Minute, $m$
- IP address, $I(u, m)$ or $\emptyset$ if no entry.
- Server attacked and attacker accessed $i$ distinct IP addresses over $t$ consecutive minutes. Therefore, in minute 1: attacker accessed address $i_1$, in minute 2: accessed address $i_2$, and so on until minute $k$.
- Need to analyse. No single user $u$ so that $I(u, m) = i_m$ for each minute m from 1 to $k$.
- Subset of users $S$ are suspicious if, for each minute $m$ from 1 to $t$, there is at least one user $u$ in $S$ for which $I(u, m) = i_m$.
- Given a collection of all values, and a number $l$, is there a suspicious coalition of size at most $l$?
- Prove NP-complete

To prove a problem is NP-complete, we have to prove that it is in NP and that every problem in NP can be reduced to the provided problem. We want to be able to prove that this problem is so hard that if we could solve it, we could also solve every other NP-complete problem.

We first show that this problem belongs to NP. Given some instance of $S$, $I$, $l$, and $u$, we can provide a certificate. To do so, we iterate over each minute, $m$, and if there is a user, $u$, such that $I(u, m) = i_m$, then we return *true*. However, if none is found we return *false*. This process can be done in polynomial time $O(l * t)$.

We will prove this problem is NP-complete by reducing it to VERTEX-COVER problem, a known NP-complete algorithm. Therefore, we want to prove VERTEX-COVER $\leq_P$ SUS-COAL.

We are given VERTEX-COVER instance which is a graph, $G = (V, E)$ and an integer $j$ representing the size of vertex cover. Given our graph, $G = (V, E)$, we can

- Set $S = V$
- $l$ = input $j$
- $| E | = t$ IP addresses
- We then map each $(u, u`) \subseteq E$ to a minute $m$
- $I(u, m) = i_m$

We now have $S$, $I$, and $l$ for SUS-COAL, and have reduced SUS-COAL to be a sub-problem of VERTEX-COVER.

We can now claim that our set, or coalition, $S$ has a size of at most $l$ if, and only if, G contains a vertex cover size of at most $l$. To prove this,

- Let $C \subseteq V$ be set to the vertex cover that has a size of maximum $l$. Therefore, we have $|C| \leq l$, and $C \subseteq S$. For each $i_m$, we have at least one user, $u \subseteq S$, that accessed address $i_m$ at minute $m$.
- Let $C \subseteq S$ be set to the suspicious coalition that has a size of maximum $l$. Therefore, we have $|C| \leq l$, and $C \subseteq V$. For each $i_m$, we have at least one user, $u \subseteq S$, that accessed address $i_m$ at minute $m$. Since $i_m$ is equal to an edge in $G$, we can say for each edge, $e \subseteq E$, a user, $u$, touches it. Therefore, $C$ is VERTEX-COVER.

# Problem 2

CELLPHONE-CAP = { $(G, k)$ : $G$ is a graph containing $k$ conversations }

We will define the capacity of a graph $G$, as $C(G)$. We want to know the largest number of conversations that can be taken simultaneously. We will represent this using the size of set $S$.

For both problems, we want to check its membership in NP. We know the set of conversations, $S$, is verifiable in polynomial time since we just check that $|S| \geq k$, $S \subset E$, and that no two edges in $S$ are neighbours.

## 3-SAT

To simply wish to prove 3-SAT $\leq_P$ CELLPHONE-CAP. To do so, we begin with some instance of 3-SAT. We let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ which is a Boolean formula in 3-CNF with $k$ clauses. For each clause, we then have exactly three distinct literals, namely $l`_1$, $l`_2$ and $l`_3$. We should then construct a graph such that $\phi$ is satisfiable if $G$ has $k$ simultaneous conversations.

$G = (V, E)$ is constructed as following:

- For each clause, $C_r = (l_1 \vee l_2 \vee l_3)$ in $\phi$, three vertices are put into $V$, namely $v_1$, $v_2$ and $v_3$.
- We then put an edge between them, if:
  - $v_i^r$ and $v_j^s$ are in different sets
  - $l_i^r$ is not the negation of $l_j^s$

We can then build this graph in polynomial time.

To prove this reduction works, we should note that exactly one of the edges in a clause can be in the set $V`$ if one or more of the adjacent edges in its literals are not in $V$. Therefore, we claim that $V`$ is a set of conversations with $k$ conversations.

If, and only if, 3-SAT is satisfiable, there are no adjacent edges between clauses and literals within $V`$. Therefore, we can set each clause and literal is set to 1. Each clause is satisfied, and so $\phi$ is satisfied.

Therefore, we know this is NP-complete.

## Vertex Cover

Provided an instance $(G = (V, E), k)$ of VERTEX-COVER. We can convert it to an instance of CELLPHONE-CAP problem, $(G` = (V`, E`), k`)$.

