# Circulation Management In Hospitals During COVID-19

by

Sana Alsadat RAZAVI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN ELECTRICAL ENGINEERING
M.A.Sc.

MONTREAL, "MARCH 3, 2023"

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

**BOARD OF EXAMINERS**

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Kim-Khoa Nguyen, Thesis supervisor
Department of Electrical Engineering, École de technologie supérieure

Mrs. Brigitte Jaumard, Thesis Co-Supervisor
Department of Computer Science and Software Engineering, Concordia university

M. Aris Leivadeas, Chair, Board of Examiners
Department of Software and IT Engineering

Mrs. Bassant Selim, External Examiner
Systems Engineering Department

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE

PUBLIC ON "FEBRUARY 27, 2023

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

## ACKNOWLEDGEMENTS

This work would not have been possible without the support of many people. First and foremost, I would like to express my honest gratitude to my supervisor, professor Kim Khoa Nguyen and professor Brigitte Jaumard for their guidance and strong support throughout my studies. I have learned and experienced new things while working with them.

Last but not least, I would like to thank my husband, Saman Abasi Ahwazi, for accompanying me during my study and my family for their support and encouragement.

**Gestion De La Circulation Dans Les Hôpitaux Pendant Le COVID-19**

Sana Alsadat RAZAVI

## RÉSUMÉ

Pendant la pandémie de COVID-19, la distanciation sociale a été mise en œuvre dans le monde entier pour réduire le risque d'infection. Cette exigence demande une nouvelle stratégie de gestion de déplacements dans les couloirs des hôpitaux pour éviter les croisements des passagers. Dans ce mémoire, nous étudions le problème de routage et d'acheminement des passagers dans l'hôpital pendant la pandémie pour deux flux de patients, COVID-19 et Non-COVID, en utilisant autant de chemins disjoints que possible.

Nous proposons deux scénarios pour modéliser le traffic et colorer des couloirs en utilisant la théorie des graphes, ensuite les comparer dans des simulations. Le premier scénario utilise une méthode hors ligne pour définir des directions pour les patients lorsqu'ils n'ont pas l'accès à un téléphone portable ou à d'autres technologies intelligentes. Le deuxième scénario est une méthode en ligne basée sur une technologie intelligente et une localisation en temps réel. Nos données se composent d'un ensemble de patients Non-COVID avec des rendez-vous prédéfinis dans différentes cliniques et d'un autre ensemble de patients COVID-19/Non-COVID arrivant aux services d'urgence avec un taux d'arrivée donné. Nous évaluons la performance de ces deux scénarios avec de différents taux d'arrivée des patients COVID-19. Le résultat de la simulation montre que les solutions proposées son meilleures qu'une solution qui se base sur l'algorihtme Dijkstra. Les résultats de ce mémoire peuvent contribuer effectivement à l'amélioration de la sécurité en définissant des directives pour les patients COVID-19 et Non-COVID dans les centres de santé pendant la pandémie.

**Mots-clés:** COVID-19, TSP, optimisation des flux, path-finding, algorithme de Christofides

**Circulation Management In Hospitals During COVID-19**

Sana Alsadat RAZAVI

## ABSTRACT

During the COVID-19 pandemic, social distancing has been applied worldwide to reduce the risk of infection. In hospitals, this requires a new strategy for circulation management in the corridors to avoid cross-overs. In this thesis, we study the problem of routing and path-finding in the hospital during the pandemic for two flows of patients, COVID-19 and Non-COVID, using as many disjoint paths as possible.

We present two scenarios to model the routing and coloring of the hospital paths and compare them in simulations. The first scenario uses an offline method to establish a guideline for the patients when they do not have access to a cellphone or other smart technologies. The second scenario is an online method based on smart technology and real-time localization. Our dataset consists of a set of Non-COVID patients with scheduled appointments in different clinics and COVID-19/Non-COVID patients arriving at the emergency unit with a given arrival pattern. We evaluate the performance of the two proposed scenarios subject to different arrival rates of COVID-19 patients.

The simulation result confirms that the proposed solutions outperform a baseline which is based on the Dijkstra algorithm. This thesis outcome will help increase safety by considering the guideline for COVID-19 and Non-COVID patients in healthcare centers during the pandemic.

**Keywords:** COVID-19, TSP, flow optimization, path-finding, Christofides algorithm

## TABLE OF CONTENTS

Page

## LIST OF TABLES

Page

## LIST OF FIGURES

Page

## LIST OF ALGORITHMS

Page

## LIST OF ABBREVIATIONS

| | |
|---|---|
| RFID | Radio Frequency Identification |
| RTLS | Real-Time Location System |
| GPS | Global Positioning Systems |
| VRP | Vehicle Routing Problem |
| TSP | Travelling Salesman Problem |
| GTSP | General Travelling Salesman Problem |
| SO | Sub-Objectives |
| SPT | Shortest Path Tree |
| MST | Minimum Spanning Tree |
| MAS | Maximum Acyclic Subgraph |
| MFAS | Maximum Feedback Arc Set |
| NP | Nondeterministic Polynomial |
| OPWS | Orienteering Problem with Random Weights |
| EOPODP | Edge Orientation Problem with Origin-Destination Pairs |
| OD | Origins and Destinations |
| LP | Linear Programming |
| DFS | Depth-first Search |
| DSU | Disjoint Set Union |
| NNM | Nuclear Norm Minimization |

| | |
|---|---|
| PMF | Probabilistic Matrix Factorization |
| HVC | Hilbert Value Clustering |
| LHVC | Leaves Hilbert Value Clustering |
| RSPT | Random Shortest Path Tree |
| SP | Shortest Path |
| TSXP | Top X Percentage |
| XNN | X Nearest Neighbor |
| SD | Source and Destination |
| ED | Emergency Department |
| ABM | Agent-Based Model |
| COPS | Chemotherapy (cyclophosphamide, Oncovin, and prednisone) |

## LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

| | |
|---|---|
| s | second |
| m | meter |
| ms | millisecond |
| min | minute |

# INTRODUCTION

## Context and motivations

In response to COVID-19, hospitals needed to make swift changes to their physical environments and usual protocols in order to minimize infection risk (Lancaster *et al.*, 2020). Several hospitals manage patient flow in the early days of the pandemic with a service design approach to way-finding, i.e., the entire patient journey from finding out about their appointment to walking out of the hospital doors. This implies behaviors that were new at the time, such as physical distancing, mandatory hand sanitizing, universal masking, and symptom screening of large crowds during peak hours.

Also, it requires movements in the corridors that reduce the number of crossings of people and satisfy the requirements of social distancing. During COVID-19, even people without signs of COVID-19 need to respect social distancing. As a result, we need to take into account social distancing also for Non-COVID patients to reduce the risk of disease spread. As many corridors in close environments are not wide enough, it is not easy to manage the distance between people, so there is a need for one-way pathways as much as possible to guarantee safe distancing. It is imperative that hospitals and other healthcare facilities improve patient flow as part of their process management efforts.

During the COVID pandemic, a strategy adopted by many hospitals is using different colors to present the COVID conditions of patients. This helps manage the priority of circulation in the hospital. Therefore, in this thesis, we assume two levels of priority: Green for patients without COVID-19 and Red for patients with COVID-19. When red and green patients are in the same corridor, the red patients have the right to move before the green patients.

**Definition 1:** The patient circulation in a hospital with priority refers to the movement of patients from one area to another based on their priority level within the hospital. We propose to reduce

the patient circulation problem to a graph problem in which the first step is to build a graph out of the floor map of a hospital; see Fig. 0.1 for an illustration. Nodes are two types: the first type

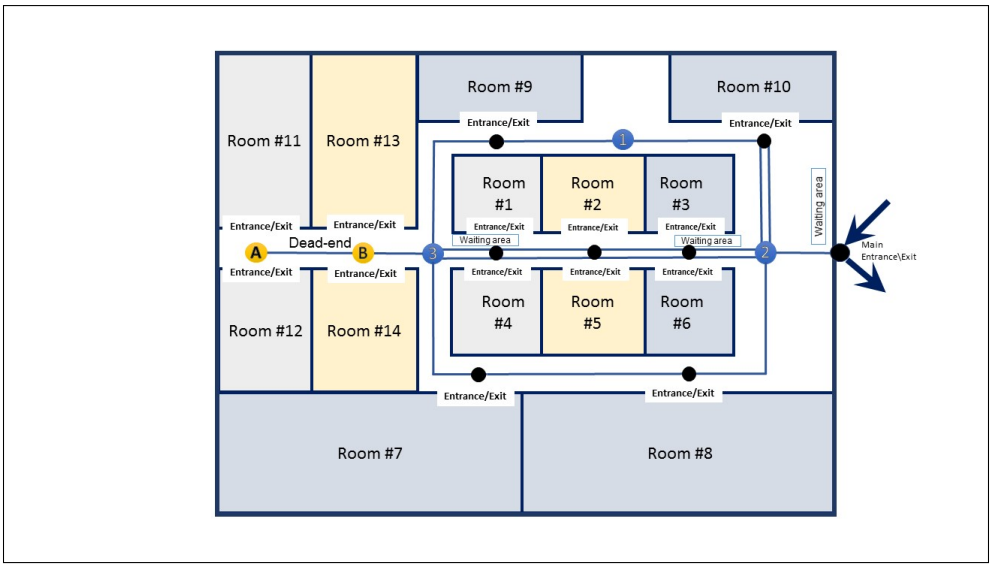


Figure 0.1 Example of floor map

includes the nodes associated with corridor intersections (e.g., nodes 2 and 3), and the second type includes the nodes associated with the entrance/exit of wards/clinics where patients need to go (e.g., black nodes, and nodes A and B). Nodes in the corridors are connected by edges.

We assume there is a waiting area with limited capacity (few patients) at every node for simplicity (in practice, it may be the case only at some nodes). In our work, every elevator, escalator, and stair is considered as a bridge to simplify the graph and merge the nodes. For example, the elevator on the first floor is a destination, and the second floor is the new starting point for the patients. We also assume each flow of patients has source and destination node pairs for their trips through the hospital. For instance, for Non-COVID patients, typical travels are from the general hospital entrance to a clinic, or from one clinic to another one, or from a clinic to the general exit of the hospital. For COVID-19 patients, the possible travels are from the emergency entrance to the emergency waiting room, or the emergency waiting room to CT-scan or X-ray, or from the emergency unit to a COVID-19 unit.

**Definition 2:** A Patient Flow refers to the physical movement, having the same priority along the same direction in the corridors of the hospital.

In our work, based on the priority of the patients, we consider two flows: Red flow and Green flow. In reality, a Red flow includes patients with COVID-19 symptoms, people who accompany them, and staff working with them; and a Green flow consists of patients without symptoms, visitors, and staff of Non-COVID wards.

**Definition 3:** The waiting time of a patient circulation is the sum of three components: 1) the time the patient waits to get access to the pathways in the hospital, 2) the total time the patient spends in the pathways, and 3) the time the patient spends in the waiting areas, from the entrance to discharge. This waiting time does not include the time the patient spends in the wards and being seen by the medical staff.

Patient waiting time is an important indicator of the quality of services offered by the hospital (Oche & Adamu, 2013). Also, it can be used to estimate the number of nurses required for patient care and manage the patient flow.

A traditional approach to avoiding congestion when managing traffic in streets is to define certain streets one-way (Robbins, 1939). Relying on the same approach, in this thesis, we examine whether it is possible to make certain designated corridors one-way and how it can be done while ensuring that it is possible to get from one place to another, with respect to the additional constraint of COVID safe distance.

The problem of defining one-way corridors in the hospital can be formulated as an orientation problem in an undirected graph theory by considering the corridor corners and the wards as vertices and drawing edges to connect them if the graph is strongly connected. An orientation problem can easily be defined in a graph; a graph is said to be strongly connected if every vertex is reachable from every other vertex (Roberts, 1978).

Therefore, the problem of COVID flow management in a hospital consists of two steps: i) Building an undirected graph from the hospital map floor, and ii) Defining an orientation in the graph. In the first step, the built graph should not contain a bridge. In an undirected connected graph, a bridge is defined as an edge if removing it disconnects the graph (e.g., dead-end, where one must return the same way).

In our work, we need tracking patients in corridors with smart technologies, which requires a tracking method, and it can be done in several ways. A person tracking system employs unique identifiers that are temporary, e.g., Radio Frequency Identification (RFID) tags or the real-time location system (RTLS). The real-time location system (RTLS) identifies and tracks objects or people in real-time, usually within a building. A wireless RTLS tag or RFID tag can be attached to patients and report their location regularly to a server computer (Miller *et al.*, 2006).
People also can be tracked permanently using personal identifiers (including biometric identifiers) or national identification numbers and by sampling their positions in real-time using Global Positioning Systems (GPS). In addition, The network-based infrastructure can be used to determine the location of a mobile phone. The advantage of network-based techniques is that they can be implemented non-intrusively without affecting handsets from a service provider's perspective. A mobile phone's location can be determined by various means, including handset-based, Wi-Fi, and hybrid positioning systems.

**Definition 4:** The patient routing involves finding a path from all sources to destinations of both Red flow and Green flow within known built environments by considering the requirements of each flow.

The patient routing, aimed at finding routes for patients, can be similar to the vehicle routing problem (VRP) (Toth *et al.*, 2002). VRP, which is generalized to the traveling salesman problem (TSP), involves delivering patients from one or more areas with a set of start points and guiding them on a route network to the set of destinations. The objective is to determine a set of routes

that will meet all patients' needs and operating constraints while reducing global transportation costs.

A traveling salesman problem (TSP) is a well-known routing problem in which a salesman wishes to visit every city once and return home at a minimal cost. However, in our problem, it might be necessary to visit a node more than once. Therefore, a more general problem called the General Traveling Salesman Problem (GTSP) should rather be applied. The GTSP is aimed at finding a minimum-travel-time closed tour for a salesman traversing a given part of nodes in a network at least once (He *et al.*, 2021). Compared to traditional TSP, GTSP has a more significant number of real-world applications, including planning tourist routes, meter readings, and goods distribution. An essential requirement to using the GTSP is that we need a **complete graph**, in which a unique edge connects every pair of distinct vertices.

In our work, we study routing in the hospital during the pandemic, which aims to find the maximum number of disjoint paths in the network. Because a tracking system may not be available in a hospital, we present two scenarios to model the routing and coloring of the hospital paths and compare them using the simulation tool. The first scenario uses an offline method to establish a guideline for the patients when they do not have access to a cellphone or other innovative technologies. Scenario 2 is an online method based on intelligent technology and real-time location. Our goal is to compare two scenarios in terms of their waiting time subject to different arrival rates for COVID-19 patients.

**Challenges**

In order to minimize the impact of COVID-19 on hospital visitors\patients traffic, it is challenging to have a strategy to manage the flows in the corridors of the hospital. There are some related challenges in terms of orientation and the patient routing problem as follows.

- It is not always possible to have all corridors one-way in a hospital due to two flow requirements and to maintain a safe distance between two flows of patients, so we need to keep some corridors two-way (with undirected edges). This makes the problem more complicated because we cannot have the same corridors for both patient flows.
- It is common for people to pass through the same paths, such as dead-end rooms, elevators, and stairs. Therefore, it is necessary to plan and schedule the use of the common pathway for the two flows. Due to the lack of paths connecting one end of the bridge to the other, a graph with a bridge cannot be strongly orientable (Robbins, 1939). Therefore, dealing with the wards with dead-ends and finding the orientation for two flows of patients will be challenging. The existing algorithms are unsuitable for solving two flow problems; therefore, a new algorithm should be designed to solve the problem effectively.
- The GTSP can be modeled as an undirected weighted graph, such that the desired wards are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. The model is often a complete graph (an edge connects each pair of vertices). We need to build a graph to connect two wards when there is no direct path between them, and completing it without affecting the optimal tour in GTSP is challenging. The way we complete the graph must have a minimum impact on other flows in GTSP.

