American Journal of Computational Mathematics, 2021, 11, 1-22
https://www.scirp.org/journal/ajcm
ISSN Online: 2161-1211
ISSN Print: 2161-1203

Mathematical Models for a Social Partitioning Problem

Vardges Melkonian

Department of Mathematics, Ohio University, Athens, OH, USA
Email: melkonia@ohio.edu

How to cite this paper: Melkonian, V. (2021) Mathematical Models for a Social Partitioning Problem. *American Journal of Computational Mathematics*, **11**, 1-22.
https://doi.org/10.4236/ajcm.2021.111001

Received: October 26, 2020
Accepted: February 26, 2021
Published: March 2, 2021

Abstract

In this paper we develop modeling techniques for a social partitioning problem. Different social interaction regulations are imposed during pandemics to prevent the spread of diseases. We suggest partitioning a set of company employees as an effective way to curb the spread, and use integer programming techniques to model it. The goal of the model is to maximize the number of direct interactions between employees who are essential for company's work subject to the constraint that all employees should be partitioned into components of no more than a certain size implied by the regulations. Then we further develop the basic model to take into account different restrictions and provisions. We also give heuristics for solving the problem. Our computational results include sensitivity analysis on some of the models and analysis of the heuristic performance.

Keywords

Health Care Operations Research, Mathematical Models for Pandemics, Graph Partitioning, Integer Linear Programming, Heuristic Algorithms

1. Introduction

Background and Problem Statement

To fight the Covid-19 pandemic, measures restricting social gatherings were taken by many countries and states. Many restrictions set limits on the number of people allowed in gatherings. For example, Germany and UK at some point limited social gatherings of more than 2 people [1] while in many US states that number was 10 [2]. While that type of restrictions might help to prevent the spread of the disease they have the following shortcoming. A sick but asymptomatic person could attend several meetings of no more than 10 people (different people in each meeting). People from those meetings could get the virus and attend other meetings of no more than 10 people. It could create a chain reaction which will result in many people getting infected.

In this paper we suggest an alternative approach of limiting interactions to prevent the spread of the disease. It is explained on the example of regulating the interactions of a small or medium size company since our approach is most suitable for that type of organizations.

Consider a company that has N employees who are supposed to have direct interactions with each other in everyday operations of the company. Each direct interaction can potentially spread a disease from one person to others. To curb the spread of the disease we suggest partitioning the set of N employees into smaller groups of at most M people each. (Number M can be determined based on state or company regulations.) Employees can have direct interactions only with people of the same smaller group. In that case, a person who has the disease can potentially infect at most $M - 1$ people within the same group. Thus, the effect of chain reaction is limited.

The set of N employees can be partitioned into smaller groups of at most M people in many different ways. What is the best way of partitioning? While limiting the spread of the disease during a pandemic is a primary goal, another important goal is to keep as much economic activity as possible. Economic activity can be promoted by not interrupting essential employee interactions within the company. In the context of our problem, the goal is the following: find a partitioning into groups of at most M people that maximizes the number of uninterrupted employee interactions.

The corresponding discrete optimization problem is the following. We have a graph G with N nodes (employees). The goal is to remove as few arcs (employee interactions) as possible such that the graph is partitioned into connected components with at most M nodes each. In other words, the number of arcs that remain in the graph should be maximized.

An example with $N = 11$ employees and $M = 4$ maximum component size is given in **Figure 1**. The ten bold arcs form an optimal solution for the example. The components in the optimal solution are {1, 2, 3, 4}, {5, 7, 8}, {6, 9, 10, 11}.

Some of the employee interactions might be more important for the company than others. Thus, it makes sense to assign weights to the arcs, with higher weights assigned to more important arcs. Then the goal is to maximize the total weight of the arcs included in the solution.

Solution methods

We give an integer linear programming model for solving the problem. The main feature of the model is a set of constraints providing that two nodes (employees) are in the same component. We give several techniques for making the model computationally more efficient. The techniques include reducing the number of variables and relaxing the integrality requirements of the largest set of variables. We also give a heuristic method for solving the problem. The heuristic output can also serve as an initial solution for the integer programming model to accelerate the solution process.

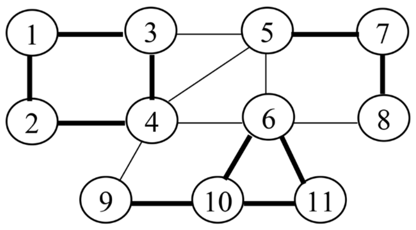

Figure 1. An example of social partitioning problem.

Variations of the main model

While the main requirement of the Social Partitioning Problem is not exceeding the maximum component size, in real life situations there might be other restrictions and provisions that should be taken into account. For example, some employees might be required to have direct interaction in the company; some specialists doing the same important task for the company might be required to be in different components; sometimes the maximum component size might be violated if it is very important for company's work but in that case a penalty is paid for the violation. We give variations of our main integer programming model to take into account each of the above-mentioned restrictions and provisions. For each of those cases we also give an extension to the heuristic.

Connection to other graph partitioning problems

Graph partitioning problems were studied extensively before. [3] gives a survey for algorithms and applications of balanced graph partitioning problems. There are a variety of solution methods for solving the problems, ranging from simple heuristics to sophisticated combinatorial optimization methods. Some well-known applications are parallel processing, road networks, image processing. [4] gives an approximation algorithm for a balanced partitioning problem which is the closest to our problem. In most of the related problems including [4], the number of components is fixed. In our problem, the number of components is not fixed which makes it harder to solve. The main characteristic of our problem is the maximum size of components. This characteristic comes from the nature of the main social partitioning application of our problem. It is also the main distinction from related problems that use different criteria for partitioning. Unlike many of the related problems, we consider the weighted version of the problem because assigning different weights to the arcs is more realistic in the context of the Social Partitioning problem. Our main solution method is integer programming while approximation algorithms or heuristics are used for many related problems. Integer programming is a suitable method for small companies (up to 50 people) when solving the Social Partitioning problem. We also give directions how to make the model more efficient so that it could be applied to bigger problems. Integer programming is also a suitable technique for adding different extra requirements to the original problem to make it more realistic in the context of the social partitioning application.

