BACHELOR

A Routing Problem for ‘Pandemic’

Aerts, Willem J.H.

Award date:
2021

A Routing Problem for 'Pandemic'

Author
Willem Aerts (1238531)

Supervisor
Bart Smeulders

Eindhoven University of Technology
Department of Applied Mathematics and Computer Science
2WH40 Bachelor Final Project

30th June 2021
Eindhoven University of Technology

Abstract

Pandemic is a board game in which four diseases are spreading around the world. In this game, players have to cooperate to win the game. Here, we aim to come up with a model, so we will not lose the game. The 'Pandemic Routing Problem' is very similar to a Vehicle Routing Problem with Time Windows. Three different formulations are made. The first two formulations are based on found literature on the Vehicle Routing Problem, and the third formulation is a variation on this. We want to compare performance of these formulations. Overall, we conclude that one of the formulation suits the Pandemic routing problem best, as this formulation has the best running times and is quite consistent.

Contents

1 Introduction **3**
1.1 Components . . . 3
1.2 Winning and Losing Conditions . . . 4
1.3 Setup of the Game . . . 4
1.4 Round Overview . . . 4
1.5 Outline of Report . . . 5

2 Problem Description **6**
2.1 Alternative Objective of the Game . . . 6
2.2 Time Windows . . . 6
2.3 Formal Problem Description . . . 7

3 Vehicle Routing Problems **8**
3.1 Vehicle Routing Problem . . . 8
3.1.1 Set Partitioning Formulation . . . 8
3.1.2 Two-index Vehicle Flow Formulation . . . 10
3.2 Vehicle Routing Problem with Time Windows . . . 10
3.2.1 Set Partitioning Formulation for VRPTW . . . 11
3.2.2 Two-index Vehicle Flow Formulation for VRPTW . . . 12
3.3 Relation with Problem Description . . . 13

4 Programming **14**

5 The Models **15**
5.1 Assumptions . . . 15
5.2 Set Partitioning Formulation . . . 15
5.3 Shortest Path Formulation . . . 17
5.4 Adjacency Formulation . . . 18

6 Results **20**
6.1 Set Partitioning Formulation . . . 20
6.2 ILP Formulation(s) . . . 20
6.2.1 Basic Example . . . 20
6.2.2 Performance no Repetition of Infection . . . 20
6.2.3 Performance Repetition of Infection . . . 20

7 Discussion **23**
7.1 Brute Force . . . 23
7.2 ILP Formulation(s) . . . 23
7.2.1 No Repetition of Infection . . . 23
7.2.2 Repetition of Infection . . . 23

8 Conclusions and Recommendations **24**
8.1 Conclusions . . . 24
8.2 Future Research . . . 24

A Appendix **26**

1 Introduction

In Pandemic [1], four deadly diseases are spreading across the globe. Two up to four players have to cooperate to battle these diseases, and can win the game by discovering the cure for each disease. The board (Figure 1) is made up of a network, containing 48 cities. Each of these cities is either red, blue, yellow, or black coloured. These colours are based on their geographical location, and each of these colours represents one disease.

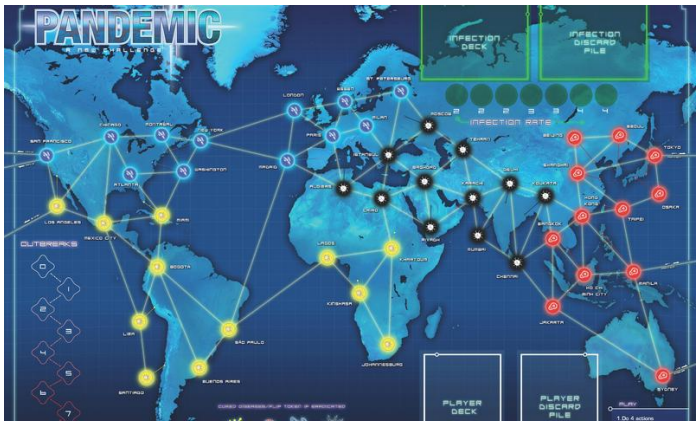

Figure 1: Overview of the board

In this report we are going to model (a part of) this board game. For the real world it is not very relevant to model this entire game, but it could be interesting to investigate whether one can contain the number of outbreaks or to try to remove all disease tokens within a certain period of time. Therefore, we want to investigate whether it is possible to remove all disease tokens before the game ends, given an ordering of the infections of the cities. These kind of problems are similar to the Vehicle Routing Problem (VRP) which we will shed some light on later.

We will now explain the rules of the game so that the game can be transformed into a mathematical model. The components of the game, the winning and losing conditions, the setup, and the overview of a player's turn will be explained in the upcoming subsections.

1.1 Components

As the game has many different components, we will first introduce them before continuing with the rules of the game.

- **Player pawns.** Each pawn represents a player on the board.
- **Disease tokens.** The number of tokens in a location represents the severity of the disease in that city. For each disease, at most 24 tokens can be placed on the board.
- **Player deck.** The player deck contains 50 cards which are used for discovering cures, among other things.
- **Epidemic cards.** The player deck also contains epidemic cards. The number of these cards is determined before the game starts, and they are equally distributed over the deck, i.e. the first card is somewhere in the first part of the deck, the second in the second, etc..
- **Infection deck.** The infection deck contains 48 cards, each card representing one of the cities on the board. From this deck, cards are drawn to infect the corresponding cities.
- **Infection discard pile.** After having drawn a card from the infection deck, the card is placed on this discard pile.

- **Infection rate marker.** The infection rate marker keeps track of the rate at which cities get infected. The rate starts at two, but can be increased to at most four.
- **Outbreak marker.** The outbreak marker shows how many outbreaks have happened.

1.2 Winning and Losing Conditions

The players can win the game by collaborating with each other and discovering the four cures. But as it is mostly interesting to see if we can prevent diseases to spread across the globe, this report will mainly focus on the spreading of the diseases. Therefore, we will now discuss what the losing conditions are of the game in a little more detail. The players lose the game immediately if either:

- **There are insufficient disease tokens (a disease is spreading too fast).** There are no more disease tokens left of one of the four colours to be placed on the board.
- **There are no cards anymore (time is up).** All cards from the player deck have been drawn, meaning that too many turns were needed to win the game.
- **8 outbreaks have taken place (the world is in panic).** An outbreak takes place when a fourth (or more) disease token *of the same colour* gets placed in a city. In this case, the extra disease token(s) will not be placed in this city, but in every adjacent city a disease token will be placed of the colour of the outbreak. If this means that another outbreak occurs, the procedure is exactly the same, except that the cities which have had an outbreak in this chain, will not have an outbreak again. If 8 of these outbreaks have occurred, the game is lost as well. (The players keep track of the number of outbreaks with the outbreak token which has been introduced in the previous section.)

If any of these conditions are met at any point in the game, so also during a player's turn, the game is immediately lost.

1.3 Setup of the Game

In the beginning of the game, every player picks one of the seven pawns. This will be their pawn for the entire game. When everyone has decided who to play with, all players put their pawn in Atlanta, the city where the Centers for Disease Control and Prevention (CDC) is located. In this city a research station is placed. At research stations, cures can be discovered, and players can fast travel from one research station to another. Next, initial disease tokens will be placed on the board by drawing nine cards from the infection deck; on the first three cities, three disease tokens of the corresponding colour will be placed. On the next three cities, two disease tokens will be placed. And on the last three cities, one disease token will be placed. Meaning that there are 18 tokens on the board. The cards that were drawn are placed on the infection discard pile and the game can now start.

1.4 Round Overview

Every player takes their turn in a clockwise fashion. Meaning that only one player is allowed to do something in a turn. In a player's turn, a few steps have to be taken:

1. **Do up to four actions.** These actions are for instance, moving on the map, removing disease tokens in a city, and discovering a cures. After having done four actions, or less if one does not want to use all four actions, the player moves on to the next step.
2. **Draw two cards from the player deck.** These cards are used to discover cures and to move around the map quicker. When there are no more cards left to be drawn, one of the losing conditions has been met and the game is immediately lost.

3. **Infect cities.** This is done by drawing two up to four cards from the infection deck, depending on the position of the infection rate marker. On every city which has been drawn, one disease token (of the corresponding colour) will be placed. Keeping in mind that outbreaks can occur, and if there are no more disease tokens left of a given colour, the game is lost immediately.

