NHH

Norwegian School of Economics
Bergen, Fall 2023

Cruise Tourist Management

A Mixed Integer Linear Programming Model for Allocating Cruise Tourists in Bergen City Centre

Nikoline Valaker and Adna Alibasic

Supervisor: Andres Velez and Stein W. Wallace

Master thesis, Economics and Business Administration

Major: Business Analytics (BAN)

NORWEGIAN SCHOOL OF ECONOMICS

Acknowledgements

This thesis is written as a part of our Master of Science in Economics and Business Administration, with a major in Business Analytics at Norwegian School of Economics. Despite the topic being challenging academically, it has been a rewarding and educational semester.

We would like to thank our main supervisor Stein W. Wallace for great guidance and continuous support. We would also like to express our gratitude to PhD candidate Andres Velez for suggesting an interesting topic and for great support and motivation throughout the semester. Furthermore, we would like to thank Nils Møllerup at Bergen Havn for providing us with the necessary insight and information about the cruise traffic and cruise tourism in Bergen.

In addition, we would like to thank our friends and family for their continuous support throughout our master's degree. Last but not least, we are very grateful for the good collaboration between the two of us.

Norwegian School of Economics

Bergen, December 2023

Nikoline Valaker

Adna Alibasic

Abstract

As Norway's largest cruise port, Bergen allows up to 8,000 cruise tourists to visit per day. Effectively allocating these tourists is crucial to prevent queues and congestion while simultaneously maximizing the tourists' satisfaction. The purpose of this master's thesis is therefore to develop an optimization model which aims to achieve these objectives.

This master thesis utilizes a model based on the Travelling Salesman Problem to generate a set of visiting routes composed of a selection of locations to visit within the city centre. These locations are based on recommendations by Visit Bergen and Tripadvisor, and information regarding them is retrieved through API calls. Further, a mixed integer linear programming model is developed to assign tourist groups to routes maximizing satisfaction while simultaneously preventing congestion. By gathering information from the tourists, the model aims to account for individual preferences and time limits when assigning tourist groups to routes.

The result of this thesis is a simplified mixed integer linear programming model utilizing a set of predefined routes for the tourist groups to be assigned to. The proposed model successfully allocates a test population, consisting of a set of tourist groups. It ensures an optimal allocation of each tourist group by maximizing overall satisfaction and preventing congestion. However, analysis shows that a real-life application of the model, aiming to assign potentially 8,000 tourists divided into groups to predefined routes, is computationally challenging despite the implemented simplifications. A balanced trade-off between computational time and the objective is therefore required.

Keywords – Cruise tourism, Bergen, Tourist allocation, Mixed Integer Linear Programming, Optimization

Contents

List of Figures

List of Tables

Abbreviations and explanations of terms

Location Collective term for both attractions, restaurants and cafés

User A tourist group using the application interface to supply with input to the model

MP Mathematical Programming

LP Linear Programming

ILP Integer Linear Programming

MILP Mixed Integer Linear Programming

TSP Travelling Salesman Problem

VRP Vehicle Routing Problem

OSSP Open Shop Scheduling Problem

JSSP Job Shop Scheduling Problem

NP-hard Non-Deterministic Polynomial-time hardness

Predecessor location The location that comes before a given location

Successor location The location that comes after a given location

1 Introduction

1.1 Background

Bergen, recognized as the most prominent tourist destination in Western Norway, plays a pivotal role in the region's economy. The economic impact of tourism in Bergen and its neighbouring municipalities is substantial, with a value creation in 2019 estimated at approximately 7.2 billion NOK (NHO Reiseliv, 2019) (VisitBergen, n.d.-c). Among the many tourists visiting Bergen, cruise tourists represent a notable fraction. As Norway's largest cruise port, Bergen welcomed 326 cruise ships in the pre-COVID-19 era, attracting approximately 576,000 passengers in 2019 (Amland, 2021).

A significant portion of the cruise traffic in Bergen occurs in the same period as other holiday traffic between May and August (Amland, 2021). Given Bergen's relatively compact unique urban structure, this intense influx often results in congestion and queues, particularly in the city centre and around key tourist spots. Efficiently managing the presence of cruise tourists is crucial to enhancing visitor satisfaction and sustaining the economic benefits this industry brings to the city. Overcrowding and excessive queues may lead to dissatisfaction among both tourists and locals, potentially impacting the value creation adversely. Long queues may prevent people from purchasing what they desire, which implies both lost sales and dissatisfied customers.

The negative consequences of tourism are not caused by the cruise traffic alone. However, cruise tourism is perceived as more visually visible than other types of tourism due to both the large ships and the cruise tourists' concentrated nature (Amland, 2021). Furthermore, cruise passengers largely come together and are concentrated in the city centre of Bergen due to their limited time docked to quay. To mitigate this, Bergen City Port has implemented a restriction, capping the number of cruise ships to three per day, with a total passenger limit of 8,000 (Amland, 2021). Despite this restriction, the simultaneous arrival of 8,000 passengers can still lead to overcrowding and congestion if not managed properly.

An analysis carried out by Amland Reiselivsutvikling on the future management of cruise passengers underlines the necessity of spreading the tourists within the city centre of

Bergen (Amland, 2021). Today, the cruise traffic into Bergen is distributed between the Bontelabo area and Jekteviken. However, with plans underway to develop a new city district at Jekteviken, it is anticipated that all cruise activity currently at Jekteviken will eventually be consolidated at Bontelabo (Kringstad, 2023). This change underscores the urgency of devising an effective strategy for tourist allocation, as such a consolidation to a single area will further amplify the need for efficient tourist management in the city.

1.2 Problem description

This master's thesis addresses the challenge of queues and congestion at tourist destinations in Bergen, with a focus on cruise tourists. The objective of the thesis is to construct a model that aims to maximize the satisfaction of the visiting cruise tourists while aligning with the wishes of Bergen municipality. This includes preventing congestion at different attractions.

The purpose of this master's thesis is to develop an optimization model which allocates cruise tourists in the city centre of Bergen, optimizing the tourist's satisfaction and at the same time preventing congestion to satisfy the municipality of Bergen.

The thesis is mainly divided into two parts. The first part involves identifying and establishing the relevant locations to include in the model. These locations will primarily be selected based on recommendations from the tourist organization in Bergen, Visit Bergen, to account for the wishes of the municipality. Further, each location will be given a score that serves as the foundation of the optimization model.

The second part of the thesis will focus on constructing the optimization model. It is envisioned that an application interface will be the main support for the input of the model. Through the application interface, tourists will be providing crucial input about for instance their time available and their personal preferences, for the model to run. The model will seek to provide each user with a visiting route suggestion that will be presented to the user through the application interface. A suggested application interface is presented in Figure B.1-B.3 in the Appendix as an outline for how the app could look for the user, both before any selections are made and after the model has generated a solution in response to all user preferences in total.

The master's thesis is divided into eight sections, where the current section serves as an introduction. Sections 2 and 3 constitute the first part of the thesis. Section 2 aims to present the data gathering process for the input of the model, and section 3 describes how the locations are weighted by providing them with a score.

