Covid Project

Ruiming Li, Rahul Palniktatr, Jiangxue Han, Isaac Chang

September 18, 2020

1 Covid-19 Class Scheduling

1.1 Design

At the beginning, we assume all classes are in person, so students are connected directly to people in the same class. We represent each student as a node and each class as a labeled edge in a graph G. Two students can be connected by an edge labeled with multiple classes such as A, B if they take both classes together.

All students are connected within each component C in G, and are completely disconnected across components. This means that if one student is taking both class A and B, all students in class A and B are connected. If there are no students taking both class A and B, and there are no students from class A and B meeting in any class D, A and B are in two distinct components.

Hence, let t_k be the number of connected students in the k^{th} component and the number of professors teaching classes in that component. We assume all t_k people are equally susceptible to COVID, so given a probability of infection p, days of infectivity d, and length of semester t, we can use the SIR formula to calculate the maximum number of infected students within a component k when one student in component k gets infected.

Since every component is disconnected from each other, infection in one component is contained in that component. Within a component, the number of infections increases exponentially with the number of students, so the ideal situation is to have many small components in G so one infection in a component will not exceed the limit imposed and keep students and professors safe.

More precisely, the expected maximum size of the infected population I in graph G at is a weighted sum of the expected maximum infected population across all k components:

$$E(max(I) \mid G) = \sum_{k=1}^{m} \frac{t_k}{T} I_k^*$$

where $T = \sum_{k=1}^{m} t_k$ is the total number of students and professors in in person classes and I_k^* is the maximum number of people infected during the SIR model simulation for component k with model population of t_k, infectious rate of p, recovery rate of $\frac{1}{d}$ and time of t.

Our algorithm runs as below: while there exists I_k^* greater than n, the maximum acceptable number of infections at any point, we find the largest component C in graph G. We partition component C into two sub-components C_1, C_2 such that the number of cutting edges between C_1, C_2 is minimal. We make all classes labeled on these cutting edges online, and update G by removing the cutting edges. We will repeatedly find the next largest component in the updated G and repeat the same process until the expected value constraint is lower than n.

1.2 Pseudocode

Function `getSchedule(`*G, studentClasses, C, p, d, t, M*`)`

```
G := graph of students
studentClasses := classes each student is taking
C := classes
p := probability of infecting
d := time to heal from covid in days
t := length of semester in days
M := maximum number of infected at any point

schedule = {}
for every c in classes do
    schedule[c] = 'In Person'
end
r = 1/d
components = getComponents(G)
while expectedMaxInfect(studentClasses, components, p, r, t) > M do
    biggest = components.extractMax()
    newComps = makeOnline(G, biggest, classes)
    for comp in newComps do
        components.insert(comp)
    end
end
return schedule
```

Function `expectedMaxInfect(`*components, p, r, t*`)`

```
e, T = 0, 0
for comp in components do
    k = number of students/professors in this component
    T += k
    e += k·(maxInfected(k, p, r, t))
end
return e/T
```

```
Function getComponents(G)
    components = [ ]
    visited = {}
    for vertex in G do
        visited[vertex] = False
    end
    start = some vertex in G
    dfs(G, start, visited, components)
    return components

Function dfs(G, start, visited, components)
    s = stack()
    s.push(start)
    while s is not empty do
        v = s.pop()
        visited[v] = True
        for each of v's unvisited neighbors do
            s.push(neighbor)
        end
    end
    comp = {}
    if all vertices are visited then
        for vertex in visited do
            comp.add(vertex)
        end
        components.insert(comp)
    end
    else
        for visited vertex in visited do
            comp.add(vertex)
            visited.remove(vertex)
        end
        if comp is not empty then
            components.insert(comp)
        end
        newStart = some unvisited vertex
        dfs(G, start, visited, components)
    end
```

2 Time Analysis for getSchedule

2.0.1 Data Structures

G is a graph where the students are vertices and classes are edges. If two students take a class c, Then they are connected by and edge labeled c.

We store the graph in an adjacency list where the keys are the student names and the values are a list of tuples. Each tuple with have the name of a neighbor student as the 0th element and a set of classes the neighbor student has in common.