**Research Questions**

During a pandemic, current path-finding solutions cannot be used in hospitals because they do not consider different patient flows. Although existing indoor routing methods can help with indoor navigation based on the explicit topology of buildings, they have not been designed to address different types of users and the distance between them. Regarding patients technological capability, we design two routing solutions for two different scenarios: with and without smartphone support. This requires us to answer the following research question:

- **RQ.** Between the two proposed scenarios, which would be more efficient in terms of the average waiting time for circulation management in the hospitals? In other words, can we really take advantage of personal devices, such as smartphones, to guide patients through hospital corridors with respect to social distancing requirements during the CoVID-19 pandemic?

**Objectives of the thesis**

The main objective of the thesis is to propose a new solution to model Patient Flow during COVID-19 based on constraints caused by the pandemic. This main objective can be divided into two sub-objectives (SO).

- **SO1**. Propose an efficient algorithm for path-finding and coloring for scenario 1, and then calculate the average waiting time.
- **SO2**. Propose an efficient algorithm for path-finding and coloring for scenario 2, and then calculate the average waiting time, and finally compare it to scenario 1.

**Thesis organization**

This thesis includes an Introduction, three chapters, and a conclusion. The Introduction includes the general context and the challenges. It also contains motivations for this thesis, followed by the thesis objectives.

In Chapter 1 we review the related work on path-finding and similar technologies for patient flows. We also reviewed different resource provisioning techniques in the graph orientation, such as TSP.

Chapter 2, we reformulate the design of an optimized patient flow in corridors as a graph orientation problem under two different traffic scenarios. At first, we present Scenario 1 in order

to meet different patient flows. Then we present Scenario 2, which assumes each patient uses either a smartphone or pager to display the direction to follow at each corridor intersection.

Chapter 3 presents the simulation results of both scenarios. we also compare them in terms of waiting time.

The Conclusion summarizes the thesis findings and presents possible future work.

# CHAPTER 1

# LITERATURE REVIEW

The purpose of this chapter is to review Methods for Converting a Map to a Graph, which can use to convert the map of the hospital to the network. We also review the current solutions for the Pedestrian Routing problem, specifically the Orientation Problem and the Travelling salesman problem. In Table 1.1, we present a summary of current papers studied on the Travelling Salesman Problem. Also, we review the related algorithms and the ways to evaluate and improve strategies.

## 1.1 Converting a map floor to a graph

The navigation graph is considered to be the basic structure for guiding movement within indoor spaces (Zhou *et al.*, 2022). This graph has edges that represent routes and nodes that represent locations where information is shared or displayed. This section reviews previous research on how maps are transformed into graphs. It also examines different models for indoor spaces and compares various graph-based models for navigation.

A planer graph is a type of graph in which the edges are connected in such a way that no two edges intersect (Mahapatra *et al.*, 2021). This means that if you were to draw the graph on paper, you would not need to lift the pencil from the paper to draw the edges.

The hospital map could be a planer graph if it is designed in such a way that all the edges are connected in a way that does not intersect. But it could also be a non-planer graph if the edges intersect or if the number of edges is greater than the number of nodes.

It is also possible that the hospital map is a combination of planer and non-planer areas, depending on the design of the hospital and the layout of the buildings and corridors. A hospital map can be non-planar if the layout of the buildings and corridors results in edges that intersect. For example, if a hospital has multiple buildings with multiple floors and there are corridors that connect the different buildings and floors, the edges representing these corridors may intersect

with one another. Another example could be if the hospital has a complex layout with multiple interconnected buildings and multiple levels. The edges representing the paths between different rooms or wards could intersect with one another, making it a non-planer graph.

There are several ways to transform a non-planar graph into a planar graph (Harrigan *et al.*, 2021). One way to embed a non-planar graph is to use a technique called "edge contraction," where two nodes that are connected by an edge are merged into a single node (Akiba *et al.*, 2013). This reduces the number of edges and can make the graph planar. However, this method can change the structure of the graph and may not be suitable for some applications. Another way is to use a technique called "edge deletion" where certain edges are removed from the graph to make it planar. This method also can change the structure of the graph, but it may be more suitable for some applications.

In summary, there are several methods to transfer non-planer graphs to planer graphs, such as edge contraction, and edge deletion. The choice of method depends on the specific requirements of the application and the desired properties of the planar graph. In any case, whether the hospital map is a planer graph or not is not the most important aspect of the problem as the main focus is on finding the most efficient path for the flow of patients through the network. Even if the map is non-planer, the goal is still to find the best path for patients to travel through the hospital.

Different ways of transforming a floor plan to a graph are also discussed by (Franz *et al.*, 2005), and shown in Figure 1.1. The **place graph model** is the appropriate topology for converting the floor map to the graph in our research.

This model clearly defines rooms or labeled places for the nodes, while graph edges indicate their connectivity. As an additional element of the justified graphs, accesses to the analyzed spatial configurations were considered root nodes. (Jensen *et al.*, 2009) proposed a uniform graph model framework for indoor tracking consisting of base and deployment graphs. They considered connectivity and accessibility, the base graph represents the topology of a possible

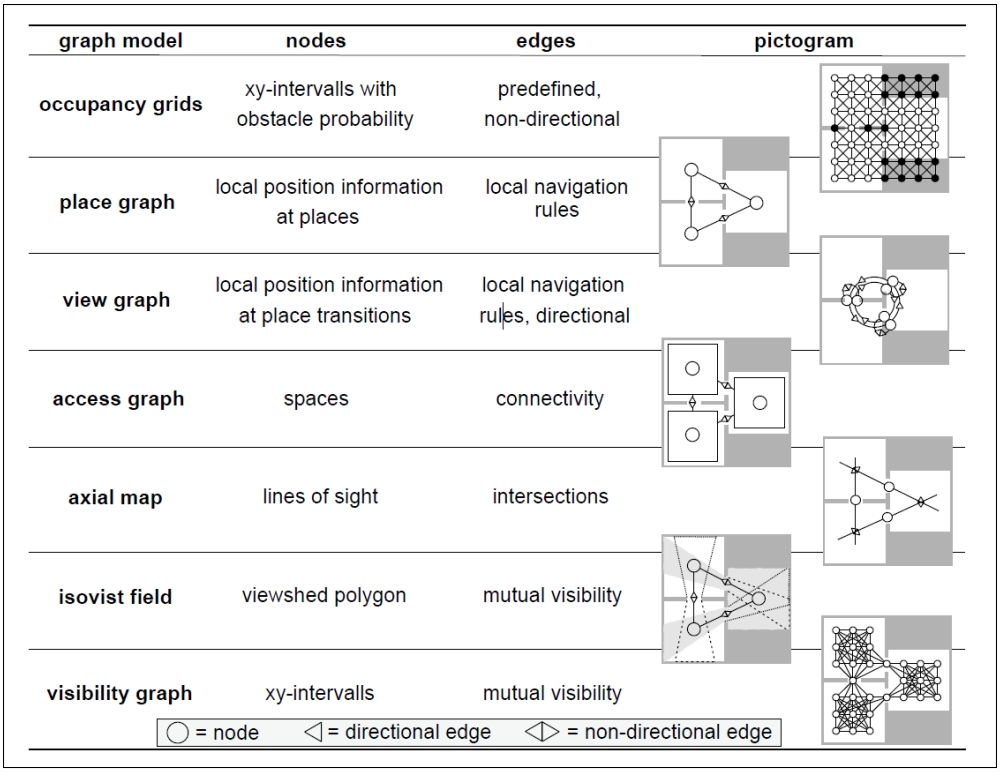


Figure 1.1 Overview of map converting to graphs
Taken from Franz *et al.* (2005)

complex indoor space plan. As a running example, they use the floor plan in Figure 1.2 for simplicity. The figure shows doors, walls, a staircase, a hallway, and rooms.

## 1.2 Graph theory

The graph is a discrete structure that is frequently used to model real-life problems, including communication networks, social networks, and biological networks (Majeed & Rauf, 2020). Graph theory has numerous applications, including computer science, sociology, and chemistry since they are simple yet effective in real-life modeling phenomena. A lot of problems we encounter every day could be paraphrased into a graph problem or a near-similar sub-problem. So it is required to have some familiarity with different graph problems related to our work and to discuss some algorithms associated with them.

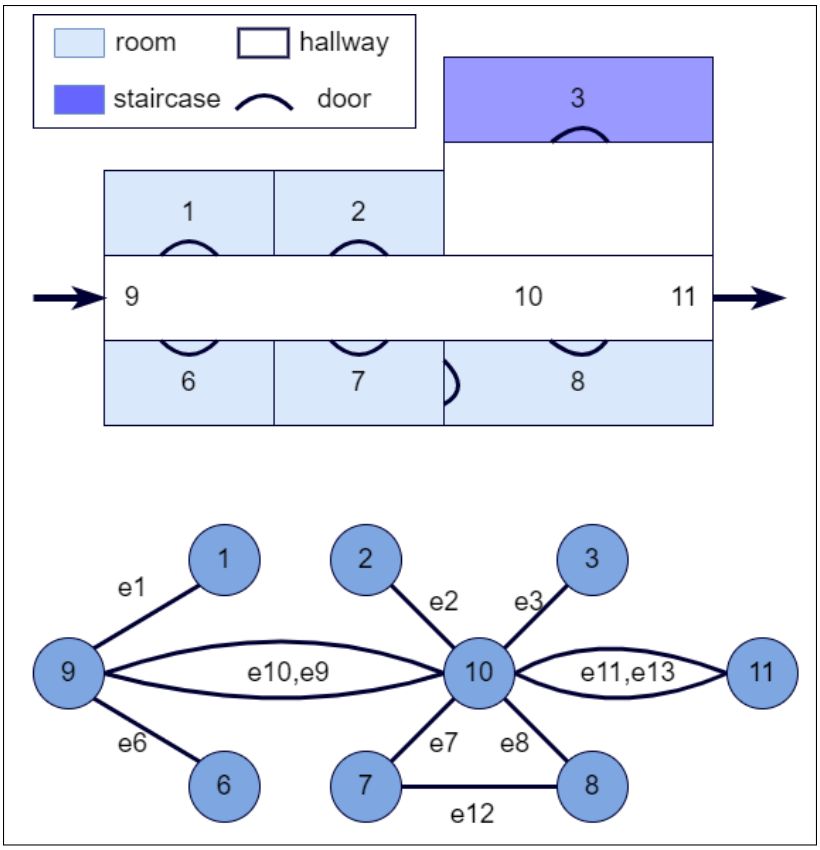


Figure 1.2 Floor plan and connectivity graph

- Simple Graph: A simple graph is a mathematical representation of a set of objects (called vertices or nodes) and the connections (or edges) between them.
- Complete Graph: A complete graph is a simple graph in which an edge connects every pair of distinct vertices. The complete graph on $n$ vertices has $n$ vertices and $n(n-1)/2$ edges. In a complete graph, each pair of distinct vertices is connected by an edge.
- Shortest Path: A path, denoted as $P_{uv}$, from vertex $u$ to vertex $v$ in a graph $G$ means that vertex $v$ is reachable from $u$, and vertex $u$ is traceable back from $v$. All vertices that can be reached from $u$, including $u$ itself, in $G$ are considered to be descendants of $u$, represented as $des(u)$. Similarly, all vertices that can be traced back to $v$, including $v$ itself, in $G$ are considered to be ancestors of $v$, represented as $anc(v)$.

  A path, denoted as $P_{uv}$, is considered the shortest path, represented as $SP_{uv}$, if it is shorter than any other possible path $P^*_{uv}$. For any vertex $v$ in the set of vertices $V$, there could be

more than one shortest path from a source $s$ in the graph $G$, and all the shortest paths from $s$ to $v$ will have the same shortest distance. The shortest distance from vertex $u$ to vertex $v$ in $G$ is denoted as $d_{uv}$.

- Shortest-Path Tree (SPT): In a directed graph $G = (V, E, w)$, a tree rooted at a source vertex $s$, represented as $T_s$, is considered a shortest path tree (SPT) if, for all vertices $v$ that are not equal to $s$ and are descendants of $s$ in $T_s$, the tree contains the shortest path $SP_{sv}$ from $s$ to $v$. Because of this structure, $T_s$ only contains one shortest path $SP_{sv}$ for each vertex $v$.
- Minimum Spanning Tree (MST): Given a digraph $G = (V, E, w)$, an acyclic subset $T \subseteq E$ is a minimum spanning tree (MST) if it connects all of the vertices and whose total weight $w(T) = \Sigma_{(u,v)\in T} w(u, v)$ is minimized. Since $T$ is acyclic and connects all of the vertices, it must form a tree.
- Hamiltonian Cycle: A Hamiltonian cycle is a cycle in a graph that passes through every vertex exactly once, visiting each vertex only once before returning to the starting point.

## 1.3 Pedestrian routing problem

The pedestrian routing problem is an important and active area of research in the field of transportation and network science (Tong & Bode, 2022). It involves finding the most efficient or shortest path for pedestrians to travel between two points in a network, taking into account various factors such as sidewalk availability, crossing signals, terrain, and accessibility for individuals with disabilities.

One approach to solving the pedestrian routing problem is to use traditional graph-based algorithms such as Dijkstra's algorithm or A* search (Dib, Manier & Caminada, 2015). These algorithms work by considering the nodes and edges in a network as a graph and then searching for the shortest path between two points by traversing the edges and accumulating a cost based on various factors, such as distance or travel time.

Another approach is to use heuristic-based methods, which are designed to find near-optimal solutions quickly. Examples of heuristic methods include genetic algorithms, simulated annealing, and ant colony optimization (Liu, Yi & Ni, 2013).

Recent research has also explored using machine learning techniques to improve pedestrian routing. For example, using deep neural networks to predict pedestrian behavior and traffic flow or using reinforcement learning to adapt routing strategies in real time based on traffic conditions.

Additionally, there are some studies that have proposed using GIS data to enhance the pedestrian routing problem (Trindade *et al.*, 2018). These studies have used data such as building information, elevation, and land use to improve the accuracy and efficiency of routing algorithms.

Overall, the pedestrian routing problem is a complex and multi-disciplinary problem that requires a combination of techniques from transportation engineering, computer science, and GIS.

## 1.4 Orientation problem

The graph orientation problem is the task of assigning orientations (i.e., directions) to the edges of an undirected graph such that certain properties are satisfied. This problem has been studied in various contexts, including algorithms for solving it, complexity and computational hardness results, and applications in areas such as computer science, operations research, and electrical engineering.

One of the most well-known algorithms for solving the graph orientation problem is the maximum acyclic subgraph (MAS) algorithm, which aims to orient the edges of the graph such that the resulting directed graph is acyclic and has the maximum number of edges (Dutta & Subramanian, 2016). This algorithm has been widely studied and has been shown to have polynomial time complexity for certain classes of graphs.

Another commonly studied algorithm is the maximum feedback arc set (MFAS) algorithm, which aims to orient the edges of the graph such that the resulting directed graph has the minimum

number of directed cycles, or feedback arc sets (Kudelić & Ivković, 2019). This algorithm has been shown to be NP-hard in the general case, but polynomial-time algorithms exist for certain classes of graphs.