Sections 4 and 5 constitute the second part of the thesis. Firstly, the relevant theoretical framework of the thesis is presented in Section 4 and then applied in Section 5 where the proposed mathematical optimization model is elaborated. An analysis that aims to evaluate the behaviour of the proposed model is then conducted in Section 6 under certain parameter tunings. Lastly, limitations and suggestions for further research are discussed in Section 7 with a conclusion drawn in Section 8 based on the discussion.

1.3 Delimitations

While tourism in Bergen attracts a diverse range of visitors, the model is intended to exclusively allocate those arriving by cruise ships. This choice is made due to the practical advantages associated with cruise tourists. Unlike the ones arriving by plane, bus, or car, cruise tourists arrive as a collective group, sharing the same point of embark, which facilitates management and control more easily. Additionally, a survey conducted by Innovation Norway in 2019, reveals that cruise tourists experience overtourism to a greater extent compared to other tourists in Bergen (Innovation Norway, 2019, p. 39). Therefore, an effective allocation of cruise tourists is of crucial importance.

In the proposed model, the selection of locations is concentrated on the most popular tourist locations within the city centre of Bergen. It is important to acknowledge that Bergen holds a multitude of attractions, places to visit, and places to eat. However, for this thesis, it is necessary to narrow the scope. The inclusion of all possible locations would result in a model of unwieldy proportions. Therefore, the cruise tourists who travel by bus outside the city centre are not considered in the model. This is also justified by analyses conducted by Amland Reiselivsutvikling, revealing the assembly points are located in the city centre of Bergen (Amland, 2021, pp. 5–6). Furthermore, the prices at the different locations are held outside the model for this thesis, but it will be discussed for further research in Section 7.

2 Data gathering

This section of the thesis will outline the data gathering process for the model. The initial phase involves gathering information on the cruise tourists' behaviour, selecting various locations to include in the model, as well as collecting the necessary data for the different locations. The latter entails retrieving information on distances and travel times between the locations, opening hours, pictures posted, and visitor reviews and ratings. In the following, the different parts of the data gathering process will be presented, beginning with elaborating the location selection procedure.

2.1 Information on cruise tourists in general

For the development of the model, it is necessary to obtain certain key metrics on the cruise tourist patterns and behaviour in general. These will be influential in the shaping of the model.

An analysis conducted by Bergen Havn reveals that the average occupancy rate on cruise ships in June 2023 equals approximately 83%. Of these, approximately 80% disembark the ship during sunny weather to go to the city centre and approximately 60-65% when it is raining (Møllerup, 2023). Hence, the number of tourists to take into account in the model is significantly smaller when it is raining. Furthermore, the analysis highlights the times during the day the cruise tourists are present in the city centre of Bergen. Regardless of the weather conditions, at least 50% of the cruise passengers who disembark the ship are present in the city between 10 AM and 2 PM (Møllerup, 2023).

In the analysis conducted by Amland Reislivsutvikling, the traffic flows from the cruise ships are studied. Figure 2.1 illustrates the traffic flow on the 6th of June 2019, and visualizes the most common assembly points, marked as red dots. The crowding seems to occur in the city centre, specifically at Bergenhus Festning, the area of Bryggen, Fisketorget, Floibanen, and at Torgalmenningen (Amland, 2021). This is taken into account when selecting the locations for the model.

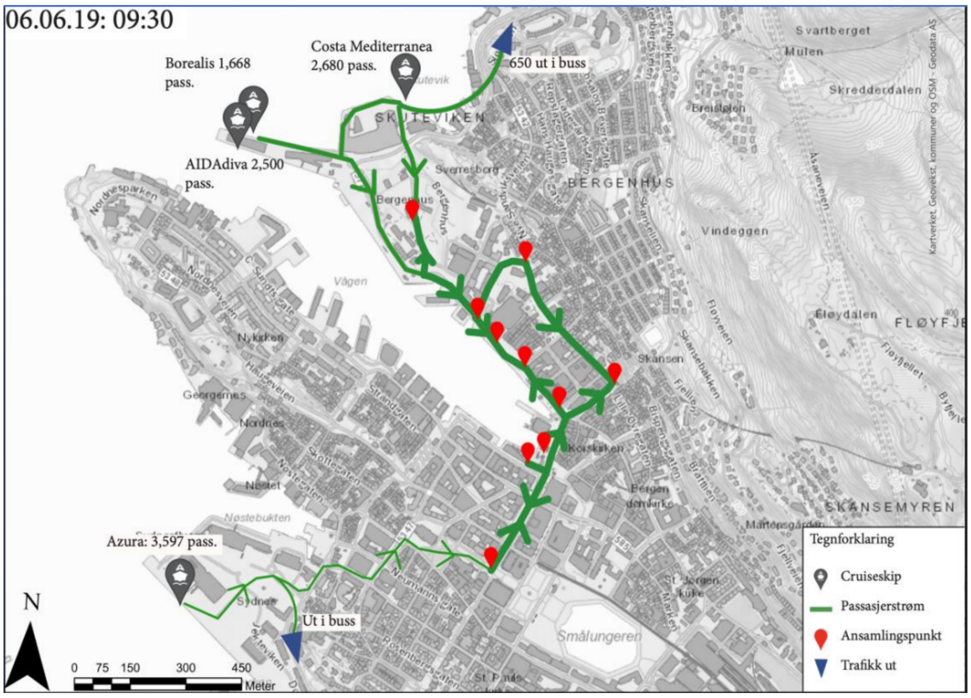


Figure 2.1: Illustration, traffic flows from cruise ships to the city centre, 6.6.19 (Amland, 2021, p. 6)

It is beneficial to acquire an understanding of the time of the day the cruise tourists arrive in Bergen and the duration of their stay. The analysis by Amland reveals that 68% of the cruise ships arrived between 7 AM and 9 AM in 2021. Forecast numbers for the same year revealed that 67% of all port calls last between 8 to 10 hours, while numbers from 2019 revealed 59% of all port calls last between 8 to 10 hours (Amland, 2021, p. 7). Of the hours the cruise ships are docked to the quay, a survey carried out by Innovation Norway reveals that on average 46% of the tourists are ashore for 2-5 hours and 41% are ashore for 5-8 hours (Innovation Norway, 2019).

Furthermore, an understanding of tourists' willingness to wait in queue and to walk between locations is advantageous. However, the research on these topics shows that determining these measures is situational, and depends on various factors to a large extent. For instance, waiting in queue can be acceptable and even expected at certain attractions, while at others it can be a major factor for dissatisfaction (Houston et al., 1999). Considering the distances between attractions in the city centre of Bergen, it is reasonable to assume the cruise tourists visiting Bergen are willing to walk for approximately 25 minutes between two consecutive locations.

2.2 Selecting locations

Bergen holds several tourist locations, yet considering all of them would be a time-consuming task. Furthermore, this might not necessarily yield superior results compared to a selective inclusion of specific locations. There are many potential ways to identify the most relevant ones, but it is essential to ensure the locations offer a diverse range of experiences to meet as many needs as possible. In this thesis, locations are defined as places tourists can be allocated to. A location in this context is therefore a collective term for both attractions, restaurants and cafés.