The player deck can also contain epidemic cards. Meaning that a player can also draw this card in step 2. When an epidemic card gets drawn, a few steps have to be taken. First of all, the infection rate will be increased by moving the infection rate marker one spot higher, meaning that (possibly) more cities will be infected in the 'infect cities' step of a player. Second, the last card of the infection deck will be drawn, and three disease tokens will be put in this city, after which the card is placed on the discard pile. And last, all cards that were placed on the discard pile, including the card that was just drawn, have to be shuffled and placed on top of the infection deck. This way, the cities that got 'recently' infected will be infected again before any other city, possibly causing outbreaks.

This concludes a player's turn, and the next player can now start at step 1. This is repeated until either the winning or losing conditions are met.

1.5 Outline of Report

Having explained the most essential rules of the game, we can now give an outline of the report. In Chapter 2 the problem description will be given, after which we will introduce VRPs in Chapter 3. Then we will shortly mention the programming software that is used in Chapter 4. Next, Chapter 5 will shed some light on different formulations of models which will be used to solve the problem, of which the results will be presented in Chapter 6. The results are discussed in Chapter 7. And lastly, a conclusion will be drawn in Chapter 8, along with some suggestions for future research.

2 Problem Description

In this chapter we will discuss a few things related to the problem description. First, we will change the objective of the game a little. This is needed because the game contains several stochastic elements which make the problem significantly more difficult to solve. This is done in Section 2.1. Next, we will discuss time windows in Section 2.2. Finally, we will give a formal problem description in Section 2.3.

2.1 Alternative Objective of the Game

Like mentioned in the previous chapter, the report will mainly focus on 'not losing' the game, rather than winning the game. In this report, we will focus on the last two losing conditions given in Section 1.2 (there are no more cards, and 8 outbreaks have taken place), this is done to simplify the game. Using these conditions, the alternative objective becomes to remove all disease tokens before the end of the game, and to prevent outbreaks. Furthermore, to eliminate the stochastic elements of the game, we assume that we are given the order that the cities will get infected, and when epidemic cards are played. Therefore, we can derive the time window in which every city must be visited by a player to achieve the alternative objective. From now on we will refer to the cities with numbers. The corresponding numbers can be found in Appendix A.

2.2 Time Windows

Every city will get a time window $[a_i, b_i]$ assigned to it, in which a_i is the earliest moment a city can be served and b_i the latest moment. The earliest moment is the moment a city gets first infected, and the latest moment is either before the end of the game, or before an outbreak happens so that the losing conditions are not met. If every city gets served by a player within this time window, our alternative objective is achieved. Hence, these time windows are dependent on the order in which the cities get infected. Note, the difference between visiting and serving a city, is that a city can only be served, and must be served, once in the time window of that city. All other visits, are called visiting a city.

In this report we will consider two different scenario's. The first scenario does not include any epidemic cards, and therefore there is no repetition of infection in the same city, meaning that there will not be any outbreaks possible. The second scenario *does* include epidemic cards, meaning that it is possible that outbreaks occur due to repetition of infection. Hence, it is possible that cities have to be visited multiple times, and there could be extra pressure to visit a city earlier in the game.

The time windows are based on the given ordering of infection of the cities. If there is no repetition of infection, then we can simply assign time windows in the order that the cities get infected. An example of time windows without using epidemic cards can be found in Appendix A. Later in the report this example will be referred to as Test 1. This example has been constructed in such a way that it is solvable for sure. This was done by taking a shortest path through all cities, and assigning them a time window in this order. The shortest path was determined by using a solver for the Traveling Salesman Problem.

If epidemic cards are used, we have to worry about possible outbreaks, as some cities will be infected multiple times, and some cities will not be infected at all. Therefore, some cities will have multiple time windows, and other cities will not have a time window at all. An example of time windows when using epidemic cards can be found in Appendix A. If a city must be served multiple times there are multiple values for a and b. If a city does not have to be served, there is a -1 for both a and b.

In Table 1 a part of the tables in the appendix is given. It contains the city numbers of the first 6 cities. As well as the example of the time windows for no repetition of infection, and the example of the time windows with repetition of infection.

City	City Number	No repetition		Repetition	
		a	b	a	b
Atlanta	0	0	100	0	100
Algiers	1	60	100	8	100
Baghdad	2	40	100	-1	-1
Bangkok	3	24	100	0	100
Beijing	4	8	100	48	100
Bogota	5	72	100	-1	-1

Table 1: Numbers that will be used instead of the names of the city, along with an example of time windows for the two different scenario's

2.3 Formal Problem Description

We are given an undirected graph $G = (V_{all}, A)$ with $V_{all} = \{0, \ldots, n\}$ the set of all cities and A the adjacency matrix. Moreover, we are given the number of players that is playing, m. In addition, we are given a set $V \subseteq V_{all}$ which is the set of all cities which must be served by the players, along with the time window $[a_i, b_i]$ in which every city $i \in V$ must be served at least once by a player. We also have that $a_i, b_i \in T = \{0, 1, \ldots, t_{max}\}$. Here, T is the set of discrete time steps. At any time step the players may either move to a different city, or wait at a city. The waiting time at city $i \in V_{all}$ is denoted by w_i.

Next, we define $0 \in V_{all}$ to be the depot. Every players starts here. Furthermore, we define a route $r = (v_0, v_1, v_2, \ldots, v_k)$ with $v_i \in V_{all}$ as a path on the graph, i.e. $(v_i, v_{i+1}) \in A \; \forall i : 0 \leq i \leq k-1$. All players start in the depot, $v_0 = 0 \in V_{all}$ in every route. We define the length of a route as $|r| = k + w \leq t_{max}$, with $w = \sum_{j=v_0}^{v_k} w_j$ the total waiting time.

The objective is to construct a set of m routes, so that every city $i \in V$ gets visited by at least one route in the time window that it is given. We call this set R. So to be more precise, for every city $i \in V$ there must be at least one route $r = (v_0, v_1, \ldots, v_l, \ldots, v_k) \in R$ with $v_l = i$ and $a_i \leq |(v_0, v_1, \ldots, v_l)| \leq b_i$.

3 Vehicle Routing Problems

In this section the Vehicle Routing Problem (VRP) will be explained, as it is a useful representation of our problem. Next, an extension of the VRP, the Vehicle Routing Problem with Time Windows, will be explained, along with the extra constraints this gives.

3.1 Vehicle Routing Problem

The Vehicle Routing Problem (VRP) consists of optimizing a delivery or collection route (or routes) from a depot to a set of customers. The problem is subject to several constraints like vehicle capacity, time windows, route length, etc. This problem is faced very often by several distributors around the world. [2]

The classical VRP is defined on an undirected graph $G = (V, E)$ where $V = \{0, 1, \ldots, n\}$ is the set of vertices/customers and $E = \{(i, j) : i, j \in V, i \neq j\}$ is the set of edges/links between customers. Vertex 0 represents the depot at which all m (identical) vehicles, with capacity Q, start their route. Every customer i in the customer set $V_c = V \setminus \{0\}$ has a demand $0 \leq q_i \leq Q$ and needs to be served exactly once. A service does not cost any time. Furthermore, a matrix c_{ij} is defined on E and represents the cost of moving from i to j. We assume that c_{ij} is symmetric. The terms travel cost, length, and travel time will be used interchangeably in this report, but do not necessarily have to be the same in problems like this.

The problem consists of determining a set of routes $r \in R$ with $r = (v_0, v_1, \ldots, v_k, v_0)$ and $v_1, \ldots, v_k \in V_c$, v_0 the depot. Together all routes in R must satisfy the following:

1. The route starts and ends in the depot.
2. Every customer is served exactly once by one vehicle.
3. The total demand of the customers on a route does not exceed Q.
4. Routing cost is minimized.

An example of a VRP can be seen in Figure 2. In this figure, the blue circles are customers and their demand is given in red. Customer 0 is the depot. A possible solution when using four vehicles can be seen in Figure 3.

There are several formulations which can be used to solve the VRP. Two of those are the set partitioning formulation which uses a set of all feasible (partial) routes [2], and the two-index vehicle flow formulation which uses the flow preservation of the vehicles. As the VRP includes the Traveling Salesman Problem (TSP) when $m = 1$ and $Q = \infty$, the problem is NP-hard.