Computational results

We implemented our models on Optimization Modeling language AMPL (A Mathematical Programming Language) [5]. The models were tested for inputs of different sizes and nature. Sensitivity analysis was done for the models that allow penalty for exceeding the maximum component size. The heuristic was also implemented and tested on AMPL, and its performance was evaluated with respect to the optimal solution.

The structure of the paper

The paper is structured as follows. Section 2 gives the main integer programming model, directions for improving its efficiency and its AMPL implementation. Section 3 discusses different variations and extensions of the model. Section 4 gives heuristics for solving the main problem and its variations. Section 5 discusses computational results. Section 6 gives several future directions.

2. The Main Model

We develop the basic model in Section 2.1. In Section 2.2, we discuss how the model can be refined to make it computationally more efficient. Section 2.3 gives the AMPL implementation of the model.

2.1. The Basic Model

Let **Nodes** be a set of N employees within a company. Let **Arcs** be a set of all possible direct interactions (links) between pairs of employees in Nodes. The regulations require that Nodes should be partitioned into components such that each component has at most M employees. No interactions are allowed between employees in different components. Each direct interaction has some value for the company. Thus, the company wants to maximize the number of possible interactions while complying with the regulations.

Variables.

We have the following set of binary variables.

Let x_{ij} be 1 if the link between employees i and j is included in the solution, and 0 otherwise.

Let y_{ij} be 1 if employees i and j are in the same component in the solution, and 0 otherwise.

Weighted versus unweighted version of the problem

In the unweighted version of the problem, the company wants to maximize the total number of links included in the solution. But in reality some links are more essential and important for the company than others. Thus, we define a parameter **weight** in the range [0, 1] for each link of the set Arcs. A higher weight implies that the link has higher importance for the company. For the rest of the paper we consider the weighted version of the problem. Note that the unweighted version is the special case when all the weights are set to 1.

Objective function

The objective function maximizes the total weight of the arcs included in the solution.

$$\max \sum_{(i,j)\in \text{Arcs}} \text{weight}(i,j) * x_{ij} \tag{1}$$

Constraints

We have the following constraints in the model.

C1) The size of any component should be no more than M. To provide it we require that for any node i, the number of nodes that are in the same component with i is no more than $M - 1$.

$$\sum_{j\in \text{Nodes}} y_{ij} \le M - 1 \tag{2}$$

Variables x_{ij} and y_{ij} should be connected with each other. We need the following two sets of constraints to provide it.

C2) If link (i, j) is included in the solution then nodes i and j should be in the same component.

Symbolically, if $x_{ij} = 1$ then $y_{ij} = 1$. The corresponding linear constraint is

$$y_{ij} \ge x_{ij} \tag{3}$$

C3) Transitivity. If nodes i and j are in the same component, and nodes j and k are in the same component, then nodes i and k also should be in the same component.

Symbolically, if $y_{ij} = 1$ and $y_{jk} = 1$ then $y_{ik} = 1$. Equivalently, if $y_{ij} + y_{jk} = 2$ then $y_{ik} = 1$.

The linear constraint that realizes the conditional constraint above is

$$y_{ik} \ge y_{ij} + y_{jk} - 1 \tag{4}$$

The constraint works the following way. If $y_{ij} + y_{jk} = 2$ then the right-hand side is 1, and y_{ik} is required to be at least 1; since y_{ik} is a binary variable it will be forced to be exactly 1. On the other hand, if $y_{ij} + y_{jk} \le 1$ then the right-hand side is ≤ 0; thus, it is not forcing anything on y_{ik}.

2.2. A Computationally Refined Version of the Basic Model

In this section, we give several directions on how to refine the model to make it computationally more efficient. The main ideas are reducing the number of y_{ij} variables and relaxing the integrality of y_{ij} variables.

Reducing number of variables by defining symmetric variables only once

In our model, the nodes are given in lexicographic order: Nodes = 1, …, N. For every pair of nodes i and j such that $i < j$, we define only one link (i, j) and corresponding variables x_{ij} and y_{ij}. Since the relationship between i and j is symmetric the opposite link is implicitly in the solution too. This approach significantly reduces the number of variables.

Reducing number of variables by not defining the variables that cannot be

1

Note that in the input of the problem, the nodes might already be partitioned into several connected components. Our model intends to divide those components into even smaller parts not exceeding size M. We will distinguish those components, by calling them *input components* versus *output components*. For example, consider the unweighted network in Figure 2 with $N = 10$ and $M = 3$. It has two input components: {1, 2, 3, 6, 7} and {4, 5, 8, 9, 10}. The output components in an optimal solution would be {1, 2}, {3, 6, 7}, {4, 5}, {8, 9, 10} with a total of 6 arcs included in the solution.

If nodes i and j are not in the same input component then i and j also cannot be in the same output component, and variable y_{ij} cannot take value 1 in the optimal solution. Thus, there is no point of defining the corresponding variable y_{ij}. For example, there is no need to define variable $y_{3,8}$ in the instance of Figure 2 since nodes 3 and 8 are in different input components. This especially makes sense for relatively sparse networks.

Another consideration for excluding variable y_{ij} is the following. Suppose nodes i and j are in the same input component but the shortest distance between i and j is greater than $M - 1$. If i and j were also in the same output component then that component would include a path with $M + 1$ nodes connecting i and j (including i and j). But an output component cannot have more than M nodes. Thus, i and j cannot be in the same output component, and there is no point of defining the corresponding variable y_{ij}. For example, there is no need to define variable $y_{1,6}$ in the instance of Figure 2 since the distance between nodes 1 and 6 is 3 which is greater than $2 = M - 1$; having 1 and 6 in the same output component would imply that all the nodes on the connecting path 1-2-7-6 should be in that output component, thus violating $M = 3$.

To realize the reduction of variables discussed above, we find the input component of each node using breadth-first search (BFS). The information from BFS computations is used to compute the shortest distance between any two nodes from the same input component. Then variable y_{ij} is defined only if 1) i and j are in the same input component; 2) the shortest distance between i and j is no more than $M - 1$. The AMPL implementation of these computations is given in Section 2.3.

Relaxing the integrality of y_{ij} variables

Theorem 1. Suppose the binary requirements of y_{ij} variables are relaxed to $0 \le y_{ij} \le 1$. Then there will be an optimal solution where y_{ij} will take only values 0 or 1.