Visit Bergen is a tourist organization for Bergen (VisitBergen, n.d.-a). They focus, among other things, on marketing the city and the region, and work closely with the municipality of Bergen and Bergen City Port. The list of locations used in the model is mainly based on the recommendations from Visit Bergen, to meet both the tourists' and the municipality's preferences.

To improve the user experience of the application interface, the locations are grouped into several categories that the user can select their preferences from. This allows the user to prefer a category in general and not necessarily a specific location. There is a possibility for each location to be associated with several categories simultaneously. The categorization framework is based on the suggested categories for attractions in Bergen from the website of Visit Bergen (VisitBergen, n.d.-b). For this thesis the following categories are selected; *Museums in Bergen*, *Family Friendly Attractions in Bergen*, *Bergen Galleries* and *Churches in Bergen*. However, *Museums in Bergen* and *Bergen Galleries* are merged to *Museums & Galleries* since there are only three suggestions for galleries, and these categories may overlap. The excluded categories are justified by the overlap in the selected categories or the long distance to the city centre.

Furthermore, the selection of categories the tourists may choose from has been enriched with categories; *Historical sites*, *Activities* and *Nature*. We consider these to represent important aspects of what Bergen has to offer for the tourists. These categories hold locations across the different categories on VisitBergen's website. *Historical sites* encompasses locations of regional historical significance, *Activities* consist of engaging experiences and *Nature* focuses on scenic attractions surrounded by nature. Lastly, the category *Food* is added with subcategories *Retaurats* and *Cafés.* For further research, one could add divisions within the category *Restaurants* corresponding to the type of food served at the restaurants to allow for more refined preferences without the need to identify specific restaurants. The final set of categories is listed in Table 2.1.

Table 2.1: Selection of categories

Category	Subcategory
Museums & Galleries	—
Churches	—
Family Friendly	—
Historical Sites	—
Activities	—
Nature	—
Food	Restaurants, Cafés

Locations that are outside a one-kilometer radius of what Visit Bergen considers the city centre are excluded. This decision is based on both the scope of the thesis and on the numbers from 2019 indicating that 59% of the cruise ships were docked at the quay for approximately 8-10 hours. Allocating the tourists to locations outside the city centre would potentially result in inefficient use of the cruise tourists' limited time.

Given the large number of suggested restaurants, only restaurants with a minimum of four in rating from Tripadvisor are included, yielding a total of 28 restaurants. Tripadvisor is a website that offers recommendations for tourists, allowing individuals to give feedback on locations they have visited (Tripadvisor, n.d.). This selection is justified by the fact that the objective is to maximize the satisfaction of the cruise tourists and then it is convenient to select the best alternatives.

The list of cafés on Visit Bergen was rather limited. Hence, the selection of cafés is based on the sorted list of "Cafés in Bergen" obtained from Tripadvisor on the 31st of October. (Tripadvisor, 2023). The top 28 cafés are included in the model to match the number of restaurants.

It is important to clarify that the selection of restaurants and cafés is not based on recommendations by the municipality of Bergen, but rather on the ratings from Tripadvisor. The final selection of categories and belonging locations is presented in Table A.1, A.2 and A.3 in the Appendix.

2.3 Location details

Based on the chosen locations, information about each is retrieved using a Google API called Places API (Google Maps, n.d.). An API, which stands for *Application Programming Interface*, is an interface that enables communication between different software components (Rossen & Nätt, 2023). This API allows one to search for each location's details using an individual search text. Hence, a detailed search text is made for each of the locations to make sure the information retrieved is from the correct location. The received information includes, among others, *location coordinates*, *visitor ratings*, *visitor reviews*, and *opening hours* if available. The response also includes information about whether the location is operational or not. Including locations that are temporarily or permanently closed when generating routes is not fortunate for the tourists. Hence, these locations are removed from the list.

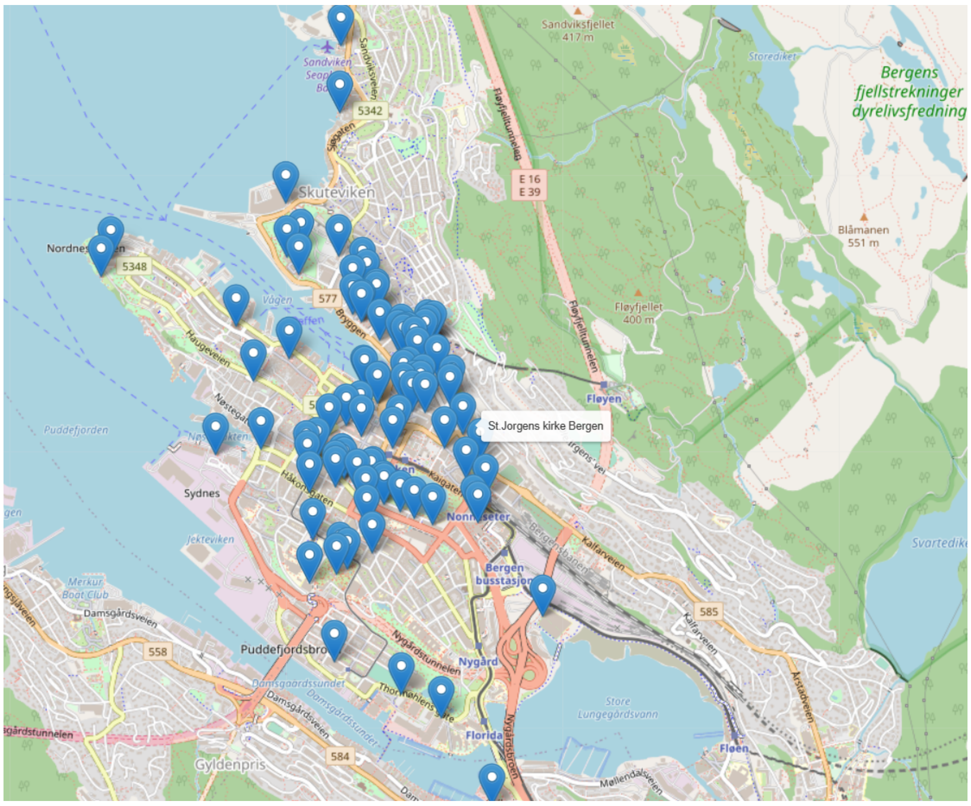


Figure 2.2: Map of locations

It is observed that certain locations have defined opening hours, regardless of being physically accessible at any time. An example of such a location is Bryggen, which despite being a location that may be visited at any hour of the day, officially lists its opening hours as 9 AM to 4 PM on Google. While Bryggen and similar locations are accessible at all hours, the official opening times may indicate when services or full visitor experiences are available. Given the scope of this thesis, the opening hours listed on Google are still put in use as a result of limiting the data gathering process.

2.4 Distance and travel time between locations

To prevent dissatisfied tourists due to excessive time spent in transit and to include as many locations as possible within the available time, it is necessary to keep track of the walking durations between two linked locations. The shortest path between Bontelabo and KODE Stenersen is illustrated as an example of a walking route, using OpenStreetMap, an open-source map (Boeing, 2017, pp. 126–139), and visualized in Figure 2.3.