The graph orientation problem has been applied in many areas, including scheduling and resource allocation, social networks, and constraint satisfaction. In scheduling and resource allocation, the problem of orienting the edges of a graph is used to model a set of tasks that need to be completed, with edges representing dependencies between tasks. In social networks, graph orientation is used to model the flow of information or influence between individuals. In constraint satisfaction, the problem is used to model the relationships between variables and constraints in a problem.

The Orienteering Problem with Random Weights (OPSW) was studied in (Sniedovich, 2010) to find the optimal path or loop by taking into account the total weight of the arc with capacity constraints. A two-stage stochastic model was developed to solve large instances of the problem. Heuristics were also used to address uncertainty based on the OPSW structure, which allows for the addition of additional nodes during runtime.

(Duraj, 2010) research focused on graph orientation problems in order to provide the best reachability[1]. The goal is to maximize the number of pairs of vertices connected by a directed path (also called simply connected pairs). Figure 1.3 presents an example graph with two orientations.

In the first orientation, some pairs of vertices are left unconnected; in the second orientation, all pairs of vertices are connected. The characterization of graphs with an orientation connecting all vertices was the unsolved question. (Robbins, 1939) provided a solution that is now known as Robbin's Theorem: such orientation exists only when there are no bridges in the graph. Since algorithms and complexity were unknown at that time, Robbins' construction can be easily

[1] reachability refers to the ability to get from one vertex to another within a graph.

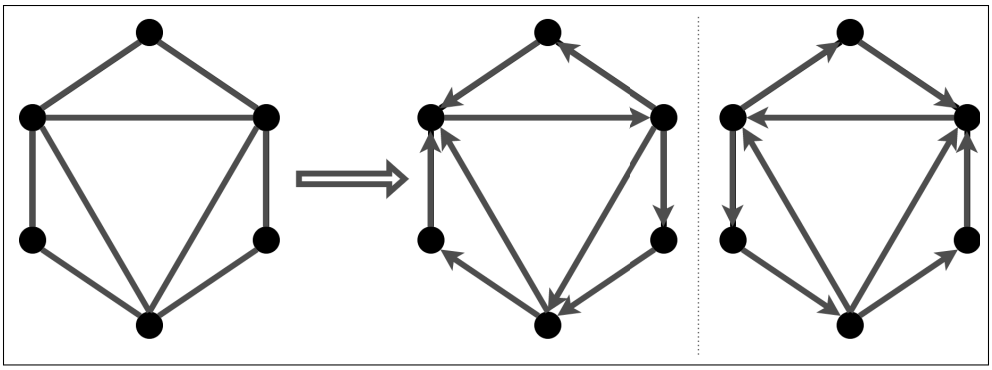

Figure 1.3 Orientation examples of graphs

adapted to obtain a linear-time algorithm, and Robbins formulated this question as a traffic control problem.

Bleichrodt *et al.* (2022) investigates The Edge Orientation Problem with Origin-Destination Pairs (EOPODP). There are two vertices in every OD pair, one representing the origin and the other representing the destination. The objective is to transform each undirected edge into a directed arc so that a direct path exists between the origin and the destination for each OD pair, and the shortest directed OD paths have the shortest total distance. They implement a polynomial-time algorithm to examine the feasibility of any particular EOP-ODP instance. The researchers compare the performance of a MIP formulation, a simple heuristic, and an Ant Colony Optimization algorithm, concluding that the Ant Colony Optimization algorithm outperforms the other algorithms.

## 1.5 Travelling salesman problem

A traveling salesman problem (TSP) is a well-known problem in which a salesman wishes to visit every city and return home at minimal cost, and in his route visits each city exactly once, which means the path between two cities can not be traversed twice (Hoffman *et al.*, 2013).

**Input:** A set of nodes $n \geq 3$ cities. To each pair of cities $(i, j)$ we associate a weight (cost, distance, etc.).

**Output:** A closed path (that starts at the same vertex and ends there) or a tour of minimum cost, where the cost of the tour is defined as the sum of the weights of the edges.

The travelling salesman problem (TSP) is a classic problem in computational mathematics and operations research. The TSP is an NP-hard problem, which means that no known polynomial time algorithm can solve it for all cases, but it is a problem with a lot of practical importance in various fields like logistics and transportation.

There are several methods for solving TSP, each with its own advantages and disadvantages. Some common methods include:

- Exact methods: These methods guarantee to find the exact optimal solution, but they can be computationally expensive for large instances of the problem. Examples of exact methods include branch-and-bound, branch-and-cut, and dynamic programming.
- Heuristic methods: These methods aim to find a good solution quickly, rather than the optimal solution. Examples of heuristic methods include greedy algorithms, simulated annealing, and genetic algorithms.
- Approximation algorithms: These methods aim to find a solution that is close to optimal, but they do not guarantee finding the optimal solution. Examples of approximation algorithms include the Christofides algorithm, the 2-opt algorithm, and the Lin-Kernighan algorithm.
- Metaheuristics: These methods are a family of heuristic optimization algorithms, inspired by nature-inspired processes, like genetic algorithm, simulated annealing, ant colony optimization, etc.
- Deep Learning: In recent years, the TSP problem has also been studied in the context of deep learning and how to use neural networks to model TSP problem.

In addition to the traditional TSP, there are also variants of the problem, such as the asymmetric TSP, in which the distance between two cities can be different in each direction, and the multi-objective TSP, in which there are multiple competing objectives, such as minimizing distance and maximizing profits.

In recent years, researchers have not only extended the classic TSP in several important aspects but also extended it in many parts to accommodate various real-life situations, as summarized in Table 1.1.

Table 1.1 Literature review of TSP

| **Reference** | **Extensions of the classic TSP** |
|---|---|
| (Cacchiani *et al.*, 2020) | The TSP with time windows |
| (Baniasadi *et al.*, 2020) | The clustered traveling salesman problem |
| (Elgesem *et al.*, 2018), | TSP with pickups and deliveries |
| (Pandiri *et al.*, 2018) | K-traveling salesman problem |
| (Pedro *et al.*, 2013) | The prize collecting TSP |
| (de Moraes *et al.*, 2019) | The moving target TSP |
| (Kaspi *et al.*, 2019) | TSP to maximize the profit per unit time |
| (Wang *et al.*, 2019) | The close-enough TSP |
| (Wang *et al.*, 2020) | The energy minimization TSP |
| (Mosayebi *et al.*, 2021) | TSP with job-times |
| (Gencel *et al.*, 2019) | TSP with hotel selection |

Due to the NP-hard property of most of the extensions of TSP, researchers have made great efforts to search and design better solution methods for these variants of TSP. Some exact algorithms have been tried, e.g. the branch-and-cut algorithm (Bleichrodt *et al.*, 2022). Among the heuristic algorithms, the ant colony optimization algorithm (Ottoni *et al.*, 2022) and (Ebadinezhad, 2020) attracted the attention of more researchers than other heuristics. Many researchers also adopted the genetic algorithms (Dong *et al.*, 2019) in their studies. The combination of the genetic algorithm and ant colony algorithm was considered by (Jiang *et al.*, 2020). (Kucukoglu *et al.*, 2019) designed a hybrid simulated annealing and tabu search method to solve the TSP with time windows.

Except for the above heuristics, many others have been proposed in the field of TSP research. They include the artificial bee colony algorithm (Pandiri *et al.*, 2018), harmony search algorithm (Boryczka *et al.*, 2019), discrete differential evolution (Ali *et al.*, 2020), discrete symbiotic organism search (Wang *et al.*, 2019), multi-agent reinforcement learning (Hu, Yao & Lee, 2020), discrete crow-inspired algorithms (Al-Gaphari *et al.*, 2021), discrete bat algorithm (Saji & Barkatou, 2021), etc.

The classic TSP and its variants have many applications in real life. In recent years, researchers have applied the theory of TSP to some new areas. For example, (Defryn *et al.*, 2018) applied TSP theory to dealing with a horizontal logistics cooperation in which multiple companies jointly solve their logistics optimization problem. (Bock & Klamroth, 2019) addressed the classical conflict between cost minimization (represented by the TSP) and customer waiting time minimization (represented by the Traveling Repairman Problem). Except the above development, one new trend in the applications of TSP showed up in recent years. The trend is to consider the use of drones in collaboration with a delivery truck. The related problems are usually formulated into some variants of the classic TSP (Boccia *et al.*, 2021); (Pina-Pardo *et al.*, 2021). Except for the above application of drones in the last mile delivery, drones were also used to surveil wildlife (Chowdhury *et al.*, 2019).

In summary, the traveling salesman problem is a well-known and widely studied problem in operations research and computational mathematics, with many important applications in various fields like logistics and transportation. There are many different methods for solving TSP, each with its own strengths and weaknesses, including exact methods, heuristic methods, approximation algorithms and metaheuristics.

### 1.5.1 Christofides Algorithm

The Christofides algorithm is an approximation algorithm for solving the travelling salesman problem (TSP). The algorithm was proposed by Nicholas Christofides (Chekuri & Quanrud, 2018), and it has a worst-case performance guarantee of 3/2 times the optimal solution. This means that for any instance of the TSP, the Christofides algorithm will return a solution that is at most 1.5 times longer than the optimal solution.

The Christofides algorithm consists of two main steps:

- Finding a minimum spanning tree (MST) of the given graph: An MST is a tree that spans all the vertices of the graph, and has the minimum possible total edge weight. This step can be done efficiently using algorithms like Kruskal's algorithm or Prim's algorithm.

- Finding an Eulerian tour of the MST: An Eulerian tour is a path that visits every edge of the graph exactly once. Once we find an Eulerian tour, we can obtain a Hamiltonian cycle (a cycle that visits every vertex exactly once) by skipping some of the edges.
- Finding the minimum weight perfect matching between the odd-degree nodes in the MST: Perfect matching is a matching that matches all nodes in the graph. This step can be done efficiently using algorithms such as the Blossom algorithm.
- Combining the Eulerian tour and the minimum weight perfect matching to form a Hamiltonian cycle.

The algorithm is based on a combination of graph theory and linear programming and has a strong theoretical foundation. The Christofides algorithm is known to produce near-optimal solutions for TSP instances with symmetric distances, and it is an efficient algorithm that can be implemented using standard graph algorithms.

### 1.5.2 Asadpour Algorithm

The Asadpour algorithm is an approximation algorithm for solving the traveling salesman problem (TSP) that was proposed by (Asadpour *et al.*, 2017). The algorithm is based on the idea of solving the TSP by formulating it as a linear programming (LP) problem and then using a technique called randomized rounding to obtain a solution.

The algorithm starts by formulating the TSP as an LP problem by introducing a set of variables that represent whether or not a particular edge is included in the solution. The LP is then relaxed to a fractional solution, which is a solution where the variables can take on fractional values instead of just 0 or 1.

The next step is to use randomized rounding to obtain an integral solution (i.e., a solution where the variables take on only 0 or 1 value) from the fractional solution. The randomized rounding procedure works by randomly rounding the fractional values of the variables to either 0 or 1, with a probability determined by the value of the variable.

The Asadpour algorithm has a performance guarantee of 2-approximation, which means that it will return a solution that is at most twice as long as the optimal solution. The algorithm is based on the linear programming relaxation of the TSP problem and the randomized rounding technique, which is a well-known technique in the field of approximation algorithms.

The algorithm has been shown to work well in practice, and it has been used to obtain near-optimal solutions for large-scale TSP instances. It is also relatively simple to implement, as it only requires solving an LP problem and applying a randomized rounding procedure.

In summary, the Asadpour algorithm is an approximation algorithm for solving the travelling salesman problem (TSP) that has a performance guarantee of 2-approximation. The algorithm starts by formulating the TSP as a linear programming problem and then uses randomized rounding to obtain an integral solution from the fractional solution obtained from linear programming relaxation. The algorithm is relatively simple to implement and has been shown to work well in practice for large-scale TSP instances.

## 1.6 General Travelling Salesman Problem

In some real problems, sometimes we need to visit each node more than once, so the general traveling salesman problem (GTSP) is defined by (He *et al.*, 2021), in which GTSP is to search a minimum-travel-time closed tour for a salesman traversing certain nodes in a network **at least once**. This problem is an extension of the standard TSP and can be classified as NP-hard problems. GTSP has more real-world applications than standard TSP, such as goods distribution, meter reading, and tourist route planning. GTSP is a NP-hard combinatorial optimization problem, which is an extension of the well-known Traveling Salesman Problem (TSP) that allows a salesman to visit certain nodes of a network more than once. The GTSP has been widely studied in the literature and various solution methods have been proposed, such as:

- Exact Methods: Branch-and-bound, branch-and-cut, branch-and-price, and branch-and-cut-and-price methods are exact methods that have been used to solve GTSP.

- Heuristic methods: Genetic algorithms, simulated annealing, tabu search, variable neighborhood search, and scatter search are heuristic methods that have been used to solve GTSP.
- Metaheuristic methods: Ant colony optimization, particle swarm optimization, and bee algorithm are metaheuristic methods that have been used to solve GTSP.
- Hybrid methods: Hybrid methods that combine exact and heuristic methods have also been proposed in the literature to solve GTSP.

In addition to solving the problem, researchers have also proposed various formulations and variations of the GTSP to model different types of problems, such as the Multi-Depot GTSP, the Prize Collecting GTSP, the Asymmetric GTSP, the Dynamic GTSP, and the Stochastic GTSP.

Overall, the GTSP is a well-studied problem in the literature, with various solution methods and formulations proposed to tackle different variations of the problem. However, solving GTSP problem instances in practice is still challenging and requires more research.

To determine if a TSP has a feasible solution sometimes is very hard. In some cases, there is no feasible solution to a TSP. Different from the standard TSP, GTSP defined on a connected network is always solvable. In other words, we can always find out a feasible solution to a GTSP. The above statement can be proved easily based on the fact that we can always transform a GTSP into an equivalent TSP. The TSP resulting from the above transformation is always solvable because there is a directed connector between any given pair of required cities. In view of the above observation, the feasible application fields of GTSP are more than that of TSP. As an extension of the classic TSP, GTSP can be applied to modeling many real-world situations. For example, to plan the route for a rider working for a delivery company.

In view of the NP-hard property, it is still a challenge to obtain the optimal or near-optimal solution of a medium or large-scale GTSP in a reasonable time.

### 1.6.1 Bridge finding algorithms

Bridge-finding algorithms are a class of algorithms used to identify important edges, called bridges, in a graph (Gould, 2012). A bridge is an edge whose removal increases the number of connected components in the graph. These edges are considered important because they play a significant role in graph connectivity and can be used to identify key structural properties in the graph.

There are several algorithms for finding bridges in a graph, each with its own advantages and disadvantages. Some common algorithms include:

- Depth-first search (DFS): DFS is a classic algorithm for traversing a graph, and it can be used to identify bridges by keeping track of the discovery and finish times of each vertex during the traversal. If an edge connects a vertex with an earlier discovery time to a vertex with a later finish time, it is considered a bridge.
- Tarjan's algorithm: Tarjan's algorithm is an efficient algorithm for identifying bridges in a graph, based on DFS. The algorithm uses a data structure called a stack to keep track of the discovery and finish times of each vertex, and it can identify bridges in linear time.
- The Decomposition algorithm: This algorithm is a computational method used to identify bridges in a graph through decomposition, which is the process of breaking down the graph into smaller parts. The algorithm seeks to locate the edges that, if removed, would increase the number of connected components in the graph(bridges).
- Kosaraju's algorithm: Kosaraju's algorithm is a linear-time algorithm for finding bridges in a directed graph. The algorithm uses two DFS traversals, one on the original graph and one on its transpose, to identify strongly connected components and bridges.
- Bridges in linear time: There is another linear-time algorithm called "Biconnected Components Algorithm" that can identify the bi-connected components and bridges at the same time.