3.1.1 Set Partitioning Formulation

In this particular formulation, R is the set of all feasible routes, and d_r is the cost of route $r \in R$. A feasible route means that the capacity of a vehicle is not exceeded by the total demand of all customers on its route. Moreover, z_r is a binary decision variable, with value 1 if route r is selected, and 0 otherwise. Lastly, a_{ir} is a binary parameter, with value 1 if customer i belongs to route r, and 0 otherwise. The objective is:

$$\text{Minimize} \sum_{r \in R} d_r z_r \tag{1}$$

And it is subject to:

$$\sum_{r \in R} a_{ir} z_r = 1 \qquad (i \in V_c) \tag{2}$$

$$\sum_{r \in R} z_r = m \tag{3}$$

$$z_r \in \{0, 1\} \qquad (r \in R) \tag{4}$$

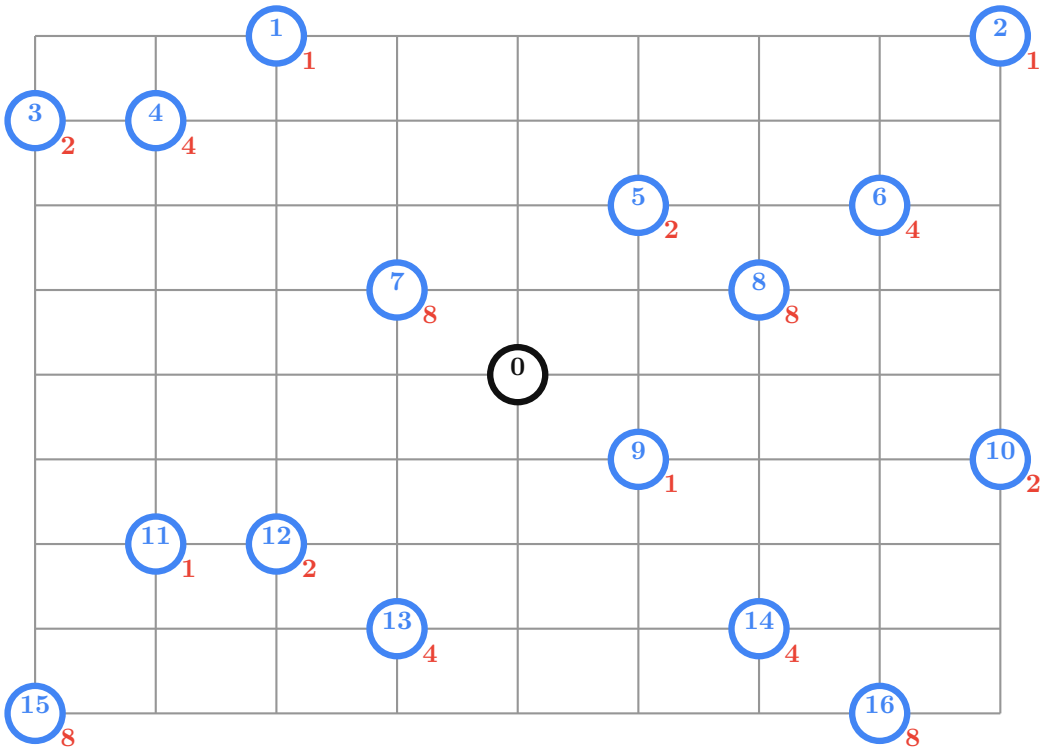

Figure 2: Example of a VRP

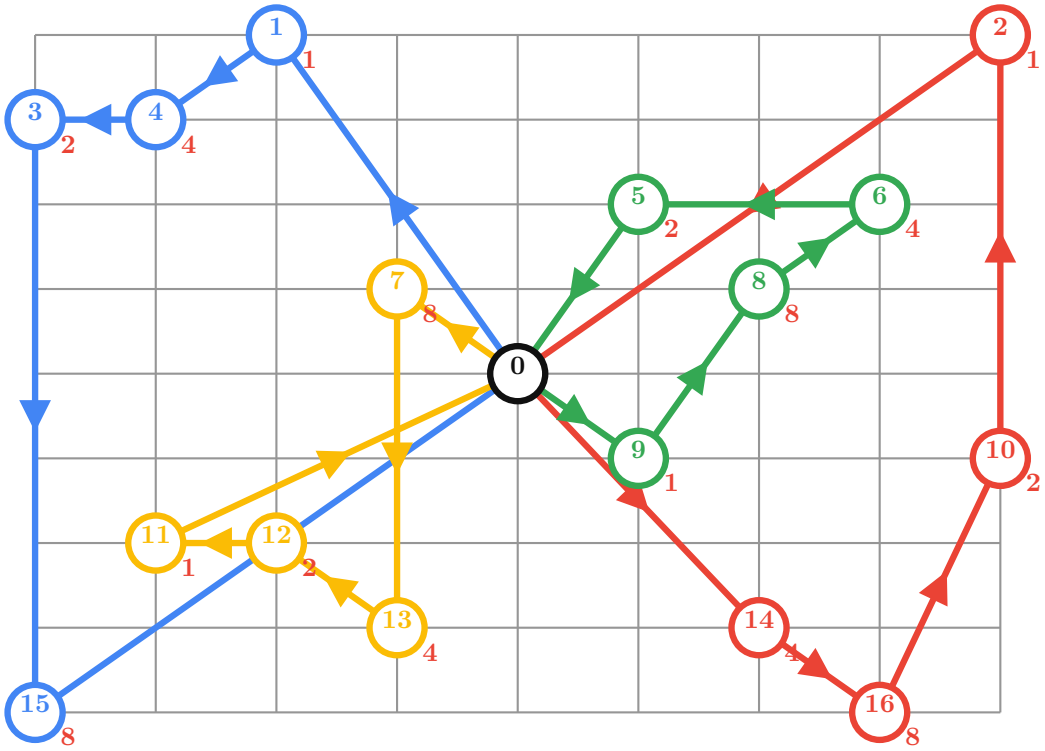

Figure 3: Possible solution of the VRP

Equation (2) makes sure that every customer gets visited exactly once in the optimal solution. Equation (3) makes sure that m vehicles are used, one route for every vehicle. And Equation (4) makes sure that a feasible route is either used or not. Partial usage of a feasible route is not allowed. The reason why we do not check for capacity, is because set R only consists of feasible routes and therefore the capacity of a vehicle can never be exceeded. Therefore, this formulation needs some preprocessing, because set R needs to be determined beforehand.

3.1.2 Two-index Vehicle Flow Formulation

The two-index vehicle flow formulation, is an extension of the classical TSP. Let the decision variable x_{ij} denote the number of times that edge (i, j) is used in the optimal solution. In all cases x_{ij} is either 0 or 1 as each edge can be used at most once. The Integer Linear Program (ILP) is then:

Objective:

$$\text{Minimize} \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{5}$$

Subject to:

$$\sum_{j\in V_c} x_{0j} = m \tag{6}$$

$$\sum_{j\in V_c} x_{j0} = m \tag{7}$$

$$\sum_{j\in V\setminus\{i\}} x_{ij} = 1 \qquad (i \in V_c) \tag{8}$$

$$\sum_{j\in V\setminus\{i\}} x_{ji} = 1 \qquad (i \in V_c) \tag{9}$$

$$\sum_{i\in S,\, j\notin S \text{ or } i\notin S,\, j\in S} x_{ij} \geq 2b(S) \qquad (S \subset V_c) \tag{10}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j \in V) \tag{11}$$

Equation (6)-(7) makes sure that the right amount of vehicles is being used, as we check the number of vehicles leaving and entering the depot. Whereas Equation (8)-(9) makes sure that every customer is entered and left by exactly one vehicle. Finally, Equation (10) plays a dual role, it prevents subtours by forcing every subset of customers to connect to the depot, and it ensures that capacity is not violated. The latter because it is common to define $b(S)$ as $\lceil \sum_{i\in S} q_i/Q \rceil$. [2] The equation works because we sum over all the edges that enter and leave the set S, and count the number of edges that is being used. As $b(S) = \lceil \sum_{i\in S} q_i/Q \rceil$, which is the minimum number of vehicles we need to serve all the customers in S when considering the capacity of the vehicles, we know that the summation must be at least twice as big as $b(S)$ (the vehicles must enter the set and leave it again).

3.2 Vehicle Routing Problem with Time Windows

The VRP has many extensions, and one of these extensions is the Vehicle Routing Problem with Time Windows (VRPTW). The objective and constraints that were mentioned in the VRP are still used, but in addition, every customer has a time window in which the customer must be served by a vehicle. These time windows are defined by a_i, the earliest moment customer i can be served, and b_i, the latest moment customer i can be served. If a vehicle arrives at a customer too early, it will wait until the customer can be served. This means our notion of feasibility has to be adjusted to the time windows. Meaning that a route is only feasible if it does not violate the time window of any customer on that route, in addition to the capacity constraint. The previously mentioned formulations can also be used to solve the VRPTW by adding the time window constraints to the problems. An example can be seen in Figure 4 along with the solution when using four vehicles in Figure 5.