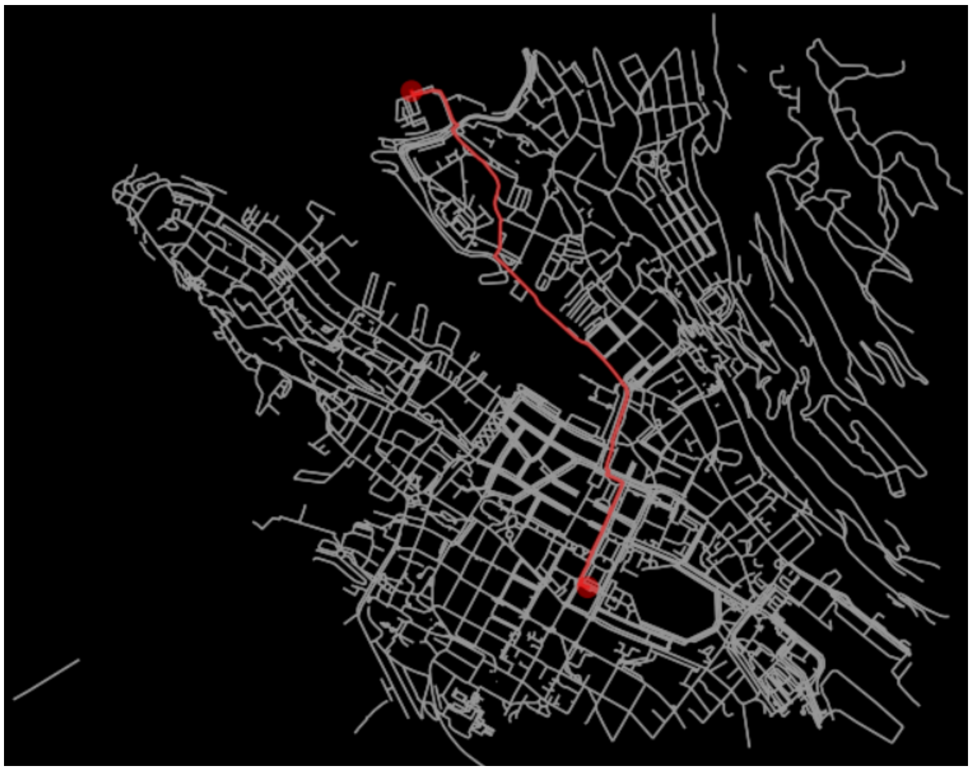


Figure 2.3: Route Example

To determine paths and associated travel time between the locations, OpenRouteService, an open-source routing site, is employed (OpenRouteService, n.d.). By utilizing each location's coordinates, retrieved through the Places API in Section 2.3, walking durations are extracted through the Matrix Service from the OpenRouteService API.

OpenStreetMap is a map of the world that is editable by the users and hence some distances and travel times might not be completely accurate. Google Maps is considered a more accurate map, but collecting these travel times from Google Maps is not for free. After comparing some of the travel times generated from OpenRouteService and Google Maps, the conclusion is that OpenRouteService is accurate enough for this purpose, especially since this is open-source.

2.5 Service times

To schedule routes tailored to the tourists' time available, the model requires information about the average service times for each of the locations. Suggested durations for each location are retrieved from Google and Tripadvisor, where the average between the two of them is determined as the service time. In instances with absent suggestions on specific locations, the average of the corresponding category is applied. Notably, for the locations within the *Food* category, minimal variations in service times are assumed in comparison to the other categories. To limit the extensive data gathering, for subcategory *Restaurants* the suggested duration for Bjerck Bergen is used as a representative for the entire selection of restaurants with an average service time equal to 120 minutes. The same simplification is done for the subcategory *Cafés*, where Godt Brød Fløyen is used as the representative café with an average service time equal to 92,5 minutes. The representative locations are chosen arbitrarily, but under the assumption that they represent diversity. This simplification is justified by the need for a trade-off between data accuracy and practicality in data gathering, ensuring that the collected data is both representative and feasible to obtain within the scope of this thesis.

Due to service times being determined on average values, there may be instances that arise where individuals spend more or less time at specific locations. Utilizing averages as a singular metric for all tourists fails to accurately capture the diverse range of behaviours across the tourists. The approach disregards variations in durations resulting in an oversimplified representation and neglecting extreme outliers from the norm. A possible alternative would be to utilize the distribution of service times and conduct a simulation.

This will be considered for further research in section 7.3.

2.6 Location capacities

In the process of data gathering for determining the capacity at the various locations, a manual approach is employed. This involves directly contacting employees at each location to gather relevant capacity information. However, to make this data gathering process more manageable, simplifications are implemented.

Capacity for locations namely Bjerck Bergen and Godt Brød Fløyen is applied as a standard measure for locations within categories *Restaurants* and *Cafés* respectively. This methodology follows the same logic as outlined in Section 2.5, where a similar approach is adopted for simplifying the data on service time. Further, for the rest of the locations with absent values, the average of the corresponding category is applied.

2.7 Pictures from Flickr

In this subsection, the number of photos taken at each location is retrieved from Flickr, which is a popular photo-sharing site (Flickr, n.d.). These measures are accessed by using a Flickr API. The number of photos at each location will contribute to indicating the popularity for determining the base score.

2.7.1 Defining keywords for locations

Distinct keywords are defined for each location, encompassing aliases and English notations to supplement the location name. For the locations in the category *Food* however, defining keywords is not performed. This is justified by the observation that restaurant names and keywords in photo searches on Flickr often result in the inclusion of irrelevant and non-specific images, thus leading to contamination. Consequently, the locations in the *Food* category are excluded from the photo extraction process presented in the following part.

2.7.2 Extracting the photos using Flickr API

Following the definition of keywords, the number of photos attributed to each location is obtained by counting the number of photos associated with each keyword. To maintain data integrity, duplicates are eliminated to ensure that each unique photo is counted only once. Additionally, this process is limited to only include publicly available photos captured within a 10-year timeframe, spanning from the 1st of January 2012 to the 31st of December 2022.

It is worth mentioning that while the Flickr API allows the user to extract photos based on coordinates, this approach is not suitable in this instance. There are several reasons for this, one being the high density of locations making it difficult to distinguish some locations and their respective photos clearly. Furthermore, the radius for extraction is decreased in an attempt to group tagged photos more effectively, resulting in the number of photos associated with different coordinates significantly decreasing.

3 Generating scores for the locations

The locations do not hold the same degree of popularity, and therefore a scoring system will be incorporated into the model. Each location will contribute a score to the model based on earlier visitors' reviews, ratings, the number of pictures posted, and user preferences. The scoring system will initially consist of a base score that considers the number of reviews, the ratings, and the number of pictures posted on Flickr for each location. After receiving user input, scores for individual preferences will be added to the base score. The implementation of the scoring system will be elaborated in this section.

3.1 Base score

The base score serves as an initial benchmark to rank locations relative to each other based on historical data and remains unaffected by tourist's preferences. Hence, this score is considered a fixed parameter per location in the model. To establish the base score for each location, retrieved photos and reviews will be utilized as an indicator of popularity, and ratings as an indicator of satisfaction.