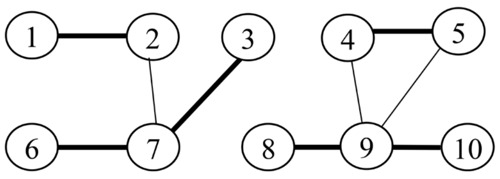

Figure 2. Example of Input and Output Components.

Proof

Let $C_1, C_2, \cdots, C_m$ be the output components formed by including arcs (*i*, *j*) with $x_{ij} = 1$ in the solution. Suppose nodes *u* and *v* are in the same component. Then there is a path *P* between *u* and *v* such that for all the arcs (*i*, *j*) on *P*, $x_{ij} = 1$. Constraint (C2) $y_{ij} \geq x_{ij}$ implies that for all those arcs on *P*, y_{ij} is also 1. Now recall the Transitivity constraint (C3): $y_{ik} + 1 \geq y_{ij} + y_{jk}$. If y_{ij} and y_{jk} both are 1 then the constraint will force y_{ik} also to be 1. Applying (C3) repeatedly for the arcs on path *P*, we will get that $y_{uv} = 1$.

Summarizing, if nodes *u* and *v* are in the same output component then $y_{uv} = 1$. The only remaining question is the following. Suppose nodes *u* and *v* are in different output components C_u and C_v. Is it possible that $y_{uv} > 0$ in the optimal solution?

Note that $y_{uv} > 0$ cannot improve the objective function value since the objective function has only x_{ij} variables. Having $y_{uv} > 0$ will make all y_{ij} variables between C_u and C_v positive (based on transitivity constraint (C3)), thus making the constraint (C1) for any node from C_u and C_v less likely to be satisfied. Even if some of those y_{uv} variables are positive they will not affect optimal x_{ij} variables and the optimal value. In other words, there might be multiple optimal solutions, and even if some of them could have fractional y_{ij} variables there is always an optimal solution with no fractional y_{ij} variables. Thus, by postprocessing the solution based on x_{ij} variables one can get rid of the fractional values by setting them to 0 and get the actual output components.

□

To illustrate the result of Theorem 1, consider the example in **Figure 3**. The numbers on the arcs are the arc weights, and the maximum component size is $M = 4$. When the integrality of y_{ij} variables is relaxed, the model returns the following optimal solution. $x_{1,2} = x_{1,3} = x_{2,3} = x_{4,5} = 1$ which means arcs (1, 2), (1, 3), (2, 3), (4, 5) are included in the solution (are bold in the figure), creating components {1, 2, 3} and {4, 5}. The optimal value is $4 \times 0.9 = 3.6$. Also, $y_{1,2} = y_{1,3} = y_{2,3} = y_{4,5} = 1$ indicating the connectivity of corresponding node pairs. But the model also returned positive fractional values for other node pairs: $y_{1,4} = y_{1,5} = y_{2,4} = y_{2,5} = y_{3,4} = y_{3,5} = 0.5$. Note that these fractional values make the (C1) constraints for nodes 1, 2, and 3 satisfied at equality: $2 \times 1 + 2 \times 0.5 = 3$. Thus, the corresponding optimal solution is likely to be a basic solution returned by the solver. But we can obtain another optimal solution by setting all fractional

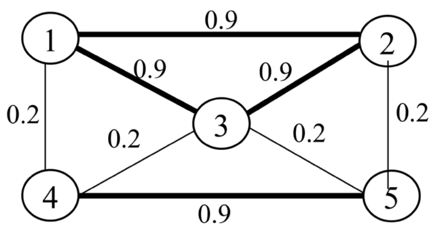

Figure 3. Example with fractional solution.

y_{ij} values to 0: $y_{1,4} = y_{1,5} = y_{2,4} = y_{2,5} = y_{3,4} = y_{3,5} = 0$. As argued above, all the constraints still will be satisfied, and the objective function value will stay at the optimal value 3.6 since the values of x_{ij} variables are not changed. Thus, this new solution without any fractional values is also optimal.

2.3. The AMPL Implementation

The AMPL implementation of the basic model is given below.

```
   ######################## PARAMETERS and SETS
##############################
   param N;
   set Nodes: = 1,...,N;
   param M;
   # maximum component size
   ############# ARCS WITH WEIGHTS ###############
   set Arcs within {i in Nodes, j in Nodes: i<j};
   # set of original input arcs; can have just one direction
   param weight{(i,j) in Arcs};
   ######### INPUT COMPONENTS AND NODE DISTANCES ##########
   set BFS{i in Nodes, k in 1,...,N-1} : =
   if k = = 1 then {j in Nodes: (i,j) in Arcs or (j,i) in Arcs} else
   {j in Nodes: exists {l in BFS[i,k-1]} ( (l,j) in Arcs or (j,l) in Arcs ) } ;
   # the set of nodes that have a path with length k from node i
   set component{i in Nodes} : = union{k in 1,...,N-1} BFS[i,k];
   # the set of all nodes that are in the same input component with node i
   param distance{i in Nodes, j in Nodes: i! = j and j in component[i]} : = min{k
in 1,...,N-1: j in BFS[i,k]}k;
   # shortest distance between nodes i and j
   ############################# VARIABLES
###########################
   var incl{(i,j) in Arcs} binary;
   # is 1 if the arc is included in the solution
   var connected{i in Nodes, j in Nodes: i < j and j in component[i] and
distance[i,j] < = M-1} binary;
   # is 1 if nodes i and j are connected in the solution
   # if the distance is > = M then cannot be in the same selected component, thus
no need to define a variable
   ######################## OBJECTIVE FUNCTION
##############################
   maximize Total_weight_of_included_arcs: sum{(i,j) in Arcs}weight[i,j]*incl[i,j];
   # the goal is to maximize the total weight of the arcs
   ###################### CONSTRAINTS ######################
   ############ Max_component_size ############
   s.t. Max_component_size{i in Nodes}:
```

```
sum{j in Nodes: i<j and j in component[i] and distance[i,j] < = M-1}
connected[i,j] +
sum{j in Nodes: i>j and i in component[j] and distance[j,i] < = M-1}
connected[j,i]
< = M − 1;
# each node can be connected to at most M-1 other nodes, making its
component size at most M
######## Included implies Connected #########
s.t. Included_is_Connected{(i,j) in Arcs: j in component[i] and distance[i,j] <
= M-1}: connected[i,j] > = incl[i,j] ;
# if arc i-j is included in the solution then nodes i and j are connected
############ Transitivity ############
s.t. Transitivity{i in Nodes, j in Nodes, k in Nodes: i! = j and i! = k and j! = k
and j in component[i] and k in component[j] and k in component[i]
and distance[i,k] < = M-1 and distance[i,j] < = M-1 and distance[j,k] < = M-1
and distance[i,k] < = distance[i,j] + distance[j,k] }:
(if i<k then connected[i,k] else connected[k,i]) + 1
> = (if i<j then connected[i,j] else connected[j,i]) +
(if j<k then connected[j,k] else connected[k,j]);
# if (i is connected to j) and (j is connected to k) then (i is connected to k).
```

3. Variations of the Model

In real-life situations there might be other requirements that should be taken into account when building the model. Below we list several possible restrictions and provisions and then show how to incorporate each of them in the model.