  Bridges in linear time with using DSU: There is another linear-time algorithm called "Bridge Finding using DSU" that can identify the bridges using DSU (Disjoint Set Union) data structure.

In addition to identifying bridges, many of these algorithms also compute other important graph properties such as connected components, bi-connected components, and articulation points.

In summary, bridge-finding algorithms are a class of algorithms used to identify important edges, called bridges, in a graph. There are several algorithms for finding bridges in a graph, each with its own advantages and disadvantages. Some common algorithms include depth-first search, Tarjan's algorithm, Kosaraju's algorithm, "Bi-connected Components Algorithm" and "Bridge Finding using DSU". These algorithms can also compute other important graph properties such as connected components, bi-connected components, and articulation points.

### 1.6.2 Completing the graph

A common assumption in the literature when solving the Traveling Salesman Problem (TSP) is that the graph is complete (Schermer, Moeini & Wendt, 2020). However, constructing a complete graph for a large number of cities can be a time-consuming process.

The problem of completing a graph, also known as the graph completion problem, is the task of adding edges to an incomplete graph such that certain properties are satisfied. The problem can be formalized as follows: Given a graph $G = (V, E)$, find the most likely set of missing edges consistent with the observed edges.

One of the main approaches to solving the graph completion problem is through matrix completion, which involves using techniques from linear algebra to fill in missing entries in a matrix that represents the graph. One popular algorithm in this category is the Nuclear Norm Minimization (NNM) algorithm, which is based on the idea of minimizing the nuclear norm (i.e., the sum of the singular values) of the matrix. This algorithm has been shown to be effective for completing low-rank matrices and has been applied to various types of graphs, including social networks, collaborative filtering, and protein-protein interaction networks (Huang & Wolkowicz, 2018).

Another popular approach to solving the graph completion problem is through the use of probabilistic models, such as the probabilistic matrix factorization (PMF) algorithm. PMF models the graph as a product of two low-rank matrices and uses probabilistic techniques to infer the missing entries. This algorithm has been applied to various types of graphs, including recommendation systems, image processing, and natural language processing (Zhou *et al.*, 2012).

A third approach is the use of graph neural networks, which is a type of neural network designed to operate on graph-structured data. Graph neural networks have been used to model graph completion problem and have been shown to be effective in various applications such as social network analysis (Wang *et al.*, 2021).

The graph completion problem has also been studied in the context of computational complexity and hardness. It has been shown that the problem is NP-hard in the general case, but polynomial-time algorithms exist for certain classes of graphs.

In recent years, the problem of graph completion has been studied in the context of large-scale and dynamic graphs, as well as in the context of graph embedding, which is the problem of representing graphs in low-dimensional vector spaces. The graph completion problem is also related to the problem of link prediction, which is the task of predicting missing edges in a graph based on the existing edges.

Before completing the graph, we need to be sure our graph is bridgeless. In a graph, a bridge is an edge whose removal causes the number of components connected to increase. Alternatively, a bridge can be defined as an edge that does not belong to any cycle. Bridges are also known as cut edges, isthmuses, or cut arcs. A graph is considered bridgeless if it does not have any bridges. This is equivalent to stating that all connected components within the graph are 2-edge-connected, or according to Robbin's theorem, every connected component has a strong orientation.

#### 1.6.2.1 Floyd-Warshall algorithm

The Floyd-Warshall algorithm is an algorithm for finding the shortest paths between all pairs of vertices in a weighted graph. The algorithm is a classic algorithm for solving the all-pairs shortest path problem (Cormen, 2022).

The algorithm dynamically updates the shortest path between each pair of vertices by considering all possible intermediate vertices. The algorithm maintains a distance matrix, where the entry $d[i][j]$ represents the shortest distance between vertex $i$ and vertex $j$. The algorithm then repeatedly updates this matrix by considering each vertex $k$ as an intermediate vertex and updating the distance between each pair of vertices $i$ and $j$ as $d[i][j] = min(d[i][j], d[i][k] + d[k][j])$. The algorithm repeats this process for each vertex k, and the final distance matrix $d[i][j]$ will contain the shortest path between each pair of vertices $i$ and $j$.

The time complexity of the Floyd-Warshall algorithm is $O(V3)$, where $V$ is the number of vertices in the graph. This makes the algorithm quite efficient for dense graphs, but it can be quite slow for sparse graphs with a large number of vertices. However, the algorithm can detect negative cycles, and if the final matrix $D[i][i]$ is negative, the graph contains a negative cycle.

The Floyd-Warshall algorithm can also be used to solve other problems, such as transitive closure and reachability, and it can be modified to handle directed graphs as well as undirected graphs.

In summary, The Floyd-Warshall algorithm is an algorithm for finding the shortest paths between all pairs of vertices in a weighted graph. The algorithm works by maintaining a distance matrix and repeatedly updating it by considering each vertex as an intermediate vertex. The time complexity of the Floyd-Warshall algorithm is $O(V3)$, where $V$ is the number of vertices in the graph. The algorithm is quite efficient for dense graphs but can be slow for sparse graphs. The algorithm can also be used to detect negative cycles and other problems, such as transitive closure and reachability, and it can be modified to handle directed graphs.

#### 1.6.2.2 Seidel's algorithm

According to (Seidel, 1995), Seidel's algorithm solves the all-pairs-shortest-path problem for connected, undirected graphs.

The mathematical foundations of Seidel's Algorithm are based on matrix theory and linear algebra. The method works by iteratively updating the approximations of the solution, using the values from the previous iteration, until the solution converges to a sufficient degree of accuracy. The convergence of Seidel's Algorithm can be accelerated by using a suitable relaxation factor, which adjusts the rate at which the approximations are updated.

Seidel's Algorithm has been applied in a wide range of fields, including numerical analysis, engineering, physics, and computer science. In numerical analysis, the method is used to solve systems of non-linear equations and to approximate solutions to partial differential equations. In engineering, Seidel's Algorithm is used to design and analyze control systems, power systems, and other complex systems. In physics, the method is used to simulate the behavior of physical systems, such as heat transfer and fluid flow. In computer science, Seidel's Algorithm is used in graph algorithms, such as finding the shortest path in a graph, and in machine learning, where it is used to train neural networks.

In conclusion, Seidel's Algorithm is a widely used iterative method for solving systems of linear equations and has a rich history and a wide range of applications in various fields. Despite its efficiency, the method can be sensitive to the choice of initial approximations and can converge slowly for certain systems of equations, making it necessary to choose appropriate stopping criteria and to monitor the convergence of the solution.

Despite of the above-mentioned algorithms, (Emami Taba, 2010), six different methods for generating non-complete graphs (XNN, TSXP, MST, HVC, LHVC, and RSPT) were introduced. These developments in generating non-complete graphs for TSP have led to significant advancements in solution methods for generating non-complete graphs for other

types of the Vehicle Routing Problem. Among the six algorithms, XNN was found to be the most efficient in terms of both the quality of the tour and the time required to generate it.

In our literature review, we found that existing works on hospital routing and path-finding mainly focus on optimizing patient flow in the emergency department. While these are important considerations, they do not adequately address the unique challenges posed by the COVID-19 pandemic, such as the need to minimize physical contact and exposure between patients, staff, and visitors. Additionally, many existing works do not consider the dynamic nature of the pandemic, which requires frequent updates to routing and path-finding strategies.

Routing and path-finding in the corridors of hospitals during the COVID-19 pandemic is a critical issue that requires urgent attention. Although there have been some attempts to address the problem, it is clear that more comprehensive and efficient solutions are needed. By building on existing algorithms studied in the literature review, we can potentially improve their performance. To successfully adapt an algorithm, it's crucial to have a deep understanding of the specific problem and the data that will be used. Additionally, it's important to thoroughly analyze the original algorithm, identifying its strengths and weaknesses to identify areas for improvement. In this thesis, we aim to fill this gap by proposing a novel approach that takes into account the specific challenges posed by the COVID-19 pandemic.

# CHAPTER 2

# METHODOLOGY

The input for this problem is a graph represented by $G = (V, E)$, where $V$ is the set of nodes and $E$ is the set of edges. Additionally, the input includes two sets of nodes, in the case of COVID-19 and Non-COVID patients, and a weight matrix $W_e$, which are explained in detail next section.

The output is an oriented graph, which means that the edges have a direction for both the flow of patients. Each edge in the graph is labeled with its type.

The graph represents a network, such as a transportation network or a hospital network, and the flow of patients refers to the movement of patients through the network. The sets $\mathcal{SD}_{\text{CO}}$ and $\mathcal{SD}_{\text{NCO}}$ represent specific nodes in the network that are designated for the patients. The weight matrix $W_e$ contains information about the weight associated with each edge in the network.

The output graph takes into account all of this information and represents the most efficient path for the flow of patients through the network, considering the $\mathcal{SD}_{\text{CO}}$ and $\mathcal{SD}_{\text{NCO}}$ nodes, as well as the edge weights.

The labeled edges provide additional information about the edges in the graph, such as the type of edge it is (e.g., one-way, two-way) and the type of patient that can be transported through the edge (e.g., Red patients, Green patients).

## 2.1 Building a floor graph

In graph theory, a planer graph is a graph with edges intersecting only at their endpoints on the plane (Trudeau, 2013). We consider the case in which a planer graph is derived from the floor map of the hospital and then used for pathway description. To model the hospital map, we first create a physical topology as graph $G(V, E)$, where $V$ denotes the set of all vertices, and $E$ denotes the set of unidirectional edges. Each vertex $n \in V$ represents an intersection, wards, and doors. All pairs of source and destination for COVID-19 and Non-COVID patients denote by $\mathcal{SD}_{\text{CO}}$ and $\mathcal{SD}_{\text{NCO}}$. The requirement for each flow of patients is to find disjoint paths and keep

a safe distance as much as possible to minimize encounters. The set of nodes $V$ is defined as follows.

- Each door is a node, and if two doors face each other in a corridor, we use a single node to represent them.
- Each intersection of corridors is a node.

For any two nodes $m, n \in V$ associated with doors, which are adjacent along the same corridors, are linked by multiple edges corresponding to the number of lanes in the corridor. If the corridor is wide enough, then more than one lane can be defined.

The edge from node $m$ to node $n$ is denoted by $e_{mn} \in E$, and each edge is labeled as follows.

$$L(e_{mn}) = \begin{cases} R & \text{if } e_{mn} \text{ belong to a COVID-19 path;} \\ G & \text{if } e_{mn} \text{ belong to a Non-COVID path.} \end{cases} \tag{2.1}$$

In which $R$ represents red color for COVID-19 patients and $G$ represents green color for Non-COVID patients. We study the problem of orienting edges in $E$ to guarantee the reachability for all $\mathcal{SD}_{\text{CO}}$ and $\mathcal{SD}_{\text{NCO}}$. The goal is to obtain a directed graph that meets certain connectivity requirements. So our objective is to find an orientation of planar graph $G = (V, E) \rightarrow G' = (V, E, A, L)$, in which $V$ is vertex, $E$ is the edges, $A$ is the arc, and $L$ is the label (green or red), to minimize the number of two-way and shared edges/links such that all pairs are connected with the COVID-19/Non-COVID patients requirements.

## 2.2 Offline scenario

In the first scenario, we address the graph orientation and labeling problem, where some of the edges will be directed and colored i.e., labeled as either COVID-19 or Non-COVID or bi-colored edges/links, in such a way as to fulfill the path requirements of sets $\mathcal{SD}_{\text{CO}}$ and $\mathcal{SD}_{\text{NCO}}$. In other words, it is a generalization of the one-way street problem, using mixed graphs (see, e.g., (Arkin *et al.*, 2002)), i.e., graphs with both edges and links, in order to guarantee paths for each node

pair. For a given source/destination, patients follow a pre-determined route. In this way, first, we

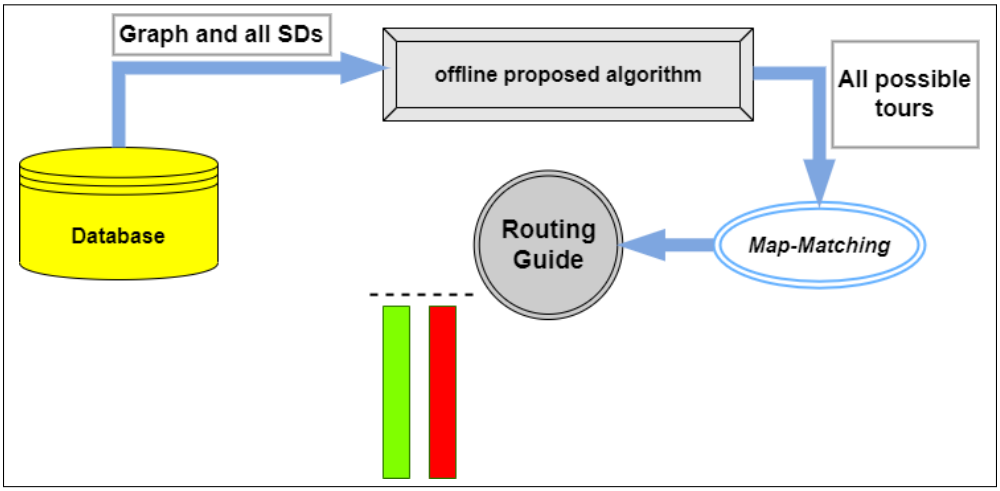


Figure 2.1 Proposed offline scenario

divide corridors into one or several lanes, label them, and then use the color/direction to travel in the hospital. Fig. 2.1 illustrates the proposed offline scenario.

### 2.2.1 Graph orientation and labeling problem

The idea is to assign circulation direction in each corridor lane as either a one-way lane or alternating lanes, with respect to COVID-19 and Non-COVID patient flows. In our problem, as we are looking for two general paths for red and green patients, it is not always possible to find the 2-disjoint path for these two flows of patients, so we will define the strategy based on the different types of edges to guarantee the reachability in the hospital. More precisely, we have four types of lanes (see Figure 2.2 for an illustration):

- **Type 1.** one-way lane dedicated to a unique flow of patients, either COVID or Non-COVID
- **Type 2.** mixed one-way lane, with alternate usage for COVID-19 and Non-COVID patients
- **Type 3.** two-way lanes dedicated to a unique flow of patients, either COVID-19 or Non-COVID, with alternate usage of each lane direction
- **Type 4.** mixed two-way lanes, with alternate usage for both ways and an alternate usage for at least one way between COVID-19 and Non-COVID patients.