3.1.1 Measure of popularity

This element is separated into two parts, *the relative number of photos* and *the relative number of reviews*. For the first part, data retrieved from the Flickr API is used. Highly popular locations are assumed in this context to be more frequently photographed and hence generate a greater number of posts on the platform. This measure is determined by dividing the number of photos taken at each location by the total number of photos collected for all locations. This is carried out to standardize the values for better comparison, and to eliminate biases. As mentioned in 2.7.1, locations falling under the category *Food* are excluded from the photo gathering process and therefore also left out from this calculation. However, this is not of significance as the locations within the *Food* category are not compared with the other locations. This will be further elaborated in Section 5.

For the second part, the number of reviews at each location on Google is extracted. The relative number is calculated by dividing the number of reviews for a location by the total number of reviews across all locations. This number is considered a measure of popularity based on the assumption that frequently visited, and hence popular, locations have a higher number of reviews on Google. For the relative number of reviews, locations from the *Food* category are included, contrary to the relative number of photos.

Assuming that tourists typically visit at most one location within the *Food* category during their stay and several locations within the other categories, it is considered inconvenient to score these locations relative to each other. For instance, it is inconvenient to weigh Akvariet relative to Godt Brød Fløyen. It is also acknowledged that locations within the *Food* category provide a different experience compared to those not in the category. Hence, these are separated when calculating the relative number of reviews.

3.1.2 Measure of satisfaction

The second part of the base score aims to capture the relative satisfaction level associated with each location. To assess the relative degree of satisfaction, already retrieved data on ratings from Google is used. This measure is utilized as its original and is a continuous number that varies from 1 to 5.

It is important to mention that for the measure of satisfaction, it is possible to carry out a textual analysis of the reviews for each location. The reviews can be both positive and negative. Hence, by analyzing positive and negative words in each of the reviews it is possible to determine whether the review would increase satisfaction or not. For the scope of this thesis, rating is used as the only measure of satisfaction. However, this is considered for further research in Section 7.3.

3.1.3 Aggregating the base score elements

The two measures; *measure of poplarity* and *measure of satisfaction*, are added together to constitute the base score. The relative number of photos and the relative number of reviews are multiplied by 100 to find the percentage measure. This is implemented to align both to the same scale and to make this value significant in the aggregated base score.

The locations in the *Food* category only have ratings and the relative number of reviews as elements of the aggregated base scores and are consequently receiving a slightly lower score compared to locations in other categories. As mentioned, this difference is not problematic, as locations within the *Food* category only will be weighted against others within the same category. The final calculations of the base scores are provided in Table D.1 - D.3 in the Appendix.

3.2 User preferences added to base score

To take into consideration the tourist groups' personal preferences, the users can add both category preferences and more specific location preferences. If the user has a category preference but no particular location preference within the category, an additional score will be assigned to the base score of all the locations within the category for the current user. Furthermore, if the user prefers specific locations, only those will be assigned an additional score.

For locations within the *Food* category, the user is first asked whether to include food or not. Then the user can prefer the subcategories; *Restaurants* or *Cafés*, providing all locations within the preferred subcategory an additional score. The user might want food but not have any specific preferences, and then all locations within the entire *Food* category will be assigned an additional score. In the same manner, as for locations not in the *Food* category, specific preferences imply an additional score assigned to only the specific ones.

4 Methodology

This section will serve as an introduction to the theoretical framework behind the proposed model. It will start by providing an introduction to basic mathematical programming and linear programming. Further, this section will explain some optimization problems that are relevant to the proposed model, as well as a possible method for solving such problems.

4.1 Mathematical programming

Mathematical programming (MP) is a widely used area in business analytics that aims to find the optimal utilization of finite resources among competing tasks within a set of constraints to achieve the objectives (Ragsdale, 2022, p.17)(Bradley et al., 1977, ch 1). MP is used to identify optimal values for a mathematical problem and is often referred to as optimization, which involves three elements: *decisions*, *constraints* and an *objective* that could either be maximized or minimized (Ragsdale, 2022, p.17-19).

4.1.1 Linear programming

When the elements of an optimization problem are formulated as linear functions, the MP problem can be categorized as a linear problem in the sense that it has straight functions and flat structures (Ragsdale, 2022, p.20-22). The general form of a Linear Programming (LP) model can be formulated as presented in the following.

Objective Function

$$\text{Maximize (or Minimize) } Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{4.1}$$

Subject to the Constraints

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \tag{4.2}$$

$$\vdots$$

$$a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kn}x_n \leq b_k \tag{4.3}$$

$$\vdots$$

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \tag{4.4}$$

Non-Negativity Constraints

$$x_1, x_2, \ldots, x_n \geq 0 \tag{4.5}$$

The objective of such an LP problem is to determine the value of the decision variables that maximize or minimize the objective function Z in function (4.1), where the coefficients $c_1, c_2, \ldots, c_n$ are the objective function coefficients representing the numeric constant associated with each decision variable $x_1, x_2, \ldots, x_n$. The variable $a_{11}, a_{12}, \ldots, a_{mn}$ represents the coefficients of the decision variables in the constraints with $b_1, b_2, \ldots, b_m$ as right-hand side values of the constraints presented throughout Equations (4.2) - (4.4). Equation (4.5) ensures non-negative decision variables (Ragsdale, 2022, p.22).

4.1.2 Integers and binary variables

Integer Linear Programming (ILP) refers to problems that are restricted to assuming only integer values (Ragsdale, 2022, p.243). Such problems may be employee scheduling problems, where a solution like 16.67 workers available at one shift is not a possible solution to the problem.

Contrary to integer variables implying any integer value, binary variables imply only two integer values, namely 0 and 1. The constraints are formulated in Equation (4.6) and Equation (4.7) respectively. When these are implemented in an LP problem, it becomes an ILP problem (Ragsdale, 2022, p.243).

Integer Constraint

$$x_1, x_2, \ldots, x_n \in \mathbb{Z} \tag{4.6}$$

Binary Constraint

$$x_{p+1}, x_{p+2}, \ldots, x_q \in \{0, 1\} \tag{4.7}$$

Variables that are not required to be integers are referred to as continuous variables, which is formulated in Equation (4.8) (Ragsdale, 2022, p.243). Problems that include both integer and continuous constraints on variables are referred to as Mixed Integer Linear Programming (MILP) problems (Bradley et al., 1977, p.272).

Continuous Constraint

$$x_{q+1}, x_{q+2}, \ldots, x_n \in \mathbb{R} \tag{4.8}$$

4.1.3 Soft and hard constraints

We distinguish between two classifications of constraints: hard and soft. By default, optimization techniques assume constraints that cannot be violated, referred to as hard constraints. Hard constraints are not always applicable, as they may be too restrictive in some problems. In such instances, soft constraints may be implemented. These types of constraints represent something desirable to achieve, but that might not be realized (Ragsdale, 2022, p.323). Violation of a soft constraint is in other words allowed, but typically violation results in a well-defined penalty in the objective.