1) Some links might be very important for company's work and cannot be removed.

2) The company might want to have several employees (specialized in the same type of important task) in different components.

3) Sometimes the maximum component size might be exceeded if it is very important for company's work. But in that case some penalty must be paid.

Restrictions 1, 2, and 3 are discussed in Sections 3.1, 3.2, and 3.3 correspondingly.

3.1. Models with Required Connections

3.1.1. Model with Required Arcs

A natural requirement is that some links are so essential to company's work that they cannot be removed.

This restriction is not hard to model. But the problem is that the required arcs might not allow to have the maximum component size restriction to be satisfied. Even if the number of required connections for each node is no more than M, transitivity might force to have more than M nodes in some components, thus making the problem infeasible. For example, consider the unweighted network of Figure 1 with $N = 11$, $M = 3$. Suppose arcs (3, 5), (4, 5), and (4, 6) are required to be in the solution. Then based on transitivity, nodes 3, 4, 5, 6 will be in the same component, thus violating the maximum component size 3.

We suggest several ways of dealing with the problem.

1) Instead of requiring the arcs to be in the solution give those arcs the highest possible weight in the weighted version of the problem.

2) Pay penalty for exceeding M. This solution idea is discussed in Section 3.3.

3) Have required interaction between subsets instead of nodes. This method is discussed in Section 3.1.2 below.

3.1.2. Model with Required Interactions between Subsets

The strict requirement of having a direct interaction between two employees might be too restrictive and has a good chance of making the problem infeasible. A relaxed version of the requirement could be the following. Given two disjoint subsets E1 and E2 of employees it is required to have at least one direct interaction between an employee from E1 and an employee from E2. Note that when each of the two subsets has only one element then it becomes the original restriction of Section 3.1.1. But the subset requirement is less restrictive, especially if the subset sizes are relatively big, and it could be a realistic restriction in many situations.

To illustrate the idea, we reconsider the example of Figure 1 ($N = 11$, $M = 4$, unweighted arcs) with the following modification: it is required to have at least one arc between subsets $\mathrm{E1} = \{1, 2, 3, 4\}$ and $\mathrm{E2} = \{5, 6, 7, 8\}$. The original optimal solution with 10 arcs given in Figure 1 does not satisfy this new requirement. A new optimal solution which includes 9 arcs is given in Figure 4. The arcs included in the solution are in bold, and they create the following four components: {1, 2}, {3, 4, 5}, {7, 8}, {6, 9, 10, 11}.

The requirement is modelled the following way. Let R be the number of all required subset interactions. For each r in 1, …, R, there is a pair of disjoint subsets E1 and E2 such that there should be at least one arc between the two subsets in the solution.

$$\sum_{i \in \mathrm{E1},\, j \in \mathrm{E2}} x_{ij} \geq 1$$

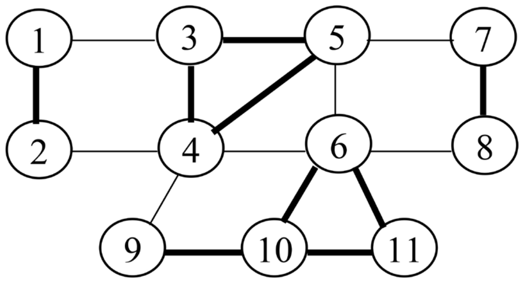

Figure 4. Example with Subset Interaction Requirement.

The complete AMPL implementation of the required subset interactions is given below.

```
############# REQUIRED SUBSET INTERACTIONS
####################
param R;
# number of required interactions
set empl_subset1{r in 1,...,R} within Nodes;
# the employee subset 1 in interaction r
set empl_subset2{r in 1,...,R} within Nodes;
# the employee subset 2 in interaction r
check {r in 1,...,R, j in Nodes}: if j in empl_subset1[r] then j not in
empl_subset2[r];
# check that empl_subset1[r] and empl_subset2[r] are disjoint
check {r in 1,...,R}: exists {i in empl_subset1[r], j in empl_subset2[r]} ( (i,j) in
Arcs or (j,i) in Arcs );
# check that there is at least one arc between empl_subset1[r] and
empl_subset2[r]
s.t. Required_interaction{r in 1,...,R}:
sum{i in empl_subset1[r], j in empl_subset2[r]: (i,j) in Arcs}incl[i,j] +
sum{i in empl_subset2[r], j in empl_subset1[r]: (i,j) in Arcs}incl[i,j]
> = 1;
# there should be at least one interaction between subsets empl_subset1[r] and
empl_subset2[r].
```

3.2. Model with Restricted Connections

If there is a group of employees specialized to do the same kind of important task for the company, then the company might want to make sure that they do not get sick at the same time. Thus, those employees should be in different components. More specifically, we require that there should be at least two different output components having employees from the group.

To illustrate the idea, we reconsider the example of Figure 1 ($N = 11$, $M = 4$, unweighted arcs) with the following modification: employees 3 and 4 should be in different components. The original optimal solution with 10 arcs given in Figure 1 does not satisfy this new requirement. A new optimal solution which includes 9 arcs is given in Figure 5. The arcs included in the solution are in bold, and they create the following three components: {1, 2, 4, 9}, {3, 5, 7}, {6, 8, 10, 11}.