For example, the adjacency list can look something like:
{
$s_0 : [(s_i, \{A, B, C\}), ..., (s_j, \{A, B\})]$
.
.
.
$s_{m-1} : [(s_s, \{A, B\}), ..., (s_t, \{A, B\})]$
}

We store the classes each student takes in a dictionary where the key is the student name and the key is a set of names of classes that they are taking.

We store the classes in a list where each element is the class name.

We store the components of the graph in a max heap where the heap property is the number of people in a component.

2.0.2 Time complexity

Let n be the number of students, m be the number of classes, k be the average number of vertices in a component.

The algorithm first requires setting each class to be in person. Since there are m classes and we have the classes in a list input, this can be done in $O(m)$.

Next, the algorithm requires getting the components of G. Since we uses the traditional depth first search algorithm to find the components, this can be done in $O(m + n)$.

Then, for the while loop, each iteration requires calculating the expected maximum size of the infected population (invariant). This requires finding the expected maximum size of the infected population for each of the components, C_i. For each C_i, we can find I_k^* in constant time finding the maximum value of the infected population model in the SIR model for C_i can be done with non-iterative mathematical calculations. But, we need to find the appropriate weight for I_k^*. We must find the number of people in C_i. The number of students in C_i is simply the number of vertices in C_i which can be found in

$O(1)$. The number of professors in C_i is the number of edges with distinct labels because each professor teaches some class to many students. We can look at every student in the C_i and keep track of the distinct classes with a set. Since a component has average size of k students, this can be done in $O(k)$ by using the studentClasses dictionary to figure out what class each student is taking in C_i. Since we find the number of students/professors in every component, finding the total number of students and professors attending in person classes follows. Since there are at most n components (n students), the expected maximum size of the infected population for the current ordering. can be calculated in $O(nk)$.

Inside the while loop, each iteration requires

1. getting the component with the largest number of students and professors
2. split the largest component minimally into two smaller components

We store the components in a max heap based on the number of people in each component. At most, there can be n components because there are n students. So, extracting the component with the largest population can be done in $O(\log n)$.

We will use Karger's minimal cut algorithm for cutting the largest component by making the fewest number of classes online. This algorithm can be done in $O(k^2 + \log^2 k)$

So, the while loop can be done in

$$O(n(nk + \log n + k^2 \log^2 k))$$

$$= O(n^2 k + nk^2 + n \log^2 k)$$

$$= O(n^2 k)$$

So, the whole algorithm can be done in

$$O(m + m + n + n^2 k)$$

$$= O(m + n^2 k)$$

2.1 Discussion

We chose this algorithm because it made sense to think about students being connected by an edge if they attend a same in person class. We chose this algorithm because we wanted to start with the maximum number of in person classes and choose the best classes to make online step by step.

These are two of the initial problems we faced:
We weren't sure how to choose which classes were best to make online, but we observed that students often are grouped in large components (major classes for example) and wanted

to find a way to split components that were too big because of Covid. One of the bigger issues we encountered was factioring in how time affected Covid spread. We tried to make metrics that used the algorithm parameters to give a score to a scheduling, but we realized that how the SIR model changes throughout the semester is important because our goal is to make sure each day does not exceed the Covid limit.

To involve time into our metric for how good a schedule is, we looked at the differential equations for the SIR model and saw how it models the expected growth of each state. Specifically, we looked at the infected population in the SIR model to for a metric because we care most about the number of infected people per day.

A big complication we encountered was finding which classes to make online to cut a big component into two smaller ones. We got an idea from cutsets of a graph because we want to find the sets of edges that connect halves of a component. We also looked at algorithms pertainig to minimal cuts for a graph and came across Karger's algorithm. We still are not fully set on how to exactly split a big component yet.

This problem is hard because the population is large and there are many ways and factors that affect how people get infected. To make it more simple, we tried to simplify how we use inputs to find the schedule. For example we didn't consider at all what time of the day a class takes place because we thought that it would add too much complexity to the problem. But, chosing what parameters to not involve was a challenge itself because each parameter is importart for how Covid spreads and that is ultimately what we have to look at. We had to look at the tradeoffs between time and accuracy of our answer.

Our algorithm seems to be most similar to a greedy algorithm because if we need to cut the largest componenet, we have to find the best set of classes to make online in that component to cut it.

3 Sources

`https://en.wikipedia.org/wiki/Karger%27s_algorithm`