Objective Function

$$\text{Maximize } Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n - p \cdot s \tag{4.9}$$

Subject to the Hard Constraints:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \leq b_1 \tag{4.10}$$
$$\vdots$$
$$a_{k1}x_1 + a_{k2}x_2 + \cdots + a_{kn}x_n \leq b_k \tag{4.11}$$
$$\vdots$$
$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \leq b_m \tag{4.12}$$

Subject to the Soft Constraints:

$$a_{(m+1)1}x_1 + a_{(m+1)2}x_2 + \cdots + a_{(m+1)n}x_n + s_1 \leq b_{m+1} \tag{4.13}$$
$$\vdots$$
$$a_{(m+k)1}x_1 + a_{(m+k)2}x_2 + \cdots + a_{(m+k)n}x_n + s_k \leq b_{m+k} \tag{4.14}$$
$$\vdots$$
$$a_{(2m)1}x_1 + a_{(2m)2}x_2 + \cdots + a_{(2m)n}x_n + s_m \leq b_{2m} \tag{4.15}$$

Non-Negativity and Slack Variable Constraints:

$$x_1, x_2, \ldots, x_n, s \geq 0 \tag{4.16}$$

The objective function maximizes Z in Equation (4.9), where the penalty term $p \times s$ is subtracted from the initial objective. Hence, a relaxation is added to the soft constraints by introducing a slack variable as a penalty on the objective if violated.

4.1.4 Infeasible solutions

For instances where constraints in an LP problem are not possible to satisfy simultaneously, the problem is considered to have no feasible solutions and hence is infeasible (Ragsdale, 2022, p.38). Infeasibility may be caused by too strict constraints. Therefore, a way to avoid infeasibility is by introducing soft constraints that increase the feasible region for the problem.

4.2 Route and scheduling optimization

MP techniques have a wide range of applications, of them being routing and logistics problems (Ragsdale, 2022, p.18). This section aims to describe three types of models that previously have been used to solve such problems.

4.2.1 Travelling salesman problem

The Travelling Salesman problem (TSP) is a route optimization problem and one of the most famously studied problems in the field of optimization (Ragsdale, 2022, p.413). A salesman seeks to find the shortest path possible between multiple cities n before returning to the starting point. The problem is simple to understand and state, but becomes conceptually and numerically difficult to solve as the number of cities n increases. Generally, for n cities in a TSP, there are $(n-1)!$ possible routes the salesman may use. For just 9 cities, the number of possible routes becomes 40,320 and 21 trillion possible routes for 17 cities. TSPs are therefore difficult to solve with traditional linear methods within a constrained time frame, and heuristics are often implemented to solve them (Ragsdale, 2022, p.413).

4.2.2 Vehicle routing problem

The Vehicle Routing Problem (VRP) is a route optimization problem of determining optimal delivery routes from a starting point to a set of geographically scattered points that may represent cities or customers for a set of vehicles (Laporte & Nobert, 2008). VRP is considered a generalization of TSP that consists of ascertaining the shortest route through each of n-points once. The main difference between VRP and TSP is that VRP introduces another layer of complexity to the problem by introducing more elements to consider.

4.2.3 Open shop scheduling

The *Open Shop Scheduling Problem* (OSSP) refers to the scheduling optimization problem of assigning a set of jobs to a set of machines with various constraints in place (Kubiak, 2022, p.1-3). Each job requires operations on each of the machines, each of the machines can only process one job at a time, and every operation is assigned to a time interval in a schedule (Woeginger, 2018, 4:2)(Kubiak, 2022, p. 1). This is to ensure that no job is being processed on two machines simultaneously. Initially, the operations have no fixed order. However, if the order is specified, the problem is referred to as a *Job Shop Scheduling Problem* (JSSP) where the different jobs have different orderings of the operations. JSSP is therefore a modification of the initial OSSP (Woeginger, 2018, 4:2).

The JSSP has similarities to TSP in the means that the formal definition is the same (Tsirlin & Balunov, 2022).In TSP, a salesman travels through an assigned set of cities, while in JSSP a job is assigned to a set of machines and hence can be considered "traveling" between machines.

4.2.4 Heuristic methods

TSP, VRP and JSSP are known as NP-hard problems as the number of cities increases for TSP and VRP, and jobs, machines and operations per job for JSSP (Jones, 2023) (Brucker et al., 2007). This is due to the vast amount of data required to be processed in such problems, where current computers frequently encounter a state of "powerlessness" when tackling numerous practical computing problems (Li et al., 2020). Therefore, computers are often incapable of resolving these issues within a reasonable timeframe, and this categorization implies that the problem requires a substantial amount of computational time. Furthermore, some instances are non-computational if the size of the instance is too large. To cope with such NP-hard problems, one can implement certain algorithms, such as heuristics, to solve them (Li et al., 2020). Heuristics, while not guaranteeing optimal solutions, are considered a pragmatic approach by yielding satisfactory solutions within a reasonable time frame, thereby balancing the trade-off between solution optimality and computational efficiency (Du et al., 2021).

5 Proposed model

In this section, the proposed model will be presented. A model solving the problem of allocating all groups of tourists to individual routes is considered too comprehensive for the scope of this thesis. As mentioned in Section 4, heuristic methods are an alternative that may be implemented to resolve this. However, based on the time limit for this master's thesis, we do not find it convenient to spend time developing a heuristic method for solving this problem. Therefore, a simplification of the model is developed.

A selection of predefined routes is generated as input to the model. Instead of allocating each tourist group to an individual route with a selection of locations, the simplified model allocates tourist groups to predefined routes. All input to the model, including the user input, must be provided before the model is solved. Hence, the model is solved under the assumption that all user input is provided within a specific time.

In the following sections, the full-scale optimization model will first be discussed briefly before the proposed simplified model is elaborated. For the latter, the generation of predefined routes will be explained and the simplified optimization model will be presented in detail.

5.1 Full-scale optimization model

An optimal model would be akin to a TSP variation, where the model generates optimized routes with minimized walking distances to the tourists based on preferred locations selected beforehand. As previously mentioned, VRP is an extension of TSP by incorporating additional complexities. In the scenario of this master's thesis, such an extension involves considering factors such as time, opening hours, multiple destinations and multiple tourists.

The ideal model would aim to allocate all 8,000 possible tourists in groups arriving during a day, simultaneously to individual routes maximizing scores and at the same time preventing congestion. This problem can be compared to a version of VRP with Time Windows, which is an extensive NP-hard problem. Simplifications are therefore implemented to achieve an optimal solution to the problem within a feasible computational time.

5.2 Simplified optimization model

An approach to address the complexity of assigning routes without implementing heuristic algorithms involves predefining collections of randomly chosen locations under certain restrictions. After the collections are generated, a TSP is solved on each collection to create a route, optimizing the order of the locations, such that the walking duration is minimized. The tourists will then be assigned to the predefined routes instead of individual locations that constitute a route.

In the following, the generating of predefined routes will initially be elaborated. Furthermore, the mathematical formulation of the simplified optimization model will be presented and explained in detail.