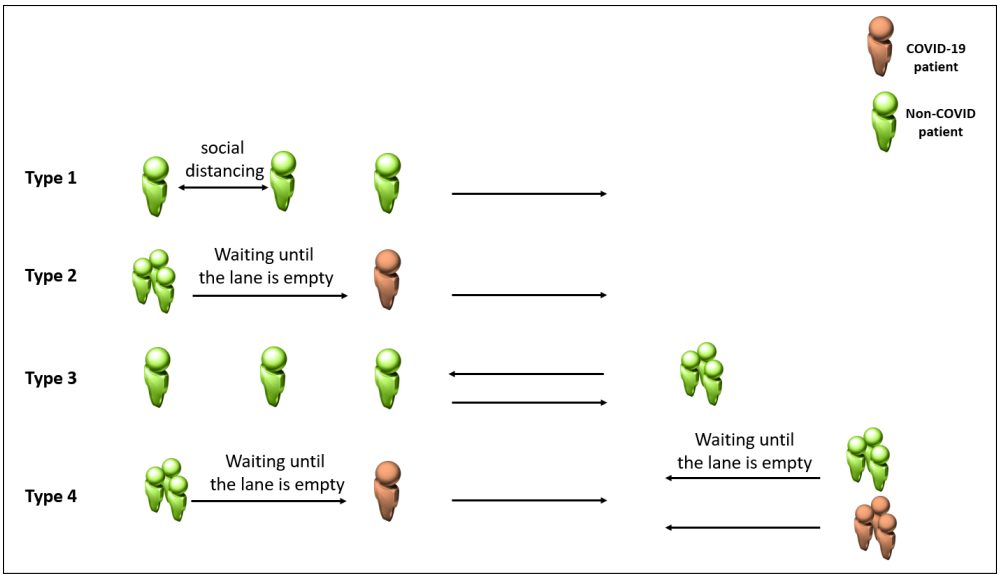


Figure 2.2 Lane types

The objective is to assign directions to as many edges as possible so as to minimize the number of shared lanes, minimizing first the number of shared lanes between COVID-19 and Non-COVID patients.

### 2.2.2 Graph orientation and labeling algorithm

In the first scenario, the idea is to define routes for all trips of interest from one place to another in the hospital for both COVID-19 and Non-COVID patients. In order to do so, we start from the graph associated with the floor map of the hospital and define a graph orientation in order to minimize the number of shared lanes while ensuring travels are as short as possible. Fig. 2.3 illustrates the associated graph of Fig. 0.1.

**Pre-processing steps.** The idea is to take care of corridors for which we cannot avoid sharing, i.e., corridors corresponding to the dead-ends (see nodes A and B in Figure 2.3). Indeed, if one of the nodes of the dead-end corridor belongs to $\mathcal{SD}_{\text{co}}$, then the edges of the dead-end edges become links of type 4; otherwise, they are type 3, with assuming that at least one door is an exit/entrance door. Otherwise, some edges could be of type 1 or 2 links.

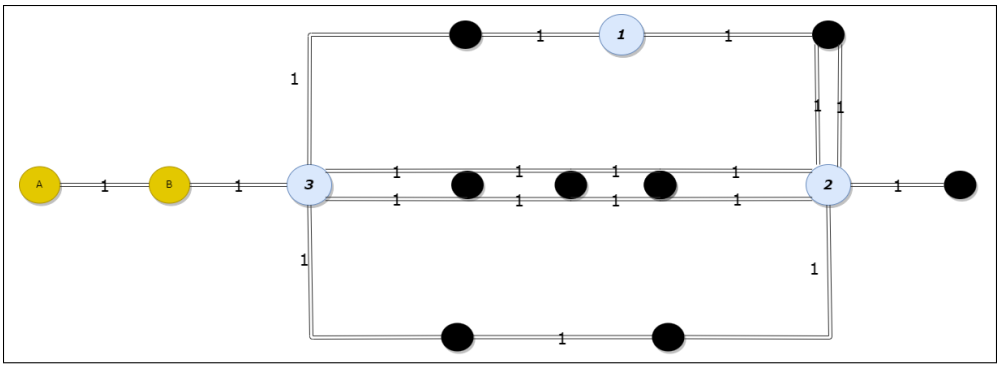


Figure 2.3 Graph associated with example floor

The second pre-processing has to do with some nodes, i.e., nodes associated with corridor intersections. We can eliminate the nodes of degree 2 (which is not source or destination), such as node 1 in Figure 0.1, and reconnect the two neighbor nodes directly. These nodes play no role in terms of patient flow bifurcation.

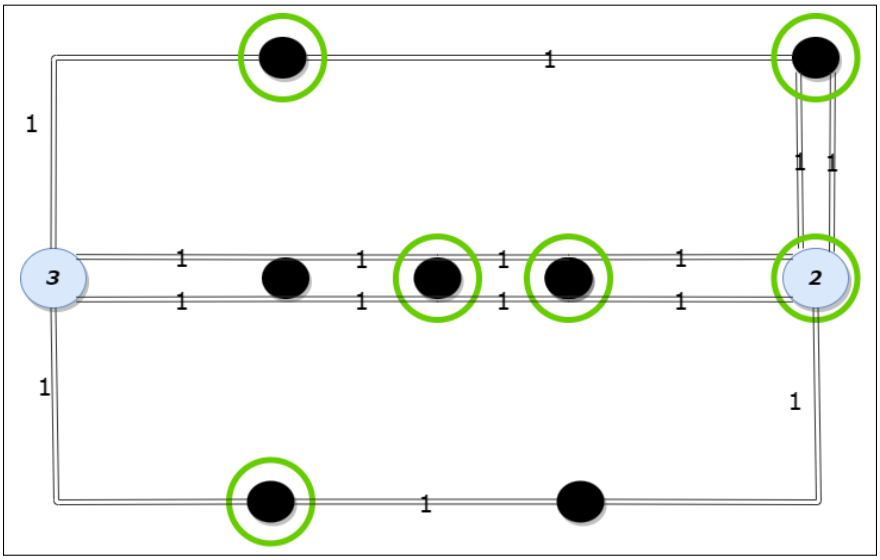


Figure 2.4 Removing the dead-ends and determining $\mathcal{SD}_{\text{NCO}}$

**Identification of bridges among the edges.** As our problem is a generalization of the one-way street problem: this problem has a solution if and only if the graph has no bridge (Robbins, 1939), i.e., no edge such that their removal breaks the graph into two components. While this

condition is no more an if-and-only-if component in our context, it remains interesting to identify the bridges insofar as they remain obligatory passages for the destinations which are not on the same side of the bridge as their sources. Bridges can be identified in an undirected graph using chain decompositions bridge-finding algorithm (Schmidt, 2013). Once bridges are identified, a search of paths can be handled either in a mixed graph as in (Arkin *et al.*, 2002) where bridges remain as edges (i.e., of links of type 3 or 4) or in each connected component of the graph followed by a phase of connecting the identified sub-paths in each connected component so as to define the paths for the requested node pairs of $\mathcal{SD}_{\text{CO}}$ and $\mathcal{SD}_{\text{NCO}}$. Fig. 2.4 illustrates the removal of bridges (in this example, we remove the dead-end as a bridge).

**General Hamiltonian tour in each connected component.** While we do not need to connect all node pairs, the identification of a Hamiltonian tour going through all the black nodes in every connected component generates paths for every requested node pair. However, as the sub-graph associated with each connected component is incomplete, we may need to go through each node more than once. However, a reduction to the classical TSP can be easily made: calculate the shortest path distance between every two non-direct nodes and set that to be the distance between the two nodes. After solving the TSP, we also remembered the shortest paths, so then we could transform it back into a solution to the original problem using the computed shortest paths using the all-pairs shortest paths algorithm. Once we have obtained a Hamiltonian tour, we can identify an orientation of edges upon deciding on an orientation of the Hamiltonian tour. For a simple graph, after pre-processing, dead ends are eliminated, and after building a complete subgraph from the source and destination, we find a Hamiltonian cycle, and then we find the corresponding path in the pre-processed graph. Fig. 2.5 and Fig. 2.6 illustrates the steps involved in finding the Hamiltonian tour for $\mathcal{SD}_{\text{NCO}}$ and determining the $\mathcal{SD}_{\text{CO}}$ for COVID patients.

In order to solve the TSP problem, we can either use an exact algorithm (see, e.g., (Applegate *et al.*, 2007)) or a heuristic. In our case, we use the approximate algorithm of Christofides (Christofides, 1976; Toth *et al.*, 2002).

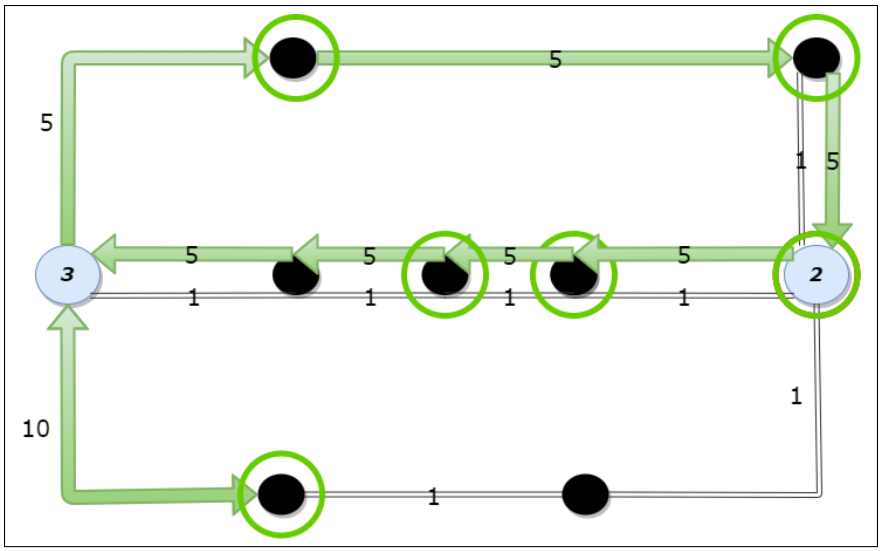


Figure 2.5 First tour for $\mathcal{SD}_{\text{NCO}}$

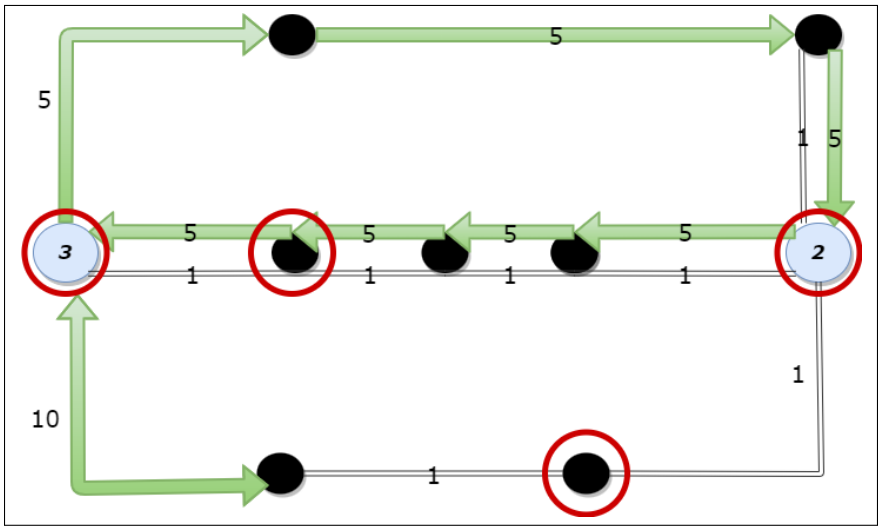


Figure 2.6 Determining the $\mathcal{SD}_{\text{CO}}$

With the Christofides algorithm, we find the shortest tour, and each selected edge increases the weight. In fact, we set a specific weight for each type of edge, which after solving the problem for flow of patients $\mathcal{SD}_{\text{NCO}}$, the edges that are selected change to one of the types (the type 1 and 3). And then, we solve the problem for the other flow of patients $\mathcal{SD}_{\text{CO}}$ (with the updated graph). We specify a weight for each type. In the beginning, all edges have a weight equal to

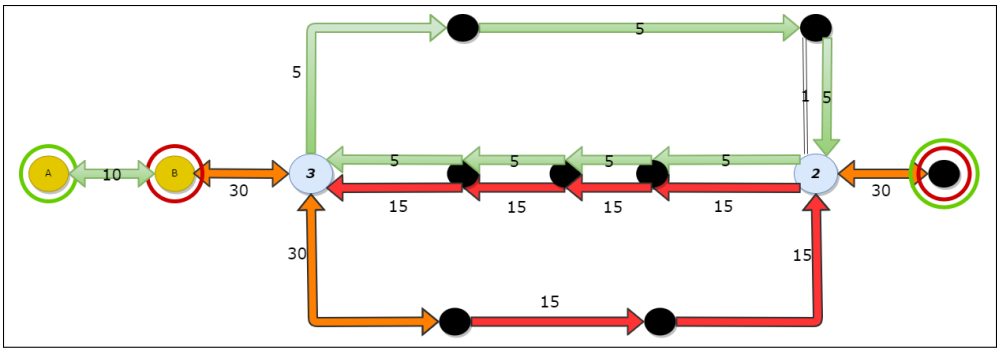


Figure 2.7 Final tours for $\mathcal{SD}_{\text{NCO}}$ and $\mathcal{SD}_{\text{CO}}$

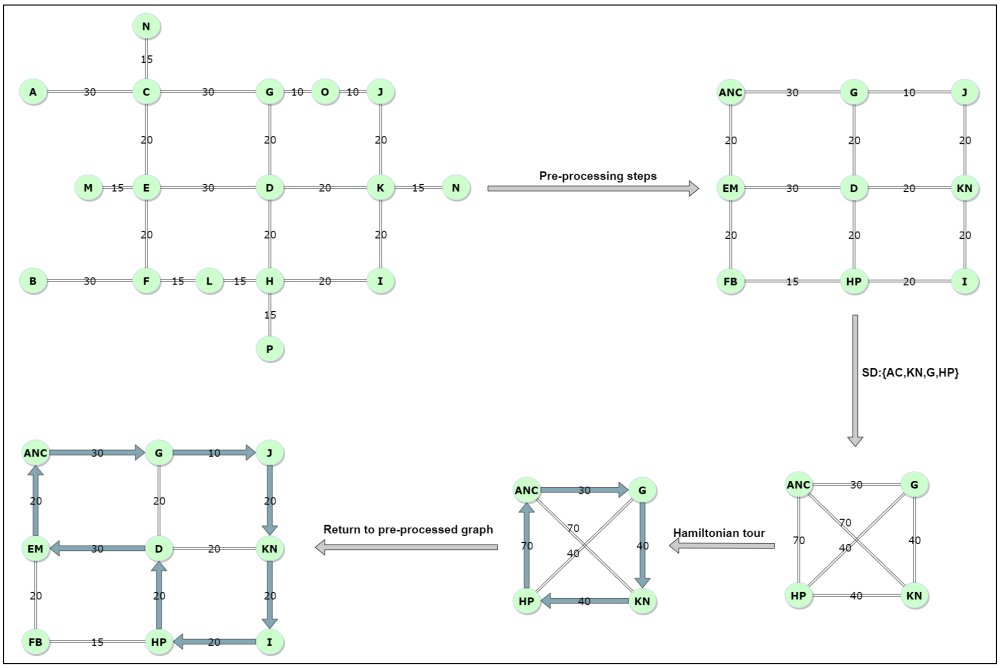


Figure 2.8 The procedure in the offline scenario

1, except for virtual edges (added edges to complete the graph), which weight is equal to the sum of the actual equivalent edges. After solving the problem for each flow of patients, we change the weight of the edges that are selected. Whatever the weight of each type depends on our graph. We assign different weights ($W_e$) to the edges for each type based on priority. For example, for types 3 and 4, the weight should be higher, and for type 1 (which is our priority),

Algorithm 2.1 Orientation algorithm for offline Scenario

**Input:** $G = (V, E)$, $\mathcal{SD}_{\text{CO}}$, $\mathcal{SD}_{\text{NCO}}$, $W_e$
**Output:** oriented graph for both flow of patients, Labeled edge: $type1, ..., 4$
**Run** $Decomposition\_Algorithm$
**if** $n, m \in \mathcal{SD}_{NCO}$ & $n, m \in \mathcal{SD}_{CO}$ **then**
  $e_{nm} \rightarrow type4$
**end if**
**else**
  $e_{nm} \rightarrow type3$
**end if**
**Remove** all bridges
**Run** $allShortestPath\_Algorithm$
**Run** $Christofides\_Algorithm$ for all $\mathcal{SD}_{\text{NCO}}$
**for** $e_{nm} \in HamiltonianTour\mathcal{SD}_{NCO}$ **do**
  $e_{nm} \rightarrow type1, 3$
  $W_e \leftarrow W_e + \alpha_{NCO}$;
**end for**
**Run** $allShortestPath\_Algorithm$
**Run** $Christofides\_Algorithm$ for all $\mathcal{SD}_{\text{CO}}$
**for** $e_{nm} \in HamiltonianTour\mathcal{SD}_{CO}$ **do**
  $e_{nm} \rightarrow type1, 2, 3, 4$
  $W_e \leftarrow W_e + \alpha_{CO}$;
**end for**
**return** $types$

we choose a weight equal to or around the weight of the virtual edge (in other words, the cost of the shortest path between two vertices).