We give the following implementation to the restriction. Let G be the number of different specialist groups. For each g in 1, …, G, we require that not all members of Specialist_Group[g] are in the same component. Let k the cardinality of Specialist_Group[g]. Note that if all members of Specialist_Group[g] were in the same component then variable y_{ij} would be 1 for any $i, j \in$ Specialist_Group[g]. Since there are $k(k-1)/2$ such variables the sum of all those variables would be $k(k-1)/2$. Thus, to make sure that not all members of

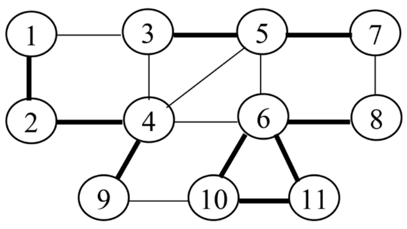

Figure 5. Example with Restricted Connections.

Specialist_Group[*g*] are in the same component we require that the sum is no more than $k(k-1)/2-1$.

Note that if there are only 2 members *i*, *j* in the group then the corresponding constraint is $y_{ij} \le 2(2-1)/2-1=0$; that is, *i* and *j* are required to be in different components.

The AMPL implementation of the constraint is given below.

s.t. Cannot_be_in_the_same_component{*g* in 1,...,*G*}:

sum{*i* in Specialist_Group[g], *j* in Specialist_Group[g]: *i*<*j* and *j* in component[*i*] and distance[*i,j*] < = *M*-1}

connected[*i,j*] < = (card(Specialist_Group[*g*])-1)*card(Specialist_Group[*g*])/2 - 1;

# for each Specialist_Group[*g*], at least 2 members of the group are in different components.

3.3. Models with Penalties for Exceeding the Maximum Component Size

In some situations the maximum component size *M* might be exceeded if it is very important for company's work. But in that case a penalty must be paid: it could be a fine paid to the government, potential extra health costs, etc.

To illustrate the idea, consider the example of Figure 6(a) (unweighted arcs, *N* = 12, *M* = 3). The arcs included in the optimal solution are in bold, and they create the following four components: {1, 3, 4}, {5, 7, 8}, {9, 11, 12}, {2, 6, 10}.

The optimal solution in Figure 6(a) includes 11 out of 20 possible arcs in the original network which means that the company has to cut almost half of the personal interactions among employees, and it can significantly affect the effectiveness of company's operations. Now suppose the company can exceed the maximum component size *M* = 3 but has to pay penalty for it. The penalty size depends on the extent *M* is exceeded. Exceeding *M* just by 1 results in a solution given in Figure 6(b). The new solution has 18 arcs (bold in figure) that form three components: {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}.

Summarizing, by slightly exceeding the maximum component size and paying corresponding penalty the company was able to increase the number of personal interactions significantly, from 11 to 18. Thus, in some situations it might make sense to pay penalty if it significantly improves the effectiveness of company's operations.

In the rest of this subsection we discuss how to incorporate penalty in the

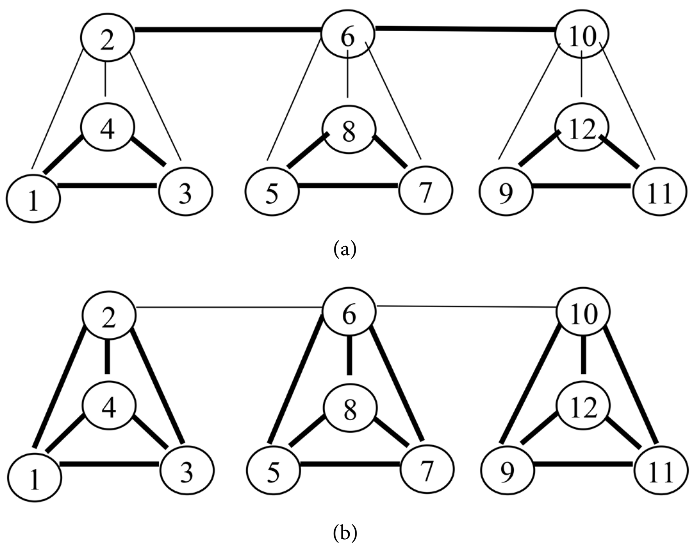

Figure 6. (a) Example with No Penalties; (b) Example with Penalties Allowed.

model. Penalty can be handled in two different ways.

1) Including the penalty in the *objective function*. In that case the goal is to minimize the total penalty.

2) Having a *constraint* for the total penalty, requiring that it cannot exceed a certain number.

Below are the modifications in the model for methods (i) and (ii) with corresponding AMPL implementations.

Modifications that are common for Methods (i) and (ii).

For both methods, we need a new penalty variable and a modified version of the constraint for maximum component size.

In our model, we define a penalty variable for each node. If a node ends up in an output component that exceeds *M* then the penalty for the node is defined to be the amount by which *M* is exceeded. Larger violations are penalized more. In reality, the way of defining penalty could be modified to reflect the government regulations of defining the fine levels.

Let p_i be the penalty for exceeding *M* by the component of node *i*. Then constraint (C1) is modified to include the new penalty variable:

$$\sum_{j \in \text{Nodes}} y_{ij} \le M - 1 + p_i$$

Note that the right-hand side of the constraint now includes a variable p_i, allowing the component size to be exceeded by p_i.

Another minor change from the basic model is the following. Recall that in Section 2.2 we argued that there is no point of defining variable y_{ij} if the shortest distance between *i* and *j* is greater than *M* – 1. While it worked in the basic model we have to define this type of variables in this section; penalties allow to have two nodes in the same output component even if the distance between them is greater than *M*-1 since component sizes can exceed *M*.

Modifications for Method (i): changes in objective function.

We have two conflicting objectives in this case: maximizing the total weight of included arcs, and minimizing the total penalty. Both parts can be included in the objective function by giving them weights that add up to 1. Let w_p be the weight of total penalty in the objective function which takes a value between 0 and 1. Then we have the following modified objective function.

$$\max\left(1-w_p\right)\cdot\sum_{(i,j)\in Arcs}\text{weight}\left(i,j\right)*x_{ij}-w_p\cdot\sum_{i\text{ in Nodes}}p_i$$

The penalty weight w_p is an input parameter that can be controlled by the user of the model. The lower the weight of the total penalty the more likely that M will be exceeded for some components.

Modifications for Method (ii): adding a constraint for the total penalty.

In this case we define a new input parameter P for the total allowed penalty level. Then we need the following constraint which says that the total penalty cannot exceed P.

$$\sum_{i\text{ in Nodes}}p_i\le P$$

The company can decide what level of total penalty is acceptable for it. We suggest setting a level proportional to N: bigger companies can afford larger penalties.