For each vertex in the set of vertices, $V$, we define a new vertex, $v`$. Therefore, we have a new graph such that every vertex in $G$ exists in $G`$. We then define each vertex, $v$, a partner vertex, $v`$. We define:

$$V` = V \cup \{ v` \mid v \subseteq V \}$$
$$E` = E \cup \{ (v, v`) \mid v \subseteq V \}$$

We can then set $k` = | V | - k$.

To proof the reduction works, we should note that the vertex cover complement is an independent set where $S$ consists of the edges connecting the vertices that are not in the vertex cover to their adjacent vertices. Therefore, if $G$ has a vertex cover of size $k$ or less, then $C(G`) \geq k`$.

If, and only if, $C(G`) \geq k`$: we can choose one vertex from the set $S$ that gives an independent set of minimum size $k` \rightarrow$ its complement is a vertex cover of size $k$ or less.

Therefore, we know this is NP-complete.

# Problem 3

***Part A***

$C^*_{max}$ represents the optimal schedule. Provided a number of processing times, such that $p_1, p_2, p_3, \ldots p_k$. Considering that $C^*_{max}$ is the most optimal schedule, it cannot be that we can have a more optimal schedule than the largest processing time.

Using the example provided, we have $C^*_{max}$ that is equal to 12, and processing times of $\{ p_1 = 2, p_2 = 12, p_3 = 4, p_4 = 5 \}$. The most optimal schedule is 12 which is also equal to our largest processing time. It simply cannot be that we have a processing smaller than any of the processing times considering we have to use these to make up the schedule.

***Part B***

$C^*_{max}$ represents the optimal schedule. Provided a number of processing times, such that $p_1, p_2, p_3, \ldots p_k$. Considering that $C^*_{max}$ is the most optimal schedule, it cannot be that we can have a more optimal schedule than the average processing time. It should be that the average of any set of numbers is always larger, or equal to, the largest number in the set.

Using the example provided, we have $C^*_{max}$ that is equal to 12, and processing times of $\{ p_1 = 2, p_2 = 12, p_3 = 4, p_4 = 5 \}$. The average for this set of processing times is 11.5, considering it was specified we have 2 machines. Therefore, for this set we know that $12 \geq 11.5$. However, consider the example where we have each processing time set to our optimal time 12 such that $\{ p_1 = 12, p_2 = 12, p_3 = 12, p_4 = 12 \}$. Therefore, even if we have four machines $\{ M_1, M_2, M_3, M_4 \}$ which would result in $C^*_{max}$ of 12 since one job will run on each schedule, we could not get a more optimal path than the average of these. So, even if we were to increase any of the values of each processing time our optimal schedule could not be larger than any of the processing times as proven in ***PART A***. Further, increasing the number of machines decreases the average since $m$ increases in $\frac{1}{m}$.

As average cannot be larger than any value in processing time, we know that $C^*_{max}$ must be larger than the average machine load.

***Part C***

The described greedy algorithm is: whenever a machine is idle, schedule any job that has not yet been scheduled. Using *J* to represent the set of jobs, and *m* to represent the number of machines. So, the pseudocode for this looks as following:

```
WHILE J NOT EMPTY:
|    MAX = 0
|    FOR i = 1 TO m
|    |   IF Mi == IDLE:
|    |   |    job = MAX(J)
|    |   |  schedule(job)
|    |   |  DEL job FROM J
|    |   END IF
|    END FOR
END WHILE
```

This greedy algorithm runs in $O(n^{\log(n)})$ since it will iterate through *n* jobs with each iteration of the while loop scheduling at least one job.

***Part D***

We want to show that the schedule returned by the greedy algorithm in ***PART C*** is less than, or equal to, the average of all processing times + the maximum processing time.

Since we proved that the average machine load (***PART B***) and the greatest processing time (***PART A***) are, at best, equal to the optimal schedule, we know that the combination of these could not be smaller than the make span of any given schedule.

Using the greedy algorithm from ***PART C***, we can determine that $M_1$ would get { 12 } with $M_2$ getting { 5, 4, 2 }. The total running time between these is 12.

Now, using the provided example, we have processing times of { $p_1 = 2$, $p_2 = 12$, $p_3 = 4$, $p_4 = 5$ }. We know from our previous parts, that the largest processing time is 12, and the average processing time is 11.5. Therefore, we can confirm that this satisfies the equation

$$C_{max} \leq \frac{1}{m} \sum_{1 \leq k \leq n} p_k + max_{1 \leq k \leq n} p_k$$

Since $12 \leq 12 + 11.5$, or $12 \leq 23.5$, is true. We then know that the RHS is the upper bound of the on the make span returned. Therefore, we can use this to calculate the ratio $\frac{23.5}{12}$, or 1.958. Since this is less than 2 but greater than 1, we can confirm this to be a 2-approximation algorithm.