**Reducing link and edge-sharing.** After the prepossessing step, we use the Christofides algorithm to find the first tour for $\mathcal{SD}_{\text{NCO}}$ and update the weight of edges for the next step. To find the tour for $\mathcal{SD}_{\text{CO}}$, we run the christofides algorithm for the second time on the updated graph and called the tour as **Ambulance Mode**, in which the tour consists of the corridors that have minimum shared corridors with the Non-COVID paths. Also, to increase the accessibility for $\mathcal{SD}_{\text{NCO}}$, we run the christofides algorithm at the end again. Fig. 2.7 illustrates the Final tours for $\mathcal{SD}_{\text{NCO}}$ and $\mathcal{SD}_{\text{CO}}$, and Fig. 2.8 illustrates the general steps involved in scenario one. Our proposed algorithm for offline scenario is presented in Algorithm 2.1.

## 2.3 Online scenario

At the outset, we compute a set of potential uncolored paths for each pair of sources/destinations. At any time, when a patient moves, paths are dynamically re-evaluated (check for the first path that is available when reaching a corridor intersection) based on their availability at each corridor intersection. For a green patient, if a green path is selected, only social distancing is required. Otherwise, the green patient has to wait until the path is free (and recolored green). For a red patient, the path has to be cleared first, labeled red, before he can go. Color is removed when a link becomes empty. In this scenario, we have a smartphone, and it gives the direction (at any time, it looks for the shortest available path, refreshing its calculation at every corridor intersection). In Scenario 2, the idea is to dynamically decide which way to go at

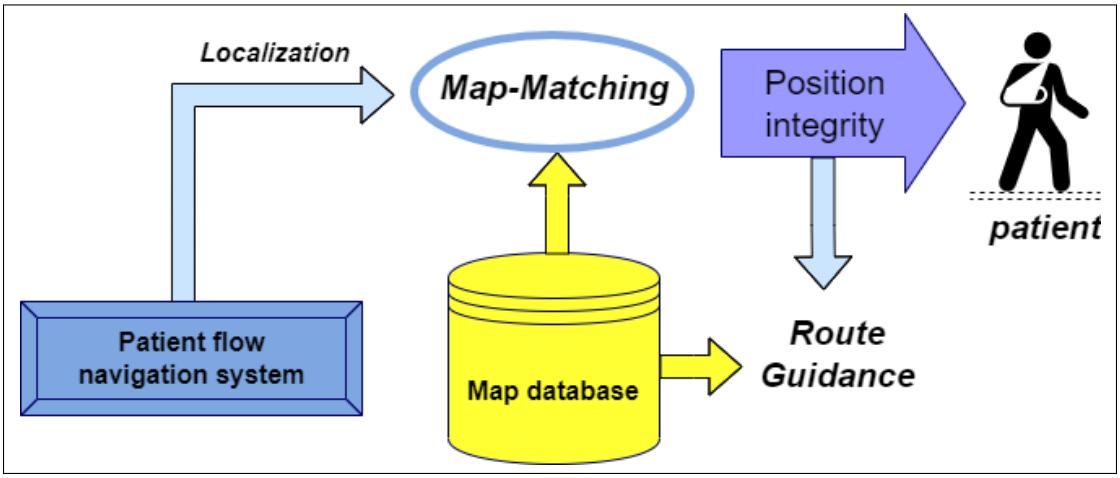


Figure 2.9 Patient flow navigation system in the online scenario

each intersection based on the real-time information of the corridor occupancy (Fig. 2.9 shows the patient flow navigation system). Indeed, at each corridor intersection, we check with the monitoring system which path towards the patient's destination has the least meets with other patients and the minimum occupancy. We consider different options to define both the least number of patient encounters and the least occupancy.

**Non-COVID patients path:** The least number of patient encounters refers to the minimum estimate of the number of Non-COVID patients encountered while traveling to the destination

because we allow having several Non-COVID patients simultaneously in the corridors (based on the social distance). As travel paths are dynamically decided, estimation of meets is not easy. Therefore an alternative is to only estimate the meets in the next 1 or 2 corridors along the path towards the destination.

**COVID-19 patients path:** We use a similar strategy to select the path towards the destination, with the difference that we wait at the entrance of each corridor until the corridor is free. COVID-19 patients will have dedicated corridors, and Non-COVID patients will wait in the corner of the entrance corridor.

In scenario 1, paths are precomputed and are colored, but in Scenario 2 paths are precomputed and are uncolored at the outset. In the second scenario, we assume that everybody has a cell phone, and we have their location. We dynamically allocate the safest and shortest routes to patients who have a smartphone. In this case, simply we can send a notification to users in order to warn them that they must change their location. The corridors are colored at the time of their selection if no patient is already on the path (with social distancing). Color is removed when a link becomes empty. In the prepossessing step, for every corridor intersection, ward, exit, and the entrance of the hospital, we compute the k shortest paths. When a patient reaches a corridor intersection: identify the "shortest path" (including the waiting time/queuing) to their destination, so that determine which way the patient go next, and then stop at the first encountered corridor intersection.

**Paths intersection constraints:** We always consider the path for red patients as one way, and we can have just one red patient at a time. The green patients must not intersect with red patients and satisfy the required social distancing (2 meters) with other patients. Although green patients have less priority than red patients, they have more freedom to allocate different routes.

**Definition of the priorities:** We define the priority of different patients group in the hospital as red is always our priority over the green, and considering social distancing for green patients. The green can consist of patients, medical staff, and security guards, accompanied by a patient.

In the green paths, we could have many people while keeping a safe distance. If the shortest path does not have the capacity, it checks the next corridor. The patient can wait at the side of the intersection (waiting area) until one of the corridors becomes available (priority with the first corridor). Our proposed algorithm for the online scenario is presented in Algorithm 2.2.

Algorithm 2.2 routine algorithm for online Scenario

**Input:** $G = (V, E)$, $\mathcal{SD}_{\text{CO}}$, $\mathcal{SD}_{\text{NCO}}$, $W_e$
**Output:** path for each patient with $\mathcal{SD}_{\text{CO}}$, $\mathcal{SD}_{\text{NCO}}$

**if** *patient* $\in \mathcal{SD}_{NCO}$ **then**
  **for** $n, m \in \mathcal{SD}_{NCO}$ **do**
    **Run** $ShortestPath\_Algorithm$
    $W_e \leftarrow W_e + \alpha_{NCO}$;
  **end for**
**end if**
**else**
  **for** $n, m \in \mathcal{SD}_{CO}$ **do**
    **Run** $ShortestPath\_Algorithm$
    $W_e \leftarrow W_e + \alpha_{CO}$;
  **end for**
**end if**
**return**

# CHAPTER 3

# SIMULATION RESULTS

In this section, we first discuss the generation of the required data and hospital map; then we compare two proposed scenarios. We read the CSV file of the hospital map floor, create our graph, and then implement our proposed algorithms using NetworkX library (Hagberg *et al.*, 2008).

## 3.1 Generation of data sets

**Patient flows in the Emergency Department:** We consider three sequences of people in the Emergency Department as shown in Table 3.1.

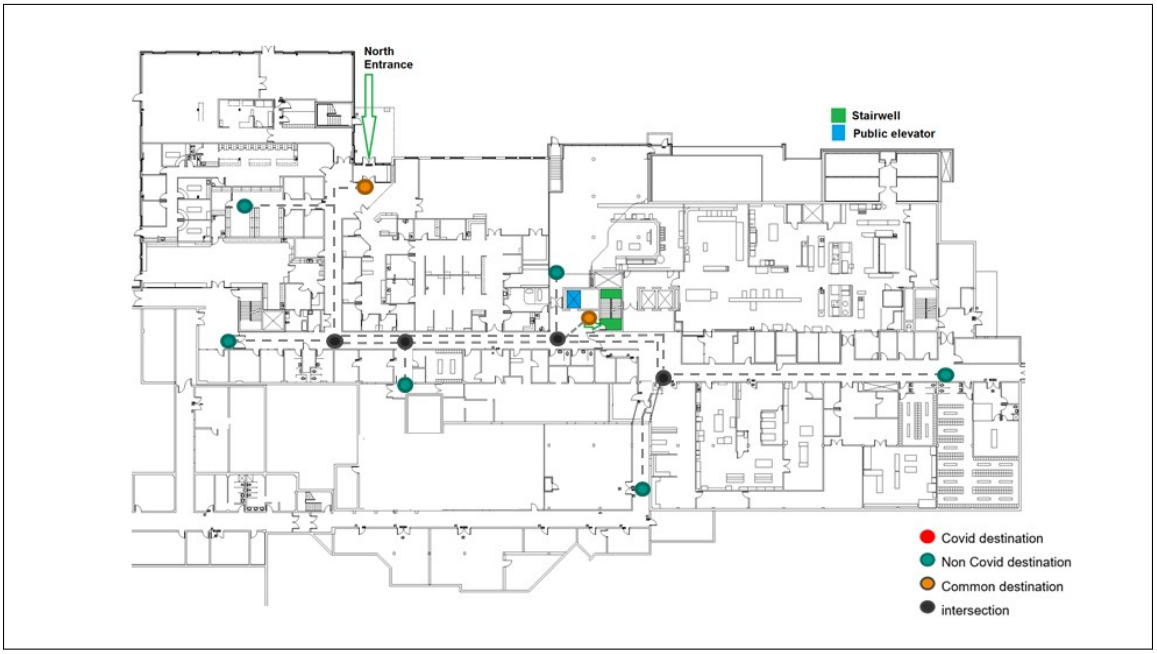


Figure 3.1 Hospital inpatient map of the first floor

Nurse assessment and doctor examination are two key steps in an emergency, possibly in addition to examinations required by the doctor, e.g., CT-scan or other tests. In this case, the patient must circulate in the hospital outside the emergency room: he takes different corridors of the hospital and crosses paths with other patients.

Table 3.1 ED Patient Sequences at the Emergency Department

| Sequence Number | Visiting flow |
|---|---|
| Seq1 (10%) | Nurse triage, clerk registration, patient discharge. |
| Seq2 (60%) | Nurse triage, clerk registration, nurse assessment, physician examination, patient discharge. |
| Seq3 (30%) | Nurse triage, clerk registration, nurse assessment, physician examination, CT scan, patient discharge. |

**Patient flows in the wards of hospital:** In this work, we consider different wards, e.g., Ambulatory Care, dialysis, Chemotherapy (COPS), Diagnostic Imaging, the Laboratory, etc. Similar to Emergency Department, we consider a different sequence of patients for hospital wards, but we suppose each destination is a new source for the next step.

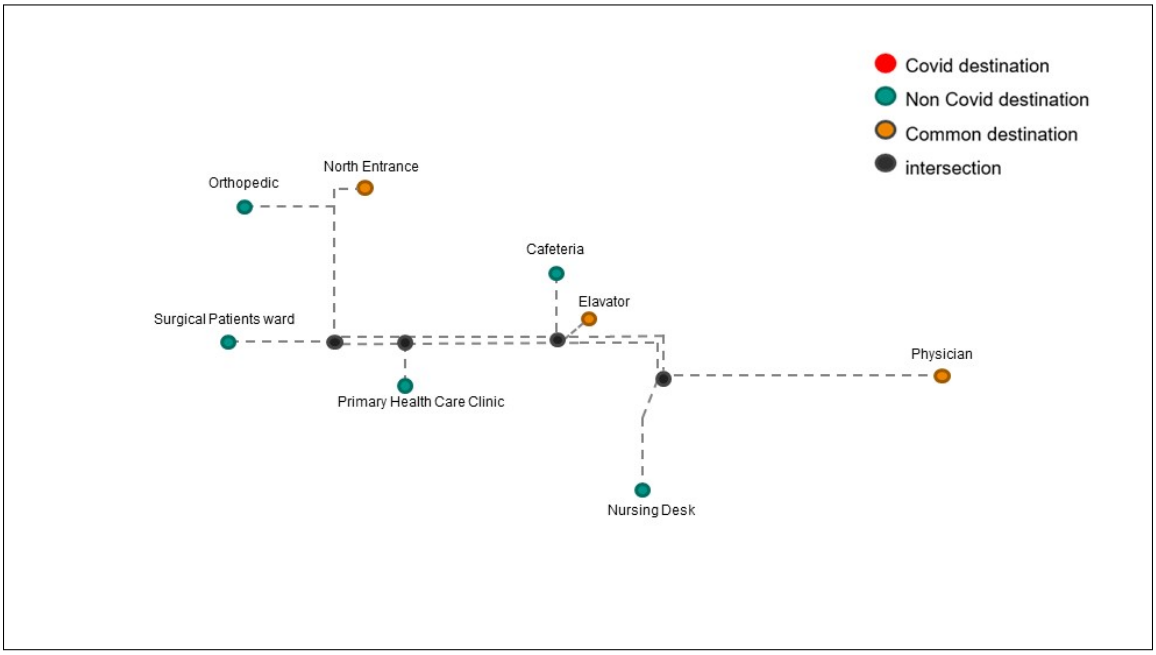


Figure 3.2 The corresponding graph of the first floor

**Patient Arrival Patterns:** We use Poisson Distribution for the Emergency Department. For other wards, we consider scheduled patient arrival time, in which they accept patients based on the scheduled appointments.