Comparison of Methods (i) and (ii).

Each method has its advantages and disadvantages. **Method (i)** allows the user to find a right balance between two important factors: the total weight of included arcs and the total penalty. But it does not limit the total penalty, and a lower value for w_p might result in unacceptably large penalty. To illustrate it, we reconsider the example of Figure 1 (N = 11, M = 4, unweighted arcs) when a penalty component is added to the objective function with weight w_p = 0.05. The model returned a solution that included all 16 arcs of the network; but the total penalty was 11 × 7 = 77 (for each of the 11 nodes its component size is exceeded by 7). Having a reasonable limit on the total penalty as in Method (ii) would not allow to have such a large penalty for a small example with 11 nodes.

Method (ii) puts a limit on the total penalty. But not having any penalty component in the objective function might result in the following situation. The company might end up paying a significant penalty (still below the penalty limit) but gain very little in the total weight of included arcs. To illustrate it, we reconsider the example of Figure 1 (N = 11, M = 4, unweighted arcs) with the following modification: the total allowed penalty level is P = 5. A new optimal solution is given in Figure 7. It has three components; one of them, {1, 2, 3, 4, 5} exceeds the maximum component size by 1, thus creating a penalty of 5. But paying the significant penalty allows to increase the number of included arcs only by one: those are the 11 bold arcs in Figure 7 versus 10 arcs in the original solution of Figure 1. On the other hand, Method (i) returns the original optimal solution of Figure 1 for w_p = 0.5 (equal weights for number of included arcs and total penalty).

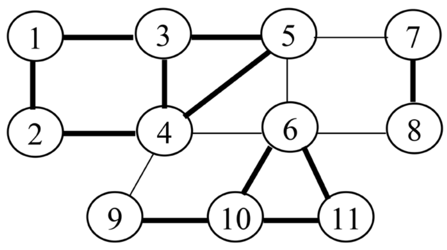

Figure 7. Example with a limit on total penalty.

Based on the discussion above, a better way to handle the penalty situation is to combine methods (i) and (ii). We suggest including both the constraint from method (ii) and the penalty component of the objective function from method (i) in the model.

An alternative way could be the following: run the model of method (i) for different values of weight w_P, consider those solutions that keep the penalty under the maximum allowed level P, and pick the solution that has the maximum weight of included arcs. We give more details about this approach in the computational results of Section 5.

4. Heuristics

The integer programming models in Sections 2 and 3 guarantee to return optimal solutions for the problem. But for a large problem size, the IP models might be slow in practice (see Section 5 for more discussion on computational efficiency). In this section, we give heuristic algorithms for solving the basic problem and its variations. Note that the heuristics do not guarantee to return optimal solutions. But they might return pretty good solutions that might serve as initial solutions for the integer programming solvers and thus accelerate the solution process.

The section is organized as follows. In Section 4.1 we give a heuristic for the basic problem given in Section 2. In Sections 4.2-4.4, the heuristic is extended to the three variations of Section 3.

4.1. A Heuristic for the Main Problem

Highest Weight First (HWF)

1) Sort the arcs in decreasing order of their weights (break ties arbitrarily);

2) Consider the arcs (one at a time) in the order given in Step 1. Include the arc in the solution if the maximum component size M is not violated in the result.

To illustrate how the heuristic works consider the example in Figure 8. The arc weights are given on the arcs, and the maximum component size is $M = 4$. The heuristic first includes all four arcs with the highest weight 0.8: (1, 2), (2, 3), (5, 6), (6, 7). At this point two components with size 3 are created: {1, 2, 3} and {5, 6, 7}. The heuristic next considers the only arc with weight 0.7: arc (2, 6). But adding it to the solution would create component {1, 2, 3, 5, 6, 7} with size 6, thus exceeding $M = 4$. So (2, 6) is not included in the solution. Then the heuristic includes all other remaining arcs with weight 0.6 without exceeding M = 4. In the result, there are two components in the solution: {1, 2, 3, 4} and {5, 6, 7, 8}.

For the example of Figure 8, the output of the heuristic happens to be the optimal solution. But the heuristic might also return a solution far from the optimal as shown in the next bad-case example. Consider the network in Figure 9. The arc weights are given on the arcs, and the maximum component size is $M = 4$. The heuristic first includes all three arcs with the highest weight 1: (1, 2), (2, 6), (6, 7). At this point one component with size 4 is created: {1, 2, 6, 7}, and no other arc can be added to this component since $M = 4$ will be exceeded. Thus, the only other arcs that can be included are (3, 4) and (5, 8), creating two more components {3, 4} and {5, 8}.

The total weight of the arcs included by the heuristic is $3 \times 1 + 2 \times 0.9 = 4.8$. But the heuristic output is not optimal. The optimal solution is given in Figure 10.

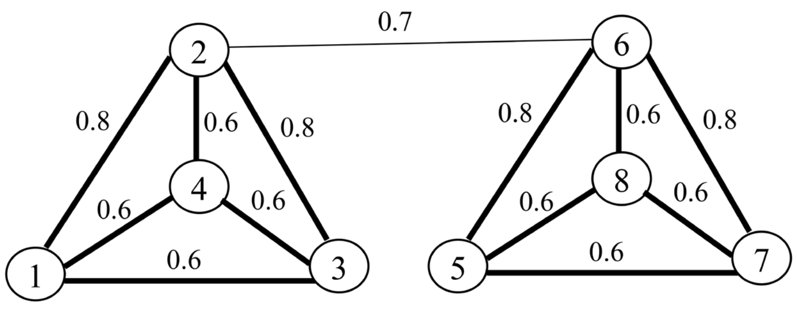

Figure 8. Example illustrating how the heuristic works.

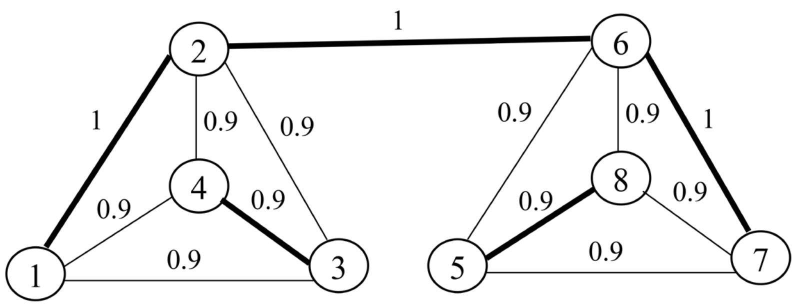

Figure 9. A bad-case example for the heuristic.

