# Macquarie University
# COMP333 Algorithm Theory and Design
# Assignment 2 Part II

October 24, 2019

Note: This is a draft version of the assignment, but a changelog will be provided at the end of the document to note any future changes.

**Problem 1.** (4 points total)

Consider the you are designing a software to automate the analysis and detection of attacks on a company or bank server. For analysis and detection you need to have *logs* of users activities accessing the server. Suppose that the software records the IP addresses that users access on the server. Suppose that each user accesses at most one IP address in any given minute; the software writes a log file that records, for each user $u$ and each minute $m$, a value $I(u, m)$ that is equal to the IP address (if any) accessed by user $u$ during minute $m$. (It writes a null symbol, say $\emptyset$, if there is no such access).

Suppose the server was attacked yesterday and the attack was carried out by accessing $i$ distinct IP addresses over $t$ consecutive minutes: In minute 1, the attack accessed address $i_1$; in minute 2, the attack accessed address $i_2$; and so on, to address $i_k$ in minute $k$.

To detect the attacks, you are required to carry out a systematic analysis of the logs. While checking the logs, it turns out that there is no single user

$u$ who accessed each of the IP addresses involved at the appropriate time; in other words, there's no $u$ so that $I(u, m) = i_m$ for each minute $m$ from 1 to $k$. So the question becomes: what if there were a small *coalition* of $k$ users that collectively carried out the attack? We will say that a subset $S$ of users is a *suspicious coalition* if, for each minute $m$ from 1 to $t$, there is at least one user $u \in S$ for which $I(u, m) = i_m$. The *Suspicious Coalition Problem* asks: Given the collection of all values $I(u, m)$, and a number $l$, is there a *suspicious coalition* of size at most $l$? Prove that this problem is NP-complete.

**Problem 2.** (6 points total)

Consider the following model of cellphone conversations. There is an undirected graph $G = (V, E)$, where the vertices are people, and each edge indicates that two people are within range of each other so that they can potentially have a conversation. However, while two people are talking, their neighbors must stay silent (on that frequency) to avoid interference. Thus a set of conversations consists of a set of edges $S \subset E$, where vertices in different edges in $S$ cannot be neighbors of each other. Formally:

$$(u, v), (w, t) \in S \Rightarrow (u, w) \notin E \ .$$

The *cellphone capacity* of $G$, denoted $C(G)$, is the largest number of conversations that can take place simultaneously on one frequency, i.e., the size of the largest such set $S$. For instance, if $G$ is a cube (with 8 vertices and 12 edges) its cellphone capacity is 2. Now consider the following problem:

Cellphone Capacity
Input: a graph $G$ and an integer $k$
Question: is $C(G) \geq k$?

Prove that Cellphone Capacity is NP-complete using both 3-SAT and Vertex Cover (*each worth 3 points*).

**Problem 3.** (10 points total)

Consider you are working at large datacenter and as engineer you are required to run $n$ jobs on $m$ identical computing machines. More specifically, you are given $n$ jobs (such as $J_1$, $J_2$, $\cdots$, $J_n$) where each job $J_k$ requires $p_k$ processing time for completion. Each of your identical computing machines ($M_1$, $M_2$, $\cdots$, $M_m$) can any job. A schedule specifies, for each job $J_k$, the machine on which it runs and the time period during which it runs. Each job $J_k$ must run on some machine $M_i$ for $p_k$ consecutive time units, and during that time period no other job may run on $M_i$. Let $C_k$ denote the completion time of job $J_k$, that is, the time at which job $J_k$ completes processing. Given a schedule, let's define $C_{max} = max_{1 \leq j \leq n}\ C_j$ to be the makespan of the schedule. Your goal is to find a schedule whose makespan is *minimum*.

To elaborate this problem, let's suppose that you have two machines $M_1$ and $M_2$ and that we have four jobs $J_1$, $J_2$, $J_3$, and $J_4$ with processing times, respectively, as $p_1 = 2$, $p_2 = 12$, $p_3 = 4$, and $p_4 = 5$. Then one possible schedule runs, on machine $M_1$, job $J_1$ followed by job $J_2$, and on machine $M_2$, it runs job $J_4$ followed by job $J_3$. For this schedule, $C_1 = 2$, $C_2 = 14$, $C_3 = 9$, $C_4 = 5$, and $C_{max} = 14$. An *optimal schedule* runs $J_2$ on machine $M_1$, and it runs jobs $J_1$, $J_3$, and $J_4$ on machine $M_2$. For this schedule, $C_1 = 2$, $C_2 = 12$, $C_3 = 6$,$C_4$ =11, and $C_{max} = 12$.

Given a this problem, we let $C^*_{max}$ denote the makespan max of an *optimal schedule*.

a) (1.5 points) Show that the optimal makespan is at least as large as the greatest processing time, that is

$$C^*_{max} \geq max_{1 \leq k \leq n}\ p_k$$

b) (1.5 points) Show that the optimal makespan is at least as large as the average machine load, that is

$$C^*_{max} \geq \frac{1}{m} \sum_{1 \leq k \leq n} p_k$$

For improved scheduling you are required to resort to some greedy algorithm: whenever a machine is idle, schedule any job that has not yet been scheduled.

c) (3 points) Write pseudocode to implement this greedy algorithm. What is the running time of your algorithm?

d) (4 points) For the schedule returned by the greedy algorithm, show that

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

And finally, conclude that this algorithm is a polynomial-time 2-approximation algorithm.