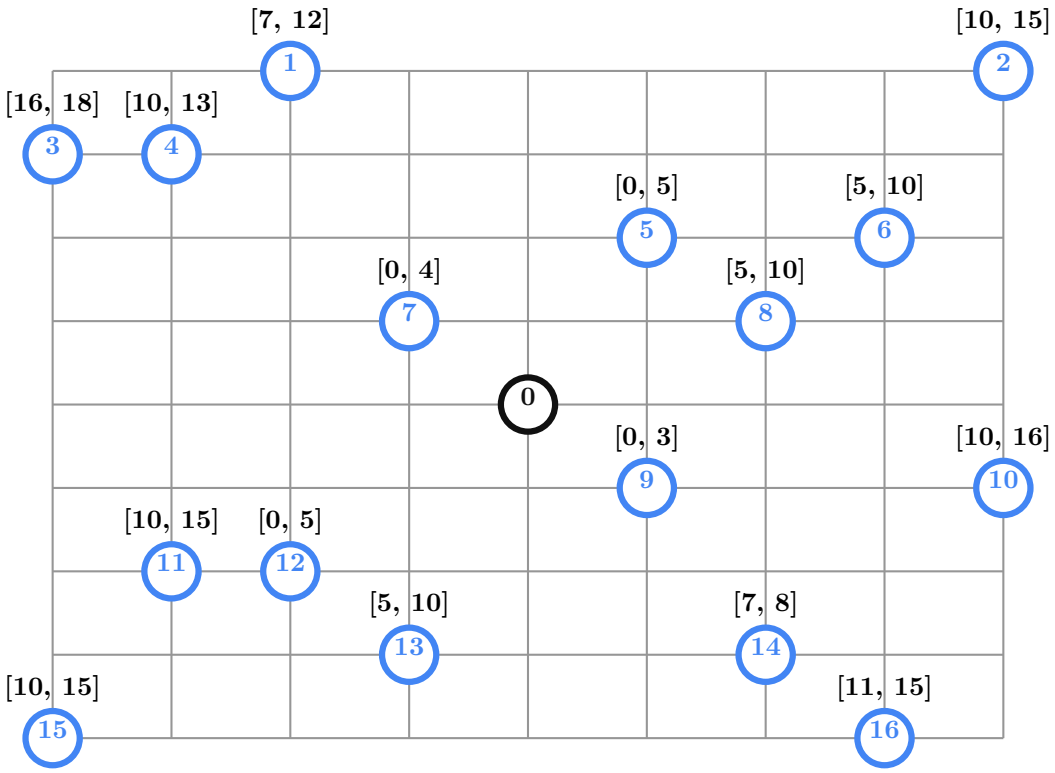

Figure 4: Example of a VRPTW

3.2.1 Set Partitioning Formulation for VRPTW

If we want to use the set partitioning formulation we do not have to add additional constraints, but we do have to determine the set of feasible routes. To do this, we define $F_i(S,t)$ as the marginal cost of the partial route starting in the depot and ending at customer i, visiting all customers in set S once, and arriving at customer i before time t. The marginal cost can be computed by solving the recurrence:

$$F_d(\emptyset, 0) = 0$$

$$F_j(S,t) = \min_{(i,j)\in E} \{F_i(S - \{j\}, t') + t_{ij} \mid t' + t_{ij} \leq t,\ a_i \leq t' \leq b_i \text{ and } \sum_{k\in S} q_k \leq Q\} \tag{12}$$

for all j, S, t such that $j \in V$, $S \subseteq V$, $a_j \leq t \leq b_j$. [3]

By using the marginal cost we can determine the set R. This is done by trying all possible combinations of j, S, and t. If it is possible to solve the recurrence, there exists a feasible partial route from the depot to customer i. By computing all these partial routes, we can determine the set R. Next, we can use the constraints and objective given in Section 3.1.1 to determine the optimal solution.

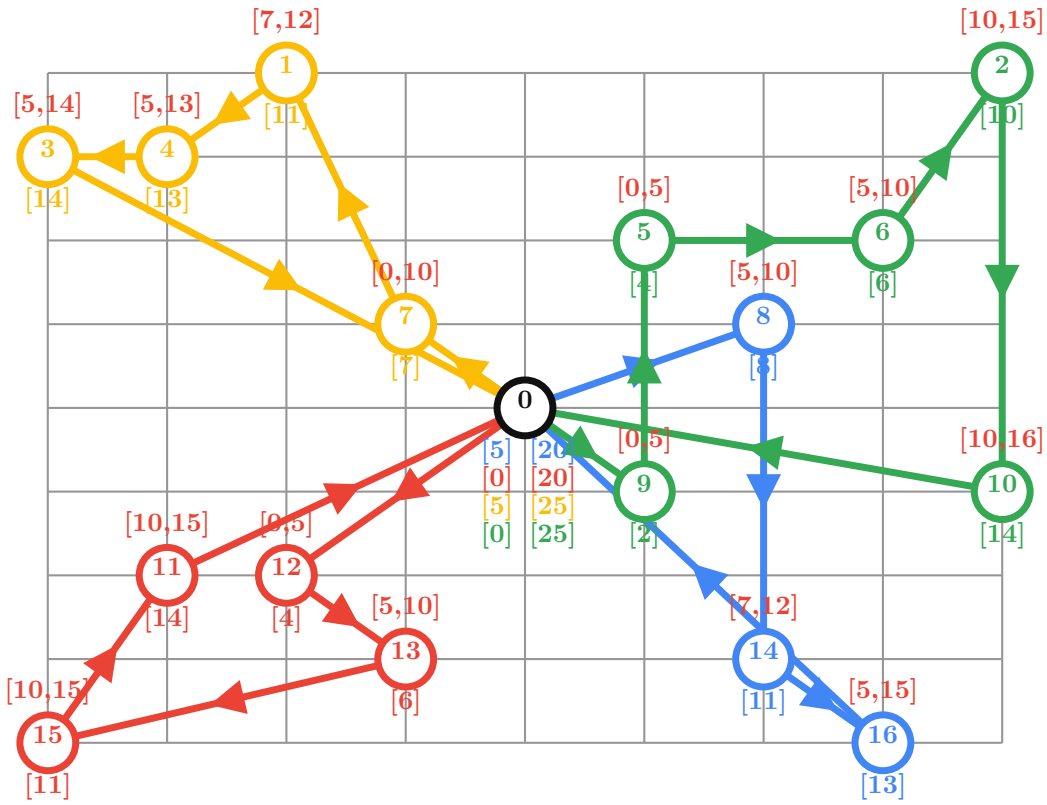

Figure 5: Possible solution of the VRPTW

3.2.2 Two-index Vehicle Flow Formulation for VRPTW

To use this formulation for the VRPTW we have to add additional constraints to the ones that we listed in Section 3.1.2. To do this, we need to define some new variables u_{ij} and v_{ij} in the following way: if $x_{ij} = 0$ then also $u_{ij} = v_{ij} = 0$, but if $x_{ij} = 1$, then u_{ij} represents the time that the vehicle starts its service at customer i, and v_{ij} represents the time that the vehicle starts its service at customer j. u_{0j} is set to 0 for all j. Then we get the following ILP:

Objective:

$$\text{Minimize} \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{13}$$

Subject to:

$$\sum_{j \in V \setminus \{i\}} x_{ij} = \sum_{k \in V \setminus \{i\}} x_{ki} = 1 \qquad (i \in V_c) \tag{14}$$

$$\sum_{j \in V_c} x_{0j} = \sum_{k \in V_c} x_{k0} = m \tag{15}$$

$$\sum_{j \in V \setminus \{i\}} u_{ij} = \sum_{j \in V \setminus \{i\}} v_{ji} \qquad (i \in V_c) \tag{16}$$

$$v_{ij} \geq u_{ij} + c_{ij} x_{ij} \qquad (i, j \in V_c; i \neq j) \tag{17}$$

$$u_{ij} \geq a_i x_{ij} \qquad (i \in V_c; j \in V; i \neq j) \tag{18}$$

$$v_{ij} \leq b_j x_{ij} \qquad (i \in V; j \in V_c; i \neq j) \tag{19}$$

$$u_{ij} \leq b_i x_{ij} \qquad (i, j \in V; i \neq j) \tag{20}$$

$$u_{ij} \leq (b_j - c_{ij}) x_{ij} \qquad (i, j \in V; i \neq j) \tag{21}$$

$$v_{ij} \geq a_j x_{ij} \qquad (i, j \in V; i \neq j) \tag{22}$$

$$v_{ij} \geq (a_i + c_{ij}) x_{ij} \qquad (i, j \in V; i \neq j) \tag{23}$$

$$x_{ij} \in \{0, 1\} \qquad ((i, j) \in E) \tag{24}$$

Equation (14) makes sure that each customer is visited and left by one vehicle, and Equation (15) makes sure that m vehicles leave and enter the depot. Next, Equation (16) links the start of service when travelling from i and the start of service when travelling to i together (they must be equal). Together these equations make sure that the flow of the vehicles is preserved.