## 3.2 Hospital map

It is not possible to navigate directly from 2D floor plan drawings or a database. For path computation, indoor spatial details must be processed to create an indoor graph. An indoor graph describes the spatial connections between indoor spaces concisely and contains enough geometrical and topological data to describe the surrounding environment. Figure 3.1, 3.2, 3.3, 3.4 are examples of two indoor floors of Saskatchewan Health Authority (Authority, 2022) including wards, accessible and inaccessible areas, entrance and exit doors, walls, and paths. As a

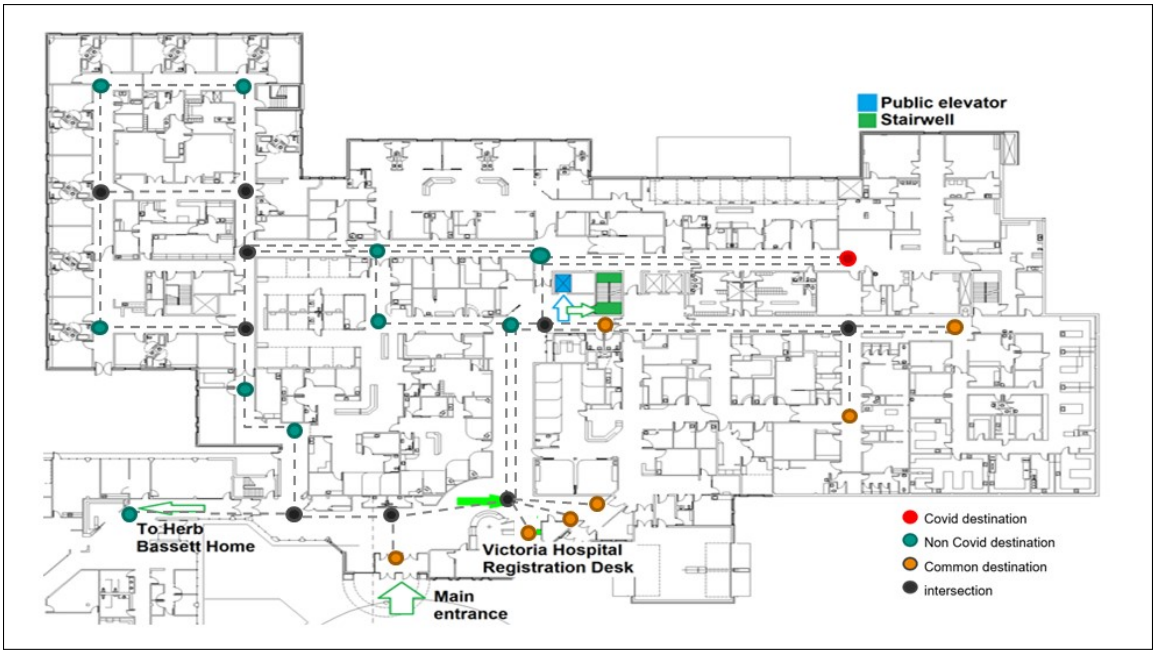


Figure 3.3 Hospital inpatient map of the second floor

simplified view, a floor map is a set of rooms with a set of doors, and each door is either an entrance only, an exit only, or a combined entrance and exit. In the COVID context, the idea is to use as many doors as possible, including some emergency exits, in order to separate the entrance and exit for COVID-19 and Non-COVID patients, reducing the meets between patients commuting to the wards. We set the speed of every patient 0.5 per second, and the simulation lasted five hours. The simulation settings are summarized in Table 3.2. We ran all experiments for both scenarios and evaluated their performance based on the following metrics.

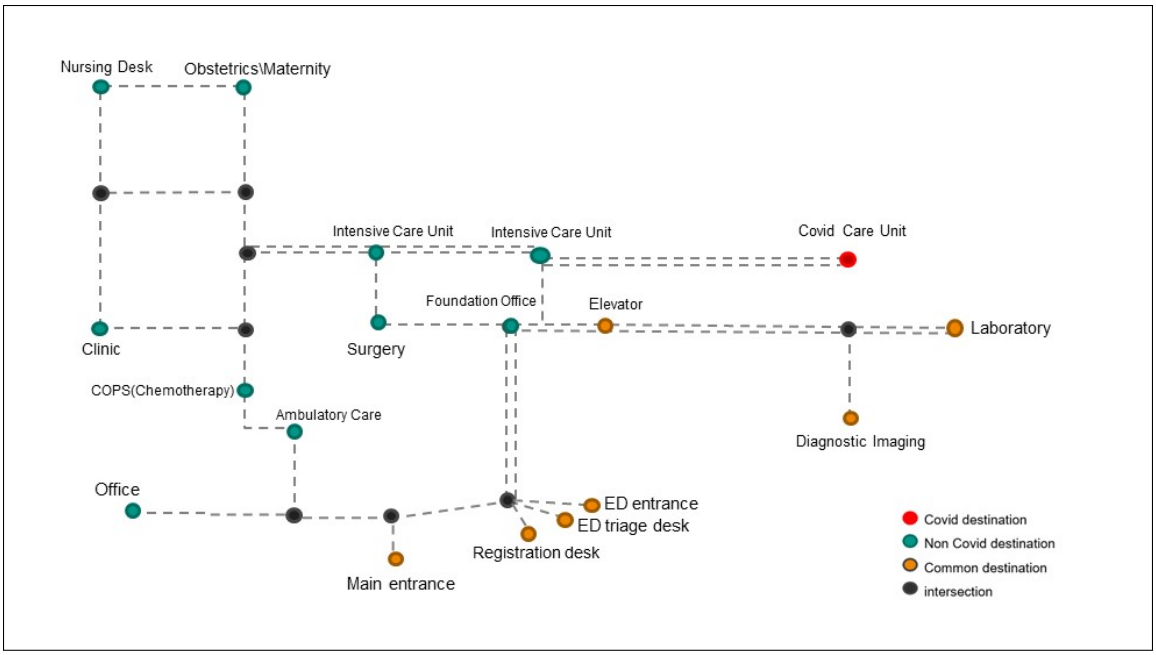


Figure 3.4 The corresponding graph of the second floor

- The number of meets between patients in each scenario: refers to the count of meets where two patients come into contact with each other. A meet can occur when patients cross paths, are in close proximity, or share a common space, such as a waiting area or corridor.
- The maximum patient waiting time for each experiment: refers to the longest interval of time a patient has to wait in the corridors.
- The total waiting time for each experiment: refers to the sum of the waiting times of all patients in a single experiment. This metric represents the overall time that patients spent waiting in corridors during their visit.
- Maximum distance between the source and destination $d(s,t)$: refers to the longest distance between a starting point (source) and an end point (destination) in a hospital.

Table 3.2 Simulation Parameters

| Parameter | Value | | | | | |
|---|---|---|---|---|---|---|
| Number of patients | 100 | 200 | 300 | 400 | 500 | 600 |
| Number of Covid-19 patients | 33 | 65 | 100 | 132 | 165 | 198 |
| Patient speed | 0.5 m/s | | | | | |

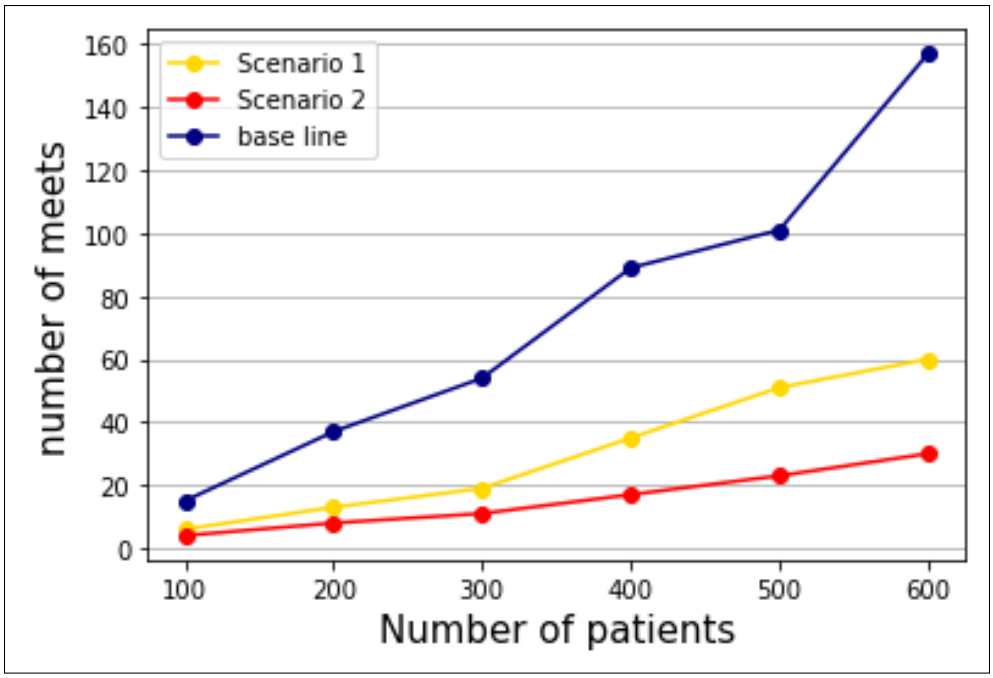


Figure 3.5 Number of meets between patients

### 3.3 Baseline

In order to better understand the effectiveness of the two scenarios, we set a baseline using a Dijkstra algorithm (Samah *et al.*, 2015). The Dijkstra algorithm is a popular shortest-path algorithm used to find the shortest path between two nodes in a graph. It selects the node with the lowest cost and repeatedly updates the cost of neighboring nodes until the destination node is reached. The algorithm is commonly used in routing, navigation, and transportation systems.

This method is frequently used in calculating the shortest route in a building and has been modified to incorporate the next node state into route selection. The performance of the Dijkstra algorithm can be compared to our scenarios to understand the performance.

### 3.4 Comparison between two scenarios

Fig. 3.5 compares the numbers of meets achieved respectively by offline, online scenario, and the baseline. We can see Scenario 2 has a better result than Scenario 1 and the baseline. It is worth noting here that all meets are the unavoidable meets, e.g., the nodes associated with a bridge.

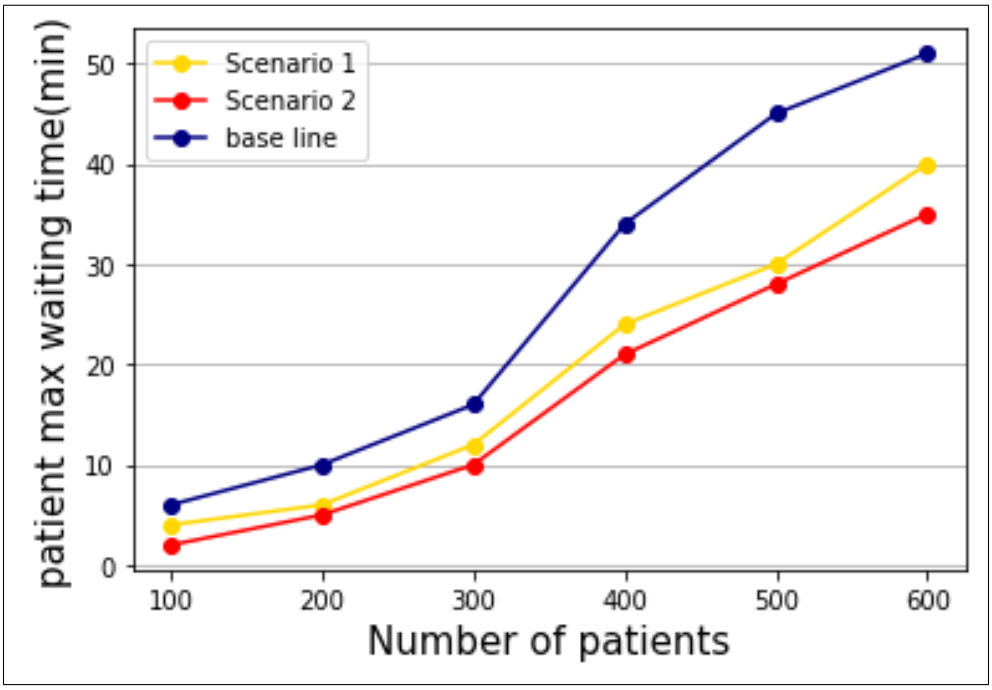


Figure 3.6 Comparison of patient max waiting time per minute

In Fig. 3.6 although Scenario 2 reduces waiting time efficiently, finding a suitable path for each patient becomes more challenging if the number of patients increases. Moreover, because of the high probability of patients crossing the corridors (i.e., patients moving in opposite directions of a corridor at the same time), the patient waiting time becomes longer.

Fig. 3.7 show the result of patient total waiting time spent in the corridors.

In Scenario 1, we define a specific tour for patients and guide them through these edges at any time. Therefore, when the number of patients is low, the patient waiting time is longer than in Scenario 2. In Scenario 2, the patient can access an empty corridor immediately; unlike

Scenario 1, the patient does not need to wait for the path to be open or move away from other patients (to respect social distancing) before reaching the next corridor.

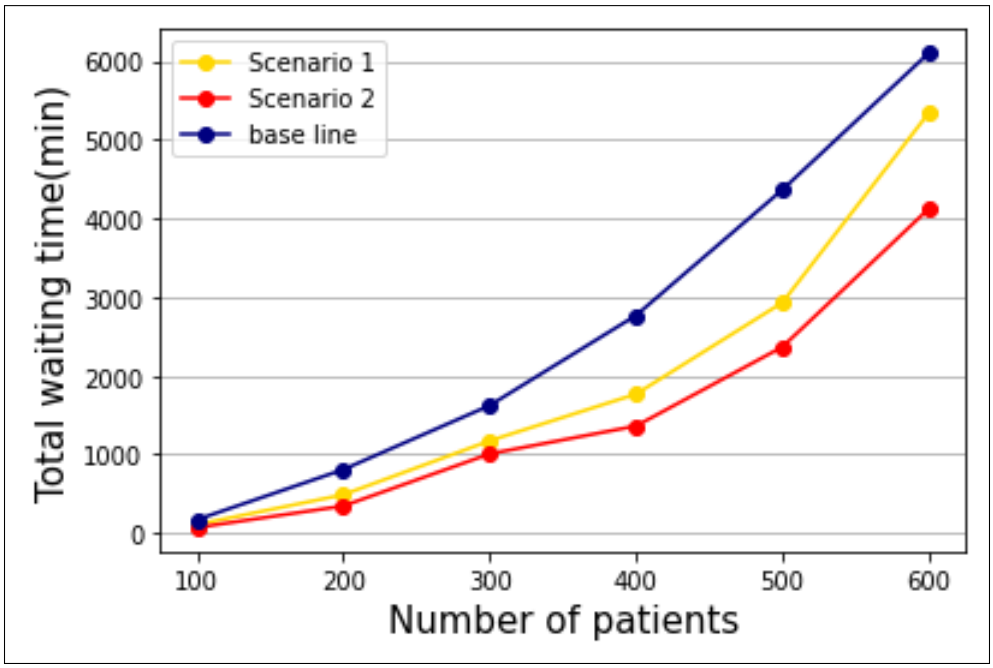


Figure 3.7 Comparison of patient total waiting time per minute

Fig. 3.8 compares the maximum distance between the source and destination. In Scenario 2, patients have to travel a longer distance than in Scenario 1 to reach their destination. Although the baseline achieves the shortest distance for the patient, it has a higher risk of the meets between patients and increases the risk of prorogation of the virus.