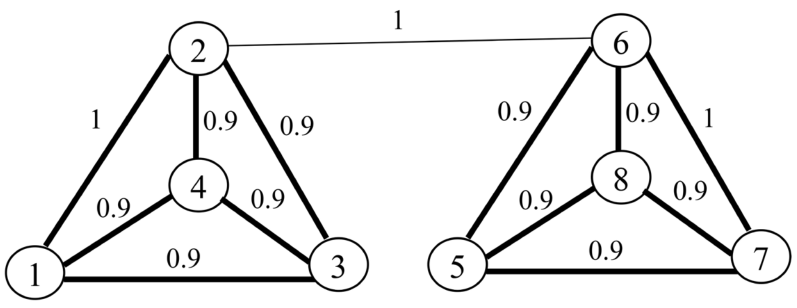

Figure 10. Optimal solution for the bad-case example.

The total weight of the arcs in the optimal solution is $2 \times 1 + 10 \times 0.9 = 11$. Thus, for this example the heuristic returns a solution far from the optimal. More discussion about the heuristic performance is given in Section 5.

Here is the AMPL implementation of the heuristic.

```
################# HEURISTIC ###################
set ordered_weights ordered by reversed Reals : = setof{(i,j) in Arcs} weight[i,j];
# ordering the arc weights in decreasing order
param heur_incl{(k,l) in Arcs} binary;
for{(k,l) in Arcs} let heur_incl[k,l] : = 0;
# is 1 if the arc is included in the heuristic solution before the heuristic starts, all values are 0
set heur_component{k in Nodes};
for{k in Nodes} let heur_component[k] : = {k};
# the heuristic component of each node originally, it consists of the node itself
for{s in 1..card(Arcs)}
  for{(k,l) in Arcs: weight[k,l] = member(s,ordered_weights)}
# go through the arcs in decreasing order of their weights
        if card(heur_component[k] union heur_component[l]) < = M
# if including the arc does not violate M for the resulting component
             then
                  {
                  let heur_incl[k,l] : = 1;
# include the arc in the solution
                  for {m in Nodes:
                       m in heur_component[k] or m in heur_component[l]}
                  let heur_component[m] : = heur_component[k] union heur_component[l];
# update the component for all the nodes in the resulting bigger component
                  }
;
##################################################
```

4.2. The Heuristic Applied to the Variation with Subset Interaction Requirements

The original heuristic is modified by including an extra stage as follows. In the first stage (step 2 below), arcs are included in the solution to satisfy the subset requirements. The second stage (step 3 below) includes arcs in the same way as the original heuristic.

Highest Weight First with Subset Interaction Requirements (HWF_SIR)

1) Sort the arcs in decreasing order of their weights (break ties arbitrarily);

2) Consider the arcs (one at a time) in the order given in Step (1). Include the arc in the solution if it satisfies a subset requirement not satisfied yet with pre-

vious arcs. Repeat until all subset requirements are satisfied.

3) Consider the arcs not included in the solution in Step 2, again in the order given in Step (1). Include the arc in the solution if the maximum component size M is not violated in the result.

4.3. The Heuristic Applied to the Model with Restricted Connections

A new provision is added to the original heuristic to make sure that not all specialists of a group end up in the same component.

Highest Weight First with Restricted Connections (HWF_RC)

1) Sort the arcs in decreasing order of their weights (break ties arbitrarily);

2) Consider the arcs (one at a time) in the order given in Step (1). Include the arc in the solution if the following two conditions are satisfied:

a) the maximum component size M is not violated in the result;

b) including the arc does not create a component that includes all the specialists of a group.

4.4. The Heuristic Applied to the Models with Penalties

We give a modified heuristic for the version when the model has a constraint which puts an upper bound on the total penalty.

Highest Weight First with Penalties (HWF_P)

1) Sort the arcs in decreasing order of their weights (break ties arbitrarily);

2) Consider the arcs (one at a time) in the order given in Step (1). Include the arc in the solution if including the arc does not exceed the total allowed penalty (note that the maximum component size M might be violated in this version).

5. Computational Results

The AMPL models for the integer programs were run on the NEOS server using the solver CPLEX [6]. The running time for randomly created instances was 22 seconds for $N = 30$, 300 seconds for $N = 40$, and 7094 seconds for $N = 50$. Thus, the integer programming model is efficient enough for small and medium size companies with number of employees up to 50. For larger companies, more efficient solution methods should be used.

Next we discuss some computational results for the penalty models and the heuristic.

Computational results with sensitivity analysis for penalty models

In Section 3.3, we discussed two methods for handling penalty, the first one including a penalty component with weight w_p in the objective function, and the second one putting an upper bound P on the total penalty allowed. But as was discussed at the end of that section, a better way to handle penalty would be to combine the two methods. Below we illustrate it based on our computational results.

Computations were run for a randomly created instance with $N = 30$, $M = 4$, unweighted arcs (which allows to better understand the results). The basic model when no penalty is allowed returned the following output: 37 arcs were included in the solution (out of total possible 139); the solution had two output components of size 3 and six output components of size 4.

Then we tried the model with different weights w_p of penalty in objective function. For each w_p, the model was run for two different scenarios: 1) when there was no restriction on total penalty, 2) with a limit $P = 30$ on total penalty. Note that the first scenario is method (i) of Section 3.3 while the second scenario represents the combined version of methods (i) and (ii). Pure method (ii) is also represented by the second scenario when $w_p = 0$. The results are summarized in **Table 1**. The values of w_p are in increments of 0.05 in the range from 0 to 0.95. The results in the range from 0.3 to 0.9 are identical and thus are not included in the table for a more compact presentation.

Below is an analysis of the results given in **Table 1** with some conclusions.

Scenario 1: *No penalty restriction*. The general trend is clear and intuitive: as w_p increases, the total penalty as well as the number of included arcs decrease. For small or no penalty weights $w_p = 0.05$ and $w_p = 0$, the model includes all possible 139 arcs in the solution, resulting in one output component of size 30 and incurring large total penalty of 780. For $w_p = 0.1$ and $w_p = 0.15$, the total penalty is still unacceptably high (above 100) for a small company with 30 employees. For $w_p \geq 0.3$, $M = 4$ is not violated and thus no penalty is occurred; the solution is the same as the output of the basic model.