5.2.1 Generating predefined routes

The full-scale optimization model requires the consideration of several constraints to effectively generate the suggested routes. These constraints must be addressed when creating the predefined routes. Initially, the routes need to start and end at the port in which the tourists arrived in Bergen to ensure a full itinerary. Further, each location can only be visited once within a route. Mathematically it is necessary with a constraint to ensure each arriving location equals the next leaving location to ensure a continuous route. To prevent routes involving an inconveniently long walking duration between two linked locations, it is necessary to include a constraint for the maximum walking duration between two linked locations. Additionally, it is necessary to monitor the cumulative time spent from the start and throughout the route and the total time spent on the route. A selection of routes must include a location within the *Food* category for those tourists who prefer food to be included in their route. To prevent tourists from getting hungry during a route, it can be argued that it is most convenient to add a food stop somewhere midway through the route. However, to prevent excessive walking the food location must be strategically placed where it results in the least additional walking instead.

5.2.1.1 Randomized collections of locations

The collections of locations are generated randomly but within the guidelines mentioned above. The collections are forced to start and end at the same cruise port, either *Jekteviksterminalen* or *Bontelabo cruise terminal*. Bergen Havn has five cruise ports; Jekteviken, Tollboden, Festningskaien, Skolten N & S and Bontelabo. Since Tollboden, Festningskaien, Skolten N & S and Bontelabo are closely located, these cruise terminals are considered as Bontelabo as a simplification. Several collections are generated both from Bontelabo cruise terminal and Jekteviksterminalen as input to the model.

Each collection can only contain a location once. This is considered by selecting randomly one location at a time from the list of locations, and then removing the selected location from the list before selecting the next location. When a location is randomly selected from the list, the location must satisfy the mentioned guidelines before it is added to the collection. If the successor location is reachable within 25 minutes of walking from the predecessor location, as determined to be the boundary in Section 2.1, then the first condition is met. In addition, if the service time of the successor location plus the walking duration back to the port is within the time limit, then the second condition is met. When both conditions are met, the successor location is incorporated into the collection. Despite that the walking duration between the locations is restricted, the walking duration back to the port from the last location is an exception. Hence, this will be accounted for in the model presented in Section 5.2.2. To ensure there are routes for the various needs of the tourists, several collections within 10 hours are generated. This is based on the information in Section 2.1.

Locations within the *Food* category are excluded from the list of locations to randomly select from. Food locations require another approach compared to others, as a route including a dominant number of food locations will be inconvenient and reduce the overall experience. Since the food locations constitute more than half of the total number of locations, many routes would consist solely of locations within the *Food* category. Only one food location is added for a selection of the collections, and whether the location belongs to the subcategory; *Cafés* or *Restaurants*, is randomly chosen.

Given that the collections are generated randomly, they are not specifically implemented to maximize the satisfaction score. Generating collections that maximize the score would result in routes only consisting of the most popular locations based on previous visitors. In the proposed model, which accounts for individual preferences, it is essential to include a diverse array of locations within the routes to address the interests of the widest possible audience.

5.2.1.2 TSP optimization to generate routes

After the collections of locations are generated, the order of the locations within the collection is optimized to compose a route. This has been achieved by utilizing a TSP optimization model. The model provides the visiting order for the locations which implies the least walking duration in total, and hence reduces the cross-walking. A TSP model is solved for each of the randomly generated collections. In the following, the mathematical formulation of the TSP model is presented.

Sets and parameters

Initially, the sets and parameters are presented in Table 5.1 and Table 5.2 respectively.

Table 5.1: Sets for TSP

Symbol	Description
V	Set of locations in the route

Table 5.2: Parameters for TSP

Symbol	Description
$D_{i,j}$	Travel time between location $i \in V$ to $j \in V$
port	Fixed parameter representing the starting and ending location of the route

Set V contains all the locations in the route. The travel duration, given in minutes, between location $i \in V$ and $j \in V$ is mapped by the parameter $D_{i,j}$. Furthermore, there is a fixed parameter, *port*, which equals the port the route starts and ends at.

Variables

The variables used in the model are presented in Table 5.3.

Table 5.3: Variables

Symbol	Description
$x_{i,j}$	Binary decision variable which is 1 if the path from i to j is included in the tour, 0 otherwise
u_i	Auxillary variable representing the position of location i in the route

$x_{i,j}$ is a binary decision variable determining whether the path between location $i \in V$ to $j \in V$ is present in the route or not. In other words, whether location j comes after location i in the route. This variable constitutes the optimized order of the route. Further, u_i is a continuous variable representing the position of location i in the route. This is an auxiliary variable bounded between 0 and $|V| - 1$, with $|V|$ being the cardinality of set V, representing the number of locations in the route.

Objective function

The objective function aims to determine the order of the locations in the route that minimize the total travel duration.

$$\text{Minimize} \quad Z = \sum_{(i,j) \in V, i \neq j} D_{i,j} \cdot x_{i,j}$$

As mentioned in Section 5.2.1.1, there is no restriction on the walking duration between the last location in the collection before the port and the port. When the order of the collections is modified by executing a TSP model, this may lead to a long walking duration between other consecutive locations in the route. However, as previously mentioned, this will be accounted for in the optimization model presented in Section 5.2.2.

Constraints

Initially, the model is subject to a hard constraint that ensures each location is visited exactly once. This is formulated in Equation 5.1 and 5.2. The port is excluded since the route starts and ends at the same port.

$$\sum_{j \in V, j \neq i} x_{i,j} = 1 \quad \forall \quad j \in V, j \neq port \tag{5.1}$$

$$\sum_{i \in V, i \neq j} x_{i,j} = 1 \quad \forall \quad i \in V, i \neq port \tag{5.2}$$

Further, Equation (5.3) and (5.4) represent a hard constraint ensuring the route starts and ends at the port.

$$\sum_{j \in V, j \neq \text{port}} x_{\text{port},j} = 1 \tag{5.3}$$

$$\sum_{i \in V, i \neq \text{port}} x_{i,\text{port}} = 1 \tag{5.4}$$

To prevent sub-tours between locations, the constraint presented in Equation (5.5) is implemented. Initially, in the TSP we seek a solution of a continuous route where every path is traversed exactly once, except for the ports. However, without specific constraints, solutions consisting of multiple sub-tours may occur. A sub-tour refers to a smaller tour where a selection of the initial set of locations are visited (Giovanni & Summa, n.d.). To address this, a sub-tour elimination constraint is implemented. This constraint applies for all paths between location i and location j in the route, except the ports. If variable $x_{i,j}$ equals 1, indicating the path from i to j is present in the route, then the position of j must be greater than i. This aims to prohibit sub-tours and ensures the route is a single connected tour.

$$u_i - u_j + |V| \cdot x_{i,j} \leq |V| - 1 \quad \forall i, j \in V, i \neq j, i \neq port, j \neq port \tag{5.5}$$

After executing the TSP model on each collection to determine the optimal order of the locations, the set of all composed routes is used as input in the proposed optimization model for the tourist groups to be assigned to.

5.2.2 Formulation of the proposed Optimization Model

The proposed model is a MILP problem that aims to assign tourist groups to the predefined routes within available timeslots, maximizing the total score while reducing congestion.

5.2.2.1 Sets and parameters

Initially, the sets and parameters are elaborated and presented in Table 5.4 and Table 5.6 respectively. Furthermore, Table 5.5 and 5.7 presents auxiliary parameters utilized in the model, but not visibly apparent in the formal mathematical formulation.