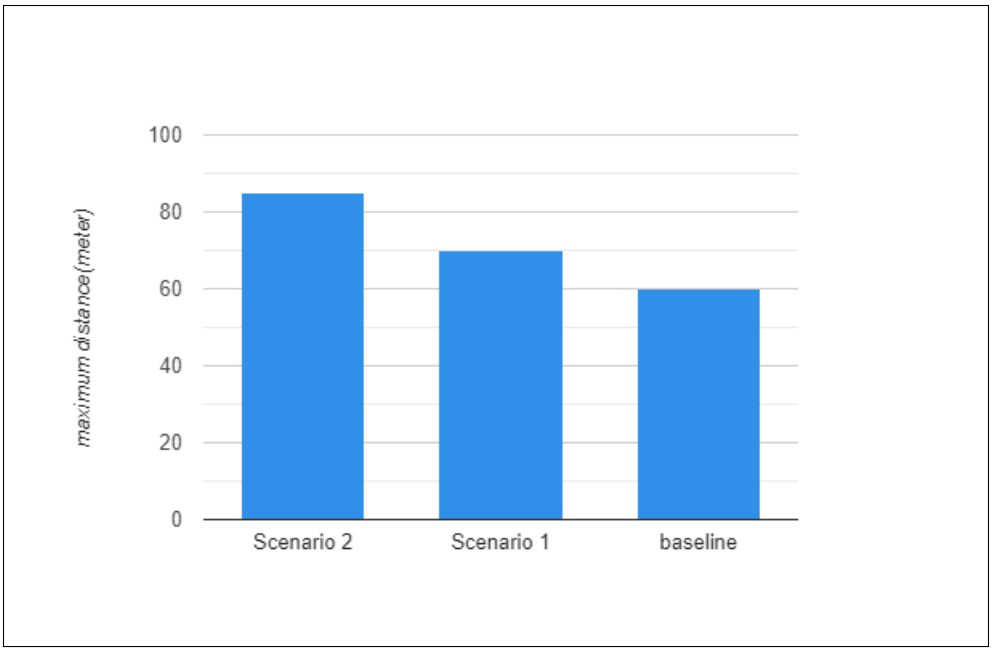


Figure 3.8 Comparison of the Maximum $d(s,t)$

# CONCLUSION AND RECOMMENDATIONS

In this thesis, we propose a new solution for path-finding in the healthcare environment during the pandemic. We present two scenarios as offline and online path-finding, in which the requirements of two flows of patients (COVID-19 and Non-COVID) are considered. In the first scenario, our model consists of finding the tour for each patient flow that *(i)* minimizes the number of two-ways corridors and *(ii)* maximize disjoint paths, while maintaining the reachability from each source to the desired destinations. In order to obtain the optimized solution in near real-time, we propose an algorithm based on the well-known Christofides' algorithm. In the second scenario, we studied an online shortest path algorithm to dynamically find a path, with a path definition that is dynamically redefined at each corridor intersection, depending on the instantaneous flow of patients. Both scenarios have been validated using simulation tools. Their performance may vary depending on the floor map of each hospital and the associated patient flows. In any case, both algorithms meet the patient flow requirements while providing the best possible circulation paths. In particular, both algorithms can identify disjoint paths as much as possible in order to reduce the number of patient meets.

Furthermore, this study can significantly impact the hospital sector, especially with the high tendency for patient flows in healthcare environments. Also, it can enhance the existing intelligent technologies in hospitals.

This study can be extended by relaxing existing assumptions in the scenarios and the simulation. It is possible to vary the speed at which patients move through the corridors, particularly for elderly patients. The capacity of hospital waiting rooms determines the maximum number of patients who can access the corridors at any given time. However, hospitals can optimize their waiting rooms to accommodate a higher number of patients. To efficiently manage patient flow, hospitals can analyze real data to determine the source and destination of each patient.

Further research can explore the potential of integrating the proposed algorithm with other intelligent technologies, such as machine learning-based predictive models, to improve the accuracy and efficiency of routing and path-finding decisions. Another area for future work is to evaluate the feasibility and effectiveness of implementing the proposed algorithm in real-world hospital environments. This involves conducting pilot studies in selected hospitals to assess the practicality of the algorithm and to identify any potential challenges or limitations that need to be addressed. The results of such studies can inform the further refinement and optimization of the algorithm to ensure its practicality and effectiveness in real-world hospital settings.

Furthermore, there is potential for the proposed algorithm to be extended beyond the scope of the COVID-19 pandemic. For example, it can be applied to improve routing and path-finding decisions in hospitals during normal operations, to optimize resource allocation and improve patient outcomes. Additionally, the algorithm can be adapted for use in other settings, such as emergency response and disaster management, to improve the efficiency of emergency response and evacuation operations. The proposed scenarios can also be used in other public environments (e.g., supermarkets or manufacturing production/warehouse floors) in order to minimize the number of customer/worker meets, which is of interest in a pandemic environment.

This research was conducted during a productive internship at MITACS and Humanitas (IT 19243), the critical link between the private sector and post-secondary institutions, where it has gained extensive attention.

# APPENDIX I

## ARTICLES PUBLISHED IN CONFERENCES

We have presented the main content of this thesis in a conference paper.

- "Optimized Circulation Management In Hospitals During COVID-19" which has been accepted in IEEE International Conference on Communications (ICC): SAC E-Health Track 2023.

## BIBLIOGRAPHY


Akiba, T. et al. (2013). Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 909–918.

Al-Gaphari, G. H. et al. (2021). Discrete crow-inspired algorithms for traveling salesman problem. *Engineering Applications of Artificial Intelligence*, 97, 104006.

Ali, I. M. et al. (2020). A novel design of differential evolution for solving discrete traveling salesman problems. *Swarm and Evolutionary Computation*, 52, 100607.

Applegate, D. et al. (2007). *The Traveling Salesman Problem - A Computational Study*. Princeton University Press.

Arkin, E. et al. (2002). A Note on Orientations of Mixed Graphs. *Discrete Applied Mathematics*, 116(3), 271–278.

Asadpour et al. (2017). An O (log n/log log n)-approximation algorithm for the asymmetric traveling salesman problem. *Operations Research*, 65(4), 1043–1061.

Authority, S. H. (2022). *Saskatchewan Health Authority floors map*. https://paphr.ca/.

Baniasadi, P. et al. (2020). A transformation technique for the clustered generalized traveling salesman problem with applications to logistics. *European Journal of Operational Research*, 285(2), 444–457.

Bleichrodt et al. (2022). The Edge Orientation Problem with Origin-Destination Pairs.

Boccia et al. (2021). An exact approach for a variant of the FS-TSP. *Transportation Research Procedia*, 52, 51–58.

Bock, S. & Klamroth, K. (2019). Combining Traveling Salesman and Traveling Repairman Problems: A multi-objective approach based on multiple scenarios. *Computers & Operations Research*, 112, 104766.

Boryczka, U. et al. (2019). An effective hybrid harmony search for the asymmetric travelling salesman problem. *Engineering Optimization*.

Cacchiani, V. et al. (2020). Models and algorithms for the Traveling Salesman Problem with Time-dependent Service times. *European Journal of Operational Research*, 283(3), 825–843.

Chekuri, C. & Quanrud, K. (2018). Fast approximations for metric-TSP via linear programming. *arXiv preprint arXiv:1802.01242*.

Chowdhury, S. et al. (2019). A modified Ant Colony Optimization algorithm to solve a dynamic traveling salesman problem: A case study with drones for wildlife surveillance. *Journal of Computational Design and Engineering*, 6(3), 368–386.

Christofides, N. (1976). *Worst-case analysis of a new heuristic for the travelling salesman problem*.

Cormen, T. H. o. (2022). *Introduction to algorithms*. MIT press.

de Moraes, R. S. et al. (2019). Experimental analysis of heuristic solutions for the moving target traveling salesman problem applied to a moving targets monitoring system. *Expert Systems with Applications*, 136, 392–409.

Defryn, C. et al. (2018). Multi-objective optimisation models for the travelling salesman problem with horizontal cooperation. *European Journal of Operational Research*, 267(3), 891–903.

Dib, O., Manier, M.-A. & Caminada, A. (2015). Memetic algorithm for computing shortest paths in multimodal transportation networks. *Transportation Research Procedia*, 10, 745–755.

Dong, X. et al. (2019). Artificial bee colony algorithm with generating neighbourhood solution for large scale coloured traveling salesman problem. *IET Intelligent Transport Systems*, 13(10), 1483–1491.

Duraj, L. (2010). *Optimal graph orientation problems*. (Ph.D. thesis).

Dutta, K. & Subramanian, C. (2016). Improved bounds on induced acyclic subgraphs in random digraphs. *SIAM Journal on Discrete Mathematics*, 30(3), 1848–1865.

Ebadinezhad, S. (2020). DEACO: Adopting dynamic evaporation strategy to enhance ACO algorithm for the traveling salesman problem. *Engineering Applications of Artificial Intelligence*, 92, 103649.

Elgesem, A. S. et al. (2018). A traveling salesman problem with pickups and deliveries and stochastic travel times: An application from chemical shipping. *European Journal of Operational Research*, 269(3), 844–859.

Emami Taba, M. S. (2010). *Solving traveling salesman problem with a non-complete graph*. (Master's thesis, University of Waterloo).

Filomena, G. et al. (2021). Modelling the effect of landmarks on pedestrian dynamics in urban environments. *Computers, Environment and Urban Systems*, 86, 101573.

Franz, G. et al. (2005). Graph-based models of space in architecture and cognitive science: A comparative analysis. *17th International Conference on Systems Research, Informatics and Cybernetics (INTERSYMP 2005)*, pp. 30–38.

Gencel, C. A. et al. (2019). Traveling salesman problem with hotel selection: Comparative study of the alternative mathematical formulations. *Procedia Manufacturing*, 39, 1699–1708.

Gould, R. (2012). *Graph theory*. Courier Corporation.

Hagberg, A. et al. (2008). *Exploring network structure, dynamics, and function using NetworkX*.

Harrigan, M. P., Sung, K. J., Neeley, M., Satzinger, K. J., Arute, F., Arya, K., Atalaya, J., Bardin, J. C., Barends, R., Boixo, S. et al. (2021). Quantum approximate optimization of non-planar graph problems on a planar superconducting processor. *Nature Physics*, 17(3), 332–336.

He, S.-x. et al. (2021). An Approach to Solving the General Traveling Salesman Problem Based on Generating and Combining Cycles. *Available at SSRN 3980642*.

Hoffman, K. L. et al. (2013). Traveling salesman problem. *Encyclopedia of operations research and management science*, 1, 1573–1578.

Hu, Y., Yao, Y. & Lee, W. S. (2020). A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs. *Knowledge-Based Systems*, 204, 106244.

Huang, S. & Wolkowicz, H. (2018). Low-rank matrix completion using nuclear norm minimization and facial reduction. *Journal of Global Optimization*, 72, 5–26.

Jensen, C. S. et al. (2009). Graph model based indoor tracking. *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, pp. 122-131.

Jiang, C. et al. (2020). A new efficient hybrid algorithm for large scale multiple traveling salesman problems. *Expert Systems with Applications*, 139, 112867.

Kaspi, M. et al. (2019). Maximizing the profit per unit time for the travelling salesman problem. *Computers & Industrial Engineering*, 135, 702–710.

Kucukoglu et al. (2019). Hybrid simulated annealing and tabu search method for the electric travelling salesman problem with time windows and mixed charging rates. *Expert Systems with Applications*, 134, 279–303.

Kudelić, R. & Ivković, N. (2019). Ant inspired Monte Carlo algorithm for minimum feedback arc set. *Expert Systems with Applications*, 122, 108–117.

Lancaster, E. M. et al. (2020). Rapid response of an academic surgical department to the COVID-19 pandemic: implications for patients, surgeons, and the community. *Journal of the American College of Surgeons*, 230(6), 1064–1073.

Lawler, E. et al. (1991). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley.

Li, Y. et al. (2005). Relationships between network topology and pedestrian route choice behavior. *Journal of the Eastern Asia Society for Transportation Studies*, 6, 241–248.

Liu, X.-j., Yi, H. & Ni, Z.-h. (2013). Application of ant colony optimization algorithm in process planning optimization. *Journal of Intelligent Manufacturing*, 24(1), 1.

Mahapatra, R. et al. (2021). Generalized neutrosophic planar graphs and its application. *Journal of Applied Mathematics and Computing*, 65(1-2), 693–712.

Majeed, A. & Rauf, I. (2020). Graph theory: A comprehensive survey about graph theory applications in computer science and social networks. *Inventions*, 5(1), 10.

Miller, M. et al. (2006). Using RFID technologies to capture simulation data in a hospital emergency department. 1365–1371.

Mosayebi, M. et al. (2021). The traveling salesman problem with job-times (tspj). *Computers & Operations Research*, 129, 105226.

Nasir, M. et al. (2014). Prediction of pedestrians routes within a built environment in normal conditions. *Expert Systems with Applications*, 41(10), 4975–4988.

Oche, M. & Adamu, H. (2013). Determinants of patient waiting time in the general outpatient department of a tertiary health institution in North Western Nigeria. *Annals of medical and health sciences research*, 3(4), 588–592.

Ottoni, A. L. et al. (2022). Reinforcement learning for the traveling salesman problem with refueling. *Complex & Intelligent Systems*, 8(3), 2001–2015.

Pandiri, V. et al. (2018). A hyper-heuristic based artificial bee colony algorithm for k-interconnected multi-depot multi-traveling salesman problem. *Information Sciences*, 463, 261–281.

Pedro, O. et al. (2013). A tabu search approach for the prize collecting traveling salesman problem. *Electronic Notes in Discrete Mathematics*, 41, 261–268.

Pina-Pardo, J. C. et al. (2021). The traveling salesman problem with release dates and drone resupply. *Computers & Operations Research*, 129, 105170.

Robbins, H. (1939). A theorem on graphs, with an application to a problem of traffic control. *The American Mathematical Monthly*, 46(5), 281 – 283.

Roberts, F. S. (1978). The One-Way Street Problem. In *Graph Theory and Its Applications to Problems of Society* (ch. 2, pp. 7-13). doi: 10.1137/1.9781611970401.ch2.

Saji, Y. & Barkatou, M. (2021). A discrete bat algorithm based on Lévy flights for Euclidean traveling salesman problem. *Expert Systems with Applications*, 172, 114639.

Samah, K. et al. (2015). Modification of dijkstra's algorithm for safest and shortest path during emergency evacuation. *Applied Mathematical Sciences*, 9(31), 1531–1541.

Schermer, D., Moeini, M. & Wendt, O. (2020). A b ranch-and-cut approach and alternative formulations for the traveling salesman problem with drone. *Networks*, 76(2), 164–186.

Schmidt, J. M. (2013). A simple test on 2-vertex-and 2-edge-connectivity. *Information Processing Letters*, 113(7), 241–244.

Seidel, R. (1995). On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of computer and system sciences*, 51(3), 400-403.

Sniedovich, M. (2010). *Dynamic programming: foundations and principles*. CRC press.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2), 146 – 160.

Tong, Y. & Bode, N. W. (2022). The principles of pedestrian route choice. *Journal of the Royal Society Interface*, 19(189), 20220061.

Toth, P. et al. (2002). *The vehicle routing problem*. SIAM.

Trindade, A. et al. (2018). A GIS-based analysis of constraints on pedestrian tsunami evacuation routes: Cascais case study (Portugal). *Natural Hazards*, 93(Suppl 1), 169–185.

Trudeau, R. J. (2013). *Introduction to graph theory*. Courier Corporation.

Wang, S., Wei, X., Nogueira dos Santos, C. N., Wang, Z., Nallapati, R., Arnold, A., Xiang, B., Yu, P. S. & Cruz, I. F. (2021). Mixed-curvature multi-relational graph neural network for knowledge graph completion. *Proceedings of the Web Conference 2021*, pp. 1761–1771.

Wang, S. et al. (2020). Approximate and exact algorithms for an energy minimization traveling salesman problem. *Journal of Cleaner Production*, 249, 119433.

Wang, X. et al. (2019). A steiner zone variable neighborhood search heuristic for the close-enough traveling salesman problem. *Computers & Operations Research*, 101, 200–219.

Zhou, T. et al. (2012). Kernelized probabilistic matrix factorization: Exploiting graphs and side information. *Proceedings of the 2012 SIAM international Conference on Data mining*, pp. 403–414.

Zhou, Z. et al. (2022). HiVG: A hierarchical indoor visibility-based graph for navigation guidance in multi-storey buildings. *Computers, environment and urban systems*, 93, 101751.