Equation (17) ensures that the service time at j is at least greater or equal than the service time at i plus the time it takes to travel there. Moreover, Equations (18)-(19) make sure that the vehicles do not get served before the opening time or closing time of the city. Equations (20)-(23) make sure that vehicles depart before the closing time and arrive after the opening time. Together these equations make sure that the time windows are not violated.

Lastly, Equation (24) makes sure that all x_{ij} are indeed either 0 or 1. All equations together make sure that there are no sub tours allowed. [4]

3.3 Relation with Problem Description

The problem as described in Chapter 2 is very similar to a VRPTW. The differences are that all vehicles must return to the depot in a VRPTW, and this is not required in our problem. Moreover, if cities get infected multiple times, it is possible that a player must serve a city multiple times. This is not possible in a VRPTW. Lastly, in a VRPTW all vehicles can move simultaneously, while only one player can move in the game. Therefore, we cannot use the literature as it is. In the remainder of the report we will use the terms customers and vehicles instead of cities and players respectively.

4 Programming

To program the model, a combination of Python and CPLEX is used. Python is used to pre- and post-process information, and CPLEX is used to solve instances of the problem. CPLEX is a solver which provides flexible, high-performance mathematical solvers for linear programming, mixed integer programming, quadratic programming and quadratically constrained programming problems [5].

CPLEX is a rather complex program and it is sometimes hard to grasp what it is actually doing, but what it does use, are methods like branch-and-bound, branch-and-cut, and solvers for constrained programming.

Branch-and-bound and branch-and-cut are two similar methods that are used to approach a/the solution of an ILP, which are Linear Program (LP) in which the unknowns are integer. As we do not have to implement these methods ourselves, we will only give a short explanation of the methods so that we have a general idea of what these methods do. A branch-and-bound algorithm uses a systematic way to enumerate candidate solutions. It can be thought of as forming a rooted tree with the full solution set at the root, without integrality constraints. The algorithm explores *branches* of this tree, which represent subsets of the full solution set. These branches are created by fixing a variable to a valid integer value. Before continuing the search in this branch, it is checked against the lower and upper *bounds* of the solutions that were found this far. The lower bound is the value of the best integral solution that has been found so far, and the upper bound is the parent node of the branch that we are in. If the branch cannot produce a better solution than the ones that were already found, or if it is not feasible, it is discarded. If the branch is not integral yet and it *is* possible to find a better solution, then we branch again, and so on until we have found the optimal solution.

Branch-and-cut is very similar to this, but it uses cutting planes in addition to tighten the linear programming relations. [6]

5 The Models

In this chapter we will introduce the models that we will use. First the assumptions will be given in Section 5.1. Then, we will introduce the models in Sections 5.2-5.4, two models are based on the literature and the third model is a variation on the literature.

5.1 Assumptions

When we look at the formal problem description given in Section 2.3, some parameters are already known. We know that $n = 47$ as there are 48 cities on the board. Moreover, we assume that a game is played in at most 25 turns. As each turn takes four actions, we have at most 100 time steps, so $t_{max} = 100$. This means that $|T| = 101$, but there are only 100 moments a vehicle can move, as we are not allowed to move at the last time step. Lastly, we assume that $1 \leq m \leq 4$, even though the game states that you cannot play the game alone.

In the last five turns of a game, no more infections cards will be drawn. If no epidemic cards are used, we have depleted the infection deck. If epidemic cards are used, the deck has not been depleted but we stop infecting cities anyway.

Like mentioned before, an important assumption is that vehicles may not move at the same time. Instead they have to take turns, and therefore, the model must control the moments that vehicles move.

5.2 Set Partitioning Formulation

The first model is based on the set partitioning formulation. In this formulation, we will only use one vehicle, and we do not allow the repetition of infection.

For this formulation to work, we first need set of all feasible routes, R. To do this, we can use the recurrence in Equation (12). However, this recurrence is rather hard to solve. Instead, we can apply brute force. To make this easier we construct a new matrix E instead of using the adjacency matrix A. The matrix contains all the shortest paths from every city i to every other city j. This way, we can travel from every city to every other city without 'visiting' the other cities. Meaning that if we talk about c_{ij}, we mean the shortest distance from customer i to j. However, as our problem is NP-hard it might be the case that we cannot get to a conclusion within a reasonable amount of time. Therefore, some pruning has to be done, and even then, finishing within a sensible amount of time is not guaranteed. To prune we can use the following lemma:

Lemma 1. *Let i and j be two customers in a VRP, so that $a_i \geq a_j$, then the route $r = (0, \ldots, h, i, j, k, \ldots, 0)$ will never be used if $a_i \geq a_j + 2 \cdot c_{ij}$ and $w_i \geq 2 \cdot c_{ij}$.*

Proof. Let i and j be two customers in a VRP, so that $a_i \geq a_j + 2 \cdot c_{ij}$, be given in a route $r = (0, \ldots, h, i, j, k, \ldots, 0)$ so that $w_i \geq 2 \cdot c_{ij}$. Note that the distance from h to k in route r is equal to $c_{hi} + c_{ij} + c_{jk}$. Next, define $r' = (0, \ldots, h, j, i, k, \ldots, 0)$. We know that $c_{hj} \leq c_{hi} + c_{ij}$, and $c_{ik} \leq c_{ij} + c_{jk}$ by the triangle inequality (recall that c_{ij} is the shortest distance from i to j). The distance from h to k in route r' is equal to $c_{hj} + c_{ij} + c_{ik} \leq c_{hi} + 3 \cdot c_{ij} + c_{jk}$. So the difference between the route length of r and r' is at most $2 \cdot c_{ij}$. So when using route r' instead of r we reduce the waiting time at customer i by at most $2 \cdot c_{ij}$. It is possible that the waiting time at other customers increases due to this, but the total waiting time will never be higher than we had in route r. Therefore, route r' has less or equal total waiting time when compared to route r. Hence, route r will not be used as there is a route which has a shorter or equal length, namely route r'. (Recall that the length of a route is the distance that is travelled plus the waiting time.) □

This lemma hold for customers which are present in a route after one another. This can be extended to the case in which there are one or multiple customers in between them. The result is the following:

Lemma 2. *Let i and j be two customers in a VRP, so that $a_i \geq a_j$, then the route $r = (0, \ldots, h, i, x, \ldots, y, j, k, \ldots, 0)$ will never be used if $a_i \geq a_j + 4 \cdot c_{ij}$ and $w_i \geq 4 \cdot c_{ij}$.*

Proof. Let i and j be two customers in a VRP, so that $a_i \geq a_j + 4 \cdot c_{ij}$, be given in a route $r = (0, \ldots, h, i, x, \ldots, y, j, k, \ldots, 0)$ so that $w_i \geq 4 \cdot c_{ij}$. Next, define $r' = (0, \ldots, h, j, x, \ldots, y, i, k, \ldots, 0)$. We split the routes in two parts which we will look at, the part from customer h to x, and the part from customer y to k. The rest of the routes is exactly the same. The difference in the cost of the part from h to x when comparing r to r' is equal to $|(c_{hi} + c_{ix}) - (c_{hj} + c_{jx})|$. Either $c_{hj} + c_{jx} \leq c_{hi} + c_{ix}$, in which case route r' is shorter here. Or $c_{hj} + c_{jx} > c_{hi} + c_{ix}$, in which case we can construct a different route from h to x, namely, h, i, j, i, x in which case r' is only $2 \cdot c_{ij}$ longer than route r. Similar reasoning can be used for the part of the route from customer y to k. So the difference between the route length of r and r' is at most $4 \cdot c_{ij}$. So when using route r' instead of r we reduce the waiting time at customer i by at most $4 \cdot c_{ij}$. It is possible that the waiting time at other customers increases due to this, but the total waiting time will never be higher than we had in route r. Therefore, route r' has less or equal total waiting time when compared to route r. Hence, route r will not be used as there is a route which has a shorter or equal length, namely route r'. □

We can now apply brute force in the following way:

Algorithm 1 Brute Force

```
1: S ← V_c
2: r ← {0}
3: procedure BruteForce(S, r)
4:     if S = ∅ then
5:         r is a solution
6:         stop
7:     end if
8:     for i ∈ S do
9:         S ← S \ {i}
10:        r ← r ∪ {i}
11:        if r is feasible and lemmas do not apply then
12:            BruteForce(S, r)
13:        end if
14:        S ← S ∪ {i}
15:        r ← r \ {i}
16:    end for
17: end procedure
```

In Algorithm 1 the sets S and r are used. Set S is the set of customers that still needs to be used, while set r (in the pseudocode we treat r as a list) contains the route that has been made so far. Furthermore, the statement 'satisfies feasibility and pruning' is mentioned. Here we check if the route r is feasible until this point, meaning that all time windows are satisfied for every customer that we have visited in the route, and in addition if the route satisfies the two lemmas we just discussed.