The most sensible w_p values are in the range from 0.2 to 0.25. When $w_p = 0.25$, 41 arcs can be included by paying penalty of 10. But if the company is willing to pay more penalty then a solution found for $w_p = 0.2$ allows 50 arcs by paying penalty of 44. We also tried the model for different values of w_p between 0.2 and

Table 1. Computational results for penalty models.

	no penalty restriction			total penalty at most $P = 30$		
penalty weight, w_p	arcs included	total penalty	component sizes	arcs included	total penalty	component sizes
0	139	780	30	45	29	6, 6, 5, 4, 4, 4, 1
0.05	139	780	30	45	26	7, 5, 2, 4, 4, 4, 4
0.1	93	317	17,12,1	45	26	7, 5, 2, 4, 4, 4, 4
0.15	63	109	11,8,4,4,3	45	26	7, 5, 2, 4, 4, 4, 4
0.2	50	44	8,6,4,4,4,4	44	21	7, 3, 4, 4, 4, 4, 4
0.25	41	10	5,5,4,4,4,4,4	41	10	5, 5, 4, 4, 4, 4, 4
0.3	37	0	3,3,4,4,4,4,4,4	37	0	2, 4, 4, 4, 4, 4, 4, 4
0.9	37	0	3,3,4,4,4,4,4,4	37	0	2, 4, 4, 4, 4, 4, 4, 4
0.95	37	0	2,4,4,4,4,4,4,4	37	0	2, 4, 4, 4, 4, 4, 4, 4

0.25, and another intermediate solution was obtained with 43 arcs and penalty of 17. The company can pick one of the last three solutions with 41, 43, and 50 arcs based on a more careful cost-benefit analysis more specific to the company's needs.

Scenario 2: *Total penalty restricted*. The upper limit is set to $P = 30$ which corresponds to one unit of penalty per employee. Of course, different upper limits can be tried; we just picked a sensible value to illustrate how the combined version of Methods (i) and (ii) of Section 3.3 would work.

For $w_p \geq 0.3$, as in Scenario 1 the solution is the same as the output of the basic model: $M = 4$ is not violated and thus no penalty is occurred. For $w_p = 0.25$, the solution again is the same as in Scenario 1: 41 arcs can be included by paying penalty of 10. But for $w_p = 0.2$, a new solution is found with 44 arcs and a penalty of 21. When w_p is between 0.05 to 0.15, the solution includes 45 arcs by paying penalty of 26. If not exceeding the penalty limit $P = 30$ is very important then the last two solutions with 44 and 45 arcs are the best ones. Note that those solutions could not be obtained in Scenario 1 because for the same values of w_p the solutions violated the penalty limit $P = 30$.

Note also that for $w_p = 0$, Scenario 2 is simply the Method (ii) and it returns a solution with 45 arcs but with penalty 29. The reason for higher penalty with the same number of arcs is that Method (ii) does not have any penalty component in the objective function, and thus any solution with a penalty under $P = 30$ can be optimal.

Thus, our conclusion is that Scenario 2 with $w_p > 0$ (combining Methods (i) and (ii)) is a better way to proceed if the penalty limit P cannot be violated.

Heuristic performance

We tested the main heuristic of Section 4.1 for different combinations of M and graph densities. Five instances were randomly created for each combination. $N = 30$ for all instances. We compared the heuristic output with the optimal solution returned by the integer program in terms of number of included arcs. The numerical results that represent the ratios of the number of arcs returned by the heuristic and the integer program are summarized in Table 2.

The heuristic performance gets slightly worse for larger M. It also gets slightly worse for graphs with higher density. On average, the heuristic output is within 0.8142 of the optimal value.

Table 2. Computational results for heuristic performance.

	dens = 0.2	dens = 0.4	dens = 0.6	average
$M = 3$	0.893	0.801	0.787	0.827
$M = 4$	0.876	0.803	0.805	0.828
$M = 5$	0.82	0.785	0.758	0.7877
average	0.863	0.79633	0.78333	0.8142

6. Future Directions

Below we discuss several future directions.

In Section 2 and 3 we developed the main model and its variations to take into account some reasonable restrictions that can be common for most companies. The models can be further developed to incorporate other government regulations and company-specific restrictions. For example, one can do more careful analysis with the penalty models by defining the penalties in monetary terms.

The integer program is a reasonably efficient technique for solving the problem for companies with up to 50 employees. For bigger problems, more time-efficient techniques are required. One direction is further refining the integer programming model to make it computationally more efficient. The other direction is developing other heuristics and approximation algorithms that are time-efficient and return close to optimal solutions in practice.

In this paper we consider how direct interactions between employees can be affected by social distancing regulations. An extension would be to model how the scheduling of more complex operations in companies can be affected by those regulations.

In our models, we assigned weights to employee links to characterize their relative importance to the company. But the weights might not be the only relevant link characteristics. Recall that the main intent of partitioning is to limit the spread of the disease. Some links might be more infectious than others. There might be different reasons for that: the closeness of the employees during the interaction, the duration of the interaction, some employees being more likely to get infected than others (different age groups), etc. Thus, one could introduce a probability indicating how infectious the link is. The resulting model will be stochastic, and more complex techniques would be needed to solve it.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Covid-19 Policy Watch: Tracking Governments' Responses to the Pandemic. https://covid19policywatch.org/

[2] U.S. State and Local Government Responses to the COVID-19 Pandemic. https://en.wikipedia.org/wiki/U.S._state_and_local_government_responses_to_the_COVID-19_pandemic

[3] Buluç, A., Meyerhenke, H., Safro, I., Sanders, P. and Schulz, C. (2016) Recent Advances in Graph Partitioning. *Lecture Notes in Computer Science*, **9220**, 117-158. https://doi.org/10.1007/978-3-319-49487-6_4

[4] Andreev, K. and Racke, H. (2006) Balanced Graph Partitioning. *Theory of Compu-*

ting Systems, **39**, 929-939. https://doi.org/10.1007/s00224-006-1350-7

[5] The AMPL Website. http://www.ampl.com/

[6] The CPLEX Website. https://neos-server.org/neos/solvers/lp:CPLEX/AMPL.html