This means that we only keep the route r is it serves all cities, but this can be easily changed in such a way that we can find the set of all feasible routes R. For instance by changing the input set for S. Moreover, it does not matter if we 'reject' a route based on the pruning by one of the lemmas. When this happens, we can use the alternative route found by the lemma instead of the route that was rejected. After this, we can apply the set partitioning formulation given in Section 3.1.1 to solve the model.

5.3 Shortest Path Formulation

In the second model, we again make use of the matrix E which contains all shortest paths from every customer to the others, as we do not allow repetition of visiting a customer. This model is based on the the two-index vehicle flow formulation as described in Section 3.2.2. But in contrast to the two-index vehicle flow formulation, we do not require vehicles to return to the depot at the end of their routes. Instead we will construct an extra customer. In addition, we have to change some constraints as we have to take into account the assumptions that were made, and the extra time window constraints. We do this by introducing two boolean slack variables sa_i and sb_i. When the value of a boolean is equal set to TRUE for $i \in V$, one of the time window constraints is violated for customer i. The ILP becomes the following:

$$\text{Minimize} \sum_{i \in V} sa_i + sb_i \tag{25}$$

Subject to:

$$\sum_{j \in V \setminus \{i\}} x_{ij} = \sum_{j \in V \setminus \{i\}} x_{ji} = 1 \qquad (i \in V_c) \tag{26}$$

$$\sum_{j \in V_c} x_{0j} = m \tag{27}$$

$$\sum_{j \in V_c} x_{j,n+1} = m \tag{28}$$

$$\sum_{j \in V \setminus \{i\}} u_{ij} = \sum_{j \in V \setminus \{i\}} v_{ji} \qquad (i \in V_c) \tag{29}$$

$$v_{ij} \geq u_{ij} + c_{ij} x_{ij} \qquad (i, j \in V_c; i \neq j) \tag{30}$$

$$u_{ij} \geq a_i x_{ij} - a_i sa_i \qquad (i \in V_c; j \in V; i \neq j) \tag{31}$$

$$v_{ij} \leq b_j x_{ij} + 100 \cdot sb_j \qquad (i \in V; j \in V_c; i \neq j) \tag{32}$$

$$u_{ij} \leq b_i x_{ij} + 100 \cdot sb_j \qquad (i, j \in V; i \neq j) \tag{33}$$

$$u_{ij} \leq (b_j - c_{ij}) x_{ij} + 100 \cdot sb_j \qquad (i, j \in V; i \neq j) \tag{34}$$

$$v_{ij} \geq a_j x_{ij} - a_i sa_i \qquad (i, j \in V; i \neq j) \tag{35}$$

$$v_{ij} \geq (a_i + c_{ij}) x_{ij} - a_i sa_i \qquad (i, j \in V; i \neq j) \tag{36}$$

$$x_{ij} \in \{0, 1\} \qquad ((i, j) \in E) \tag{37}$$

We will only explain the equations that are different from the ones in Section 3.2.2.

In Equation (28) you see the extra $n + 1^{th}$ customer. This is the artificially constructed customer and acts as a depot in which all vehicles must finish their route. Together, Equations (27)-(28) make sure that m vehicles depart from the depot and that they all arrive in this artificially constructed finish.

In Equation (31)-(36) the slack variables have been added to the constraints. These extra terms have been constructed in such a way, that if the slack variable is set to TRUE, the constraint will always be met. As the Objective (25) is to minimize the use of slack variables, we have found a solution to the problem if this is equal to 0, meaning we do not violate any constraints.

The model is useful in situations where vehicles are allowed to move simultaneously, but problems arise when this model is used in a game with more than one player. The major problem is that there is no way to control the moments that players are allowed to move. In a VRP, vehicles are allowed to move at the same time, but in Pandemic this is not the case. Therefore, a new model

will be introduced in the next section in which it is possible to control the moments vehicles may move.

When this model is used when repetition of infection is allowed, additional artificial customers have to be constructed for every customer that must be served multiple times, because a customer can only be visited, and therefore served, once in this formulation. This must be taken into account in matrix E as well by duplicating the corresponding row and column. Moreover, if a customer does not have to be served, the customer can be deleted, as well as the corresponding row and column in matrix E.

5.4 Adjacency Formulation

In the third model we use adjacency matrix A as cost matrix. Meaning that we now travel from a customer to one of its neighbors. This also means that we can visit customers multiple times. In addition to this, we put 1's on the diagonal of matrix A. This allows vehicles to move from a customer to the same customer, representing the possibility of waiting at a customer.

This third model is a variation on the two-index vehicle flow formulation, but in addition two more indices are added to the x_{ij} variables, namely p and t, resulting in the variables x_{ij}^{pt}. The variable indicates whether vehicle $p \in P = \{1, \ldots, m\}$ moves from customer i to j at time step $t \in T = \{0, \ldots, 99\}$ or not, with P the set of vehicles and T the set of all time steps. If $c_{ij} = 0$ with $i, j \in V$, then also $x_{ij}^{pt} = 0$ for all $p \in P, t \in T$. Furthermore, we introduce the parameter $tr_{p,t}$ indicating whether vehicle p is allowed to travel at time step t, and the parameter st_p which shows on which $t \in T$ every vehicle starts its first turn. And lastly, we make use of slack variables s_i, which indicate if a time window constraint is violated. This results in the following objective and constraints:

$$\text{Minimize} \sum_{i \in V} s_i \tag{38}$$

Subject to:

$$\sum_{i,j \in V} x_{ij}^{pt} = tr_{pt} \qquad (p \in P, t \in T) \tag{39}$$

$$\sum_{j \in V} x_{0j}^{p,st_p} = 1 \qquad (p \in P) \tag{40}$$

$$\sum_{i \in V} x_{ij}^{pt} = \sum_{k \in V} x_{jk}^{p,t+1} = 1 \qquad (j \in V, p \in P, t \in T : t \leq 98, tr_{pt} = tr_{p,t+1} = 1) \tag{41}$$

$$\sum_{i \in V} x_{ij}^{pt} = \sum_{k \in V} x_{jk}^{p,t+1+(m-1)\cdot 4} = 1 \qquad (j \in V, p \in P, t \in T : t \leq 98 - (m-1)\cdot 4, tr_{pt} = 1, tr_{p,t+1} = 0) \tag{42}$$

$$\sum_{i \in V,\ p \in P,\ t \in T} (a_j \leq (2 \cdot x_{ij}^{pt} - 1) \cdot (t+1) \leq b_j) + s_j \geq 1 \qquad (j \in V) \tag{43}$$

First of all, Equation (39) ensures that a vehicle moves if that vehicle is allowed to move on a certain time step, and that a vehicle does not move if it is not allowed to. Next, Equation (40) lets all vehicles start in the depot from the first moment a vehicle is allowed to travel.

Then, Equations (41)-(42) link the traveling from one city to another. Equation (41) tells us that if a vehicle travels from city i to j, and this vehicle is allowed to travel on the next time step also, then it must continue its route from city j. If the vehicle is not allowed to move the next time step,

we look at Equation (42). This equation makes sure that a vehicle continues moving from city j but at a later time step, dependent on the number of vehicles that is being used.

Finally, Equation (43) ensures that every city gets served by at least one vehicle in between the time window it was assigned. We need the construction $(2 \cdot x_{ij}^{pt} - 1)$ because a_j can be 0. If we simply used x_{ij}^{pt}, this constraint could also be met if the variable is equal to 0. In this equation we use the slack variable so the constraint may be violated if we set the slack variable to 1.

We do not have to check that every customer gets served (at least) once, because this is automatically done by Equation (43). Moreover, we do not have to check for in- and out-degree of every customer, because Equations (41)-(42) make sure that flow is preserved. In addition, vehicles do not have to return to the depot, this can also be seen in these two equations as we do not require vehicles to travel on a later time step than 99.

When this model is used in combination with epidemic cards, we must again construct additional artificial customers if a customer has to be served multiple times. This also has to be taken into account in matrix A by duplicating the corresponding row and column. In contrast to the previous model, we should not remove customers which do not have to be served, as this would mean that we cannot travel via this customer, eventually meaning that we cannot use certain edges anymore. Instead, we keep the time window $[-1, -1]$ for these customers, because Equation (43) will always be satisfied, as there will always be a moment that this city will *not* be visited by a vehicle.

6 Results

In this section of the report we will discuss the results of the three models that were described in Chapter 5. We will show the results of Test 1 (the basic example introduced in Section 2.2) if possible, and compare running times of the models with and without using repetition of infection in the same city.

6.1 Set Partitioning Formulation

Unfortunately, we did not successfully get any results from the brute force. After running for more than four hours, the brute force still had not found the entire set of feasible routes, we even ran into memory issues. Moreover, a constructed timer showed that it would take days before the brute force was finished. Therefore, we decided to stop the running of the brute force and move on to the next model, meaning that we could not use the set partitioning formulation.

6.2 ILP Formulation(s)

In this section we will present the results from the CPLEX models. We will look at both models, and compare running times.

6.2.1 Basic Example

A solution to Test 1 can be seen in the first two tables in Table 2 when using 1 vehicle. In these tables you see the exact path that the vehicle takes. The vehicle starts in the depot (customer 0) and serves all customers at least once in the time window of every customer. The first time the vehicle visits the customer within the time window is highlighted in bold. The result of this can be seen in the third table in Table 2.

6.2.2 Performance no Repetition of Infection

The performance of the two different models is tested by generating 10 different sets of time windows for all customers. These are solved by both models and the time it takes to solve the instances is compared. The running times of the tests can be seen in Tables 3-4. In the table you can see two running times. The first running time is the total running time, which includes the preprocessing that CPLEX automatically does. The second running time is the actual solving of the problem, by using branch-and-cut and other methods. We stopped running Test 2-5 for the shortest path formulation after four hours as this took too long and started running into memory issues. We only solve the first five instances with the shortest path formulation due to time issues. Recall that the running times of the shortest path formulation might be effected by the fact that we cannot control the moment of movement of the vehicles in this formulation.

6.2.3 Performance Repetition of Infection

The performance of the different models is again tested by generating 10 different sets of time windows. This time however, some customers need to be served multiple times, meaning that other customers will not need to be visited by a vehicle at all. Therefore, these customers have a time window of $[-1, -1]$ so we never have to visit it. Moreover, some customers have to be served multiple times. When generating the time windows, we only considered the situation in which we have 1 vehicle and 3 epidemic cards. The reason for this is to save time determining the running time, because as you can see in Table 4, running times can become very long. The resulting running times can be found in Table 5.

Time	Customer
0	**0**
1	28
2	0
3	0
4	0
5	0
6	0
7	**9**
8	**30**
9	9
10	0
11	0
12	0
13	0
14	0
15	**47**
16	28
17	0
18	9
19	**37**
20	**46**
21	46
22	46
23	**34**
24	**44**
25	44
26	44
27	**41**
28	**40**
29	**4**
30	41
31	44
32	**13**
33	**12**
34	**26**
35	44
36	13
37	26
38	44
39	26
40	**43**
41	**24**
42	9
43	0
44	0
45	0
46	0
47	0
48	0
49	0
50	0
51	47
52	**33**
53	47
54	33
55	33
56	**23**
57	**11**
58	**42**
59	11
60	**35**
61	**29**
62	**14**
63	42
64	**31**
65	31
66	**45**
67	**2**
68	**7**
69	**18**
70	**16**
71	**19**
72	16
73	18
74	19
75	**21**
76	21
77	**39**
78	**6**
79	**5**
80	**22**
81	**38**
82	38
83	22
84	**27**
85	**28**
86	47
87	33
88	**25**
89	**1**
90	7
91	**36**
92	**17**
93	**32**
94	**8**
95	**15**
96	**3**
97	3
98	**20**
99	8
100	**10**

Customer	a	b	t
0	0	100	0
1	60	100	89
2	40	100	67
3	24	100	96
4	8	100	29
5	72	100	79
6	68	100	78
7	56	100	68
8	28	100	94
9	0	100	7
10	32	100	100
11	48	100	57
12	20	100	33
13	16	100	32
14	44	100	62
15	16	100	95
16	60	100	70
17	36	100	92
18	64	100	69
19	64	100	71
20	24	100	98
21	68	100	75
22	76	100	80
23	52	100	56
24	0	100	41
25	56	100	88
26	12	100	34
27	0	100	84
28	76	100	85
29	52	100	61
30	0	100	8
31	40	100	64
32	28	100	93
33	0	100	52
34	8	100	23
35	48	100	60
36	32	100	91
37	0	100	19
38	80	100	81
39	72	100	77
40	4	100	28
41	4	100	27
42	44	100	58
43	20	100	40
44	12	100	24
45	36	100	66
46	0	100	20
47	0	100	15

Table 2: A possible solution to Test 1 when using 1 vehicle

Test	Shortest Path Formulation			
	1 vehicle	2 vehicles	3 vehicles	4 vehicles
Test 1	0:15.13/0:14.72	0:16.05/0:15.69	0:18.98/0:18.66	0:12.12/0:11.76
Test 2	4+ hours	1:11:48.22/1:11:47.89	0:12.94/0:12.61	0:17.06/0:16.75
Test 3	4+ hours	0:15.36/0:15.06	0:12.42/0:12.11	0:09.22/0:08.80
Test 4	4+ hours	1:00.54/1:00.20	0:19.99/0:19.59	0:15.77/0:15.44
Test 5	4+ hours	15:33.21/15:32.84	0:17.22/0:16.84	0:23.90/0:23.55
Average	unknown	17:46.68/17:46.34	0:16.31/0:15.96	0:15.61/0:15.26

Table 3: Running times of the different tests without repetition of infection when using shortest path formulation

Test	Adjacency Formulation			
	1 vehicle	2 vehicles	3 vehicles	4 vehicles
Test 1	4:40.81/0:27.53	26:07.26/0:31.19	49:56.75/0:29.86	1:02:39.87/0:29.77
Test 2	5:42.78/0:41.59	28:00.46/0:41.02	42:37.98/0:44.28	1:06:17.04/0:32.63
Test 3	7:18.35/0:56.77	19:56.82/0:41.19	41:50.95/0:39.08	0:58:32.17/0:29.91
Test 4	6:48.34/0:57.97	20:26.61/0:38.23	41:58.52/0:35.41	1:28:37.34/0:50.91
Test 5	7:57.62/1:12.44	19:10.64/0:30.33	40:49.34/0:38.34	1:16:06.80/0:37.51
Test 6	7:09.38/0:57.11	18:55.21/0:29.70	46:10.12/1:00.44	1:01:32.49/0:59.24
Test 7	7:07.87/1:00.20	18:51.29/0:37.73	54:10.61/0:34.78	1:09:13.71/0:37.39
Test 8	7:55.31/0:58.64	16:05.58/0:29.58	56:56.11/1:00.72	1:15:37.28/0:35.75
Test 9	7:49.30/1:06.89	16:47.12/0:49.44	47:14.85/0:30.02	1:21:25.86/0:46.19
Test 10	6:59.99/0:49.73	21:31.64/0:38.24	1:04:41.92/1:06.86	1:11:23.87/0:37.42
Average	6:56.97/0:54.89	20:29.47/0:36.67	48:38.72/0:43.98	1:10:30.84/0:39.67

Table 4: Running times of the different tests without repetition of infection when using the adjacency formulation

Test	Shortest Path Formulation	Adjacency Formulation
	1 vehicle	1 vehicle
Test 1	0:04.54/0:04.28	3:16.67/0:21.00
Test 2	0:01.66/0:01.55	2:51.94/0:16.66
Test 3	5:09.06/5:08.86	3:42.88/0:27.03
Test 4	0:00.82/0:00.69	2:27.59/0:18.63
Test 5	0:01.03/0:00.80	2:59.29/0:13.95
Test 6	0:00.99/0:00.80	2:56.31/0:24.05
Test 7	0:02.20/0:02.05	3:01.43/0:13.14
Test 8	0:01.85/0:01.66	2:55.07/0:18.95
Test 9	0:09.10/0:08.94	3:20.94/0:26.13
Test 10	0:02.99/0:02.77	3:25.86/0:40.13
Average	0:33.42/0:33.24	2:38.80/0:21.97

Table 5: Running times of the different tests with repetition of infection for both formulations

7 Discussion

In this chapter we will discuss the results that were presented in Chapter 6.

7.1 Brute Force

Sadly, there is not much to discuss, other than that we could not successfully find results for Test 1. The reason we did not try other tests, is because Test 1 was solvable for sure, and the other tests were generated at random. It might be the case that other tests can be solved much quicker, but this is due to the way the brute force is formulated. Therefore, it is heavily dependent on the input that is given. Moreover, it is possible that brute force is much faster if the set of customers V is smaller.

7.2 ILP Formulation(s)

First a short note on the running time of the second ILP formulation. When looking at Table 4 we can see a significant difference in total running time, and the time that CPLEX is solving the ILP. This can be explained by the huge amount of variables x_{ij}^{pt} that is used. These all have to be initialised, and this is one of the things that CPLEX does before it starts solving. We will now look at the two scenario's and the performance of the two formulations. Note that the running times could increase when other programs are running as well, and that we will not give the specifications of the hardware that was used, as we will compare the running times relatively between the two formulations.

7.2.1 No Repetition of Infection

When cities cannot get infected multiple times, it is immediately visible that the running times when using one vehicle are enormous when using the shortest path formulation. When using two vehicles, the shortest path formulation is faster, but not very consistent. However, when using more than two vehicles, it is clear that the shortest path formulation is much faster than the adjacency formulation when looking at the total running time, even when only considering the time that CPLEX is solving the model. Like mentioned before, this could be due to the fact that this formulation is not able to control the moments that the vehicles move, while these constraints are used in the adjacency formulation.

7.2.2 Repetition of Infection

When repetition of infection is allowed, the shortest path formulation is faster when considering the total running time. However, it is not always consistent with its faster running times as we can see in Test 3 in Table 5. But when we look at the actual time that CPLEX is solving the model, we see that the adjacency formulation is faster *on average*. When comparing the running times per test individually, the shortest path formulation is better. This could be explained by the size of the matrix that is used. The matrix that is used by the shortest path formulation shrinks compared to the scenario without repetition of infection. It shrinks, because there are more customers that can be eliminated than customers that have to be duplicated, this is also true when more epidemic cards are used. This is because of the fact that a city can be infected three times before it *must* be visited by a player. In contrast to the shortest path formulation, the matrix that is used by the adjacency formulation increases in size, because customers are not eliminated from this matrix. Otherwise you cannot use this customer to visit its neighbors. In addition, it is expected that the running time of the adjacency formulation increases even more when additional vehicles are used, as this is also visible in Table 4. Like mentioned before, a possible explanation for this is the number of variables x_{ij}^{pt}.

8 Conclusions and Recommendations

In this chapter we will give a conclusion on which model the Pandemic routing problem suits best, and what other research one could do in the future.

8.1 Conclusions

If we want to model the Pandemic routing problem as described in Chapter 2, it is best to use the adjacency formulation which is formulated in Section 5.4. As this formulation can make sure that vehicles only move at the times that they are allowed to. However, if this is not required, then the shortest path formulation is a better option in most situations. When using more vehicles, it is not only faster at solving the model, the initialization is significantly faster. In addition, when using more customers and vehicles, the adjacency formulation will take even more time to initialize, while the shortest path formulation barely takes any time to initialize. The big downside of this formulation however, is the situation in which one vehicle is used. In this situation it takes way too long to compute a possible route for the vehicle.

8.2 Future Research

As future research one could look into the shortest path formulation and try to make it in such a way that you can control the moments vehicles are allowed to move. If this is done, the model is of more value, and only then, running times can be properly compared when using multiple vehicles.

Also, one could try to alter the adjacency formulation in such a way that you do not have to duplicate customers when they have to be visited multiple times. This way, the matrix does not increase more, and this saves initialization time.

Moreover, infeasible time windows could be generated and the behaviour of the different formulations could be tested. Maybe a formulation is faster at determining whether an instance is infeasible, instead of determining whether it is feasible.

Lastly, one could try to use heuristics [7]. CPLEX also uses heuristics, but in combination with other methods. Heuristics are very useful when trying to compute a good result fast, but there is no guarantee that you find the best result. Which is what we have been aiming for, as we only want to know if a setting is feasible or not.

References

[1] Matt Leacock. Pandemic. URL https://images.zmangames.com/filer_public/53/ed/53edbee8-adfb-4715-899f-dd381e1420d7/zm7101_rules_web.pdf.

[2] Gilbert Laporte. What you should know about the vehicle routing problem. *Naval Research Logistics*, 54(8):811–819, 12 2007. ISSN 0894069X. doi: 10.1002/nav.20261.

[3] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354, 3 1992. ISSN 15265463. doi: 10.1287/opre.40.2.342.

[4] Adam N. Letchford and Juan José Salazar-González. Projection results for vehicle routing. *Mathematical Programming*, 105(2-3):251–274, 2 2006. ISSN 14364646. doi: 10.1007/s10107-005-0652-x.

[5] CPLEX Optimizer — IBM. URL https://www.ibm.com/analytics/cplex-optimizer.

[6] John E Mitchell. Branch-and-Cut Algorithms for Combinatorial Optimization Problems 1. Technical report, 1999. URL http://www.math.rpi.edu/~mitchj.

[7] G. Clarke and J. W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 12(4):568–581, 8 1964. ISSN 0030-364X. doi: 10.1287/opre.12.4.568.

[8] Marius M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987. ISSN 0030364X. doi: 10.1287/opre.35.2.254.

[9] Roberto Baldacci, Nicos Christofides, Aristide Mingozzi, R Baldacci, N Christofides, and A Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Math. Program., Ser. A*, 115(2):351–385, 10 2008. doi: 10.1007/s10107-007-0178-5. URL https://www.researchgate.net/publication/220589791.

[10] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *pubsonline.informs.org*, 39(1):104–118, 2005. doi: 10.1287/trsc.1030.0056. URL https://www.researchgate.net/publication/220413310.

A Appendix

City	City Number
Atlanta	0
Algiers	1
Baghdad	2
Bangkok	3
Beijing	4
Bogota	5
Buenos Aires	6
Cairo	7
Chennai	8
Chicago	9
Delhi	10
Essen	11
Ho Chi Minh City	12
Hong Kong	13
Istanbul	14
Jakarta	15
Johannesburg	16
Karachi	17
Khartoum	18
Kinshasa	19
Kolkata	20
Lagos	21
Lima	22
London	23
Los Angeles	24
Madrid	25
Manila	26
Mexico City	27
Miami	28
Milan	29
Montreal	30
Moscow	31
Mumbai	32
New York	33
Osaka	34
Paris	35
Riyahd	36
San Francisco	37
Santiago	38
Sao Paolo	39
Seoul	40
Shanghai	41
St. Petersburg	42
Sydney	43
Taipei	44
Tehran	45
Tokyo	46
Washington	47

Table 6: Numbers that will be used instead of the names of the city

City Number	a	b
0	0	100
1	60	100
2	40	100
3	24	100
4	8	100
5	72	100
6	68	100
7	56	100
8	28	100
9	0	100
10	32	100
11	48	100
12	20	100
13	16	100
14	44	100
15	16	100
16	60	100
17	36	100
18	64	100
19	64	100
20	24	100
21	68	100
22	76	100
23	52	100
24	0	100
25	56	100
26	12	100
27	0	100
28	76	100
29	52	100
30	0	100
31	40	100
32	28	100
33	0	100
34	8	100
35	48	100
36	32	100
37	0	100
38	80	100
39	72	100
40	4	100
41	4	100
42	44	100
43	20	100
44	12	100
45	36	100
46	0	100
47	0	100

Table 7: Example of time windows without using epidemic cards

City Number	a	b
0	0	100
1	8	100
2	-1	-1
3	0	100
4	48	100
5	-1	-1
6	0	100
7	-1	-1
8	-1	-1
9	4	100
10	44	100
11	-1	-1
12	[48, 64]	[60, 100]
13	[0, 76]	[72, 100]
14	-1	-1
15	-1	-1
16	-1	-1
17	36	100
18	40	100
19	-1	-1
20	-1	-1
21	-1	-1
22	[8, 68]	[64, 100]
23	36	100
24	-1	-1
25	[0, 32]	[28, 100]
26	-1	-1
27	32	100
28	0	100
29	-1	-1
30	-1	-1
31	-1	-1
32	-1	-1
33	-1	-1
34	40	100
35	-1	-1
36	4	100
37	-1	-1
38	[0, 52]	[48, 100]
39	-1	-1
40	44	100
41	-1	-1
42	-1	-1
43	-1	-1
44	32	100
45	[0, 20]	[16, 100]
46	-1	-1
47	-1	-1

Table 8: Example of time windows when using epidemic cards