Table 5.4: Sets

Symbol	Description
L	Set of locations included in the model
R	Set of predefined routes consisting of locations from L
P	Set of tourist groups
D	Set of days in a week
T	Set of timeslots (range by minutes)
C	Set of cumulative times, integers from 0 to 600

The model is constructed from five sets. Set L represents all the possible locations included in the model. R is a set of predefined routes, consisting of the route names and further, each name refers to a predefined route as a list of locations from the set of locations. Set P is a set of tourist groups to allocate in the model. A tourist group refers to a group of tourists walking together and can range from one tourist and upwards. Furthermore, the model includes a set D of days in a week. Set C contains integers from 0 to 600, representing cumulative minutes from starting time 0. Lastly, set T is a set of timeslots with each minute raging from 08:00 to 22:00.

Table 5.5: Auxiliary parameters for the set of routes

Symbol	Description
$locCumulatives_r$	List of cumulative times for the locations constituting route r

As mentioned, the model utilizes a parameter that is not visibly apparent in the formal mathematical formulation. The parameter $locCumulatives_r$ consists of the cumulative times at each location constituting route r.

Table 5.6: Parameters

Symbol	Description
$numP_p$	Number of tourists in tourist group $p \in P$
$fromB_p$	Binary parameter which is 1 if tourist group $p \in P$ starts at Bontelabo cruise terminal, 0 otherwise
ST_p	Starting time for tourist group $p \in P$
G_p	Time available for tourist group $p \in P$
A_r	Duration of route $r \in R$
Cap_l	Capacity at location $l \in L$
$travel_{i,j}$	The travel duration between location $i \in L$ and location $j \in L$
$cumul_{r,l,c}$	Binary parameter which is 1 if route $r \in R$ includes location l ∈ L at cumulative time $c \in C$, 0 otherwise
$openT_{l,d,t}$	Binary parameter which is 1 if location $l \in L$ is open at day $d \in D$ at time $t \in T$, 0 otherwise
$portB_r$	Binary parameter which is 1 if route $r \in R$ is from Bontelabo cruise terminal, 0 otherwise
$today$	Fixed parameter representing the day of the week
$capPenalty$	Fixed parameter providing the penalty for exceeding the capacity at a location
$walkPenalty$	Fixed parameter providing the penalty for exceeding the maximum walking duration between locations
$walkMax$	Fixed parameter with the maximum walking duration in minutes between two consecutive locations
$RouteScore_{p,r}$	The total score tourist group $p \in P$ receives from route $r \in R$

Several parameters constitute the framework of the model. The first parameters presented in Table 5.6 are determined by input from the users. Parameter $numP_p$ contains integer values for the number of persons in tourist group $p \in P$. ST_p holds the starting time for tourist group $p \in P$ and G_p holds the time available for tourist group $p \in P$, presented in minutes. Further, there is a binary parameter determined by user input. Parameter $fromB_p$ indicates whether tourist group $p \in P$ arrived in Bergen with a cruise ship docking at Bontelabo cruise terminal or not. If this variable equals 1, tourist group

p arrived at Bontelabo cruise terminal, and if it equals 0, tourist group p arrived at Jekteviksterminalen.

In addition to parameters determined by user input, the model consists of parameters determined by the data collected in Section 2. A_r is a parameter that represents the duration of route $r \in R$ which is based on the walking durations and the service times. Cap_l represents the visitor capacity at each location $l \in L$. $travel_{i,j}$ is a parameter representing the walking duration between location $i \in L$ and location $j \in L$. Further, the model contains a binary parameter $cumul_{r,l,c}$ which indicates whether route $r \in R$ contains location $l \in L$ at the cumulative time (minutes) $c \in C$ from the start of the route equal to zero. This parameter equals 1 if route r contains location l at cumulative time c, and 0 otherwise. To keep track of the opening hours at each location, there is a binary parameter $openT_{l,d,t}$ which is 1 if the location is open at day $d \in D$ and at time $t \in T$, and 0 otherwise. To keep track of which port the routes are based on, a binary parameter $portB_r$ indicates whether route $r \in R$ is based on Bontelabo cruise terminal or not. If route r starts at Bontelabo cruise terminal, $portB_p$ equals 1, and otherwise, if route r is based on Jekteviksterminalen, it equals 0.

There are four fixed parameters in the model. $capPenalty$ is a parameter representing the penalty added to the objective if the capacity constraint is exceeded. Parameter $walkPenalty$ represents the penalty for being assigned to a route r exceeding the maximum walking between two locations. Furthermore, $today$ is a fixed parameter holding the day of today, hence the day for which the tourist groups are assigned to routes. Lastly, $walkMax$ represents the boundary for walking between two consecutive locations, which is determined to be equal to 25 minutes as denoted in Section 2.1.

Parameter $RouteScore_{p,r}$ represents the total score, as the measure of satisfaction, tourist group $p \in P$ obtain from route $r \in R$. The value of this parameter is dependent on whether the route contains a location the tourist group prefers, whether the route is based on the port in which the tourist group arrived, and whether the route satisfies food preferences. Hence, this parameter is composed of the auxiliary parameters presented in table 5.7. To reduce the number of parameters in the model, these parameters are assembled into the parameter $RouteScore_{p,r}$.

Table 5.7: Auxiliary parameters to construct $routeScore_{pr}$

Symbol	Description
S_l	Base score for location $l \in L$
$prefScore$	Fixed parameter representing the preference score which is added if a location is preferred
$isPref_{p,l}$	Binary parameter which is 1 if tourist group $p \in P$ prefers location $l \in L$ is preferred by tourist $p \in P$, 0 otherwise
$wantFood_p$	Binary parameter which is 1 if tourist group $p \in P$ prefers a food stop in the route, 0 otherwise
$inRoute_{r,l}$	Binary parameter which is 1 if route $r \in R$ contains location $l \in L$, 0 otherwise
$haveFood_r$	Binary parameter which is 1 if route $r \in R$ includes a location within either the restaurant or the café category, 0 otherwise

S_l is a parameter representing the base score each location $l \in L$ provides, and $prefScore$ is a fixed parameter representing the score to be added to a location if it is preferred by a tourist group. $isPref_{p,l}$ is a binary parameter based on user input, indicating whether tourist group $p \in P$ prefers location $l \in L$. This parameter is equal to 1 if tourist group p prefers location l, and 0 otherwise. $wantFood_p$ is another binary parameter based on user input, which is equal to 1 if tourist group $p \in P$ prefers a route including a location within the *Food* category, and 0 otherwise. To determine whether route $r \in R$ contains location $l \in L$, there is a binary parameter $inRoute_{r,l}$.

To determine whether route $r \in R$ contains a food location, regardless of whether it is a restaurant or a café, there is a binary parameter $haveFood_r$. This is implemented to ensure the $RouteScore_{p,r}$ for a route r without a food location equals 0 for a tourist group p wanting food, and the same if a route r contains a food location for a tourist group p not wanting food. To the extent it is possible, the model aims to assign tourist groups who prefer food to a route including food, and conversely for those who do not prefer food. If this is not possible, it is still preferable to assign them to a route such that the model does not give an infeasible solution. This is the reason why there is no hard constraint forcing $wantFood_p$ to be equal to $haveFood_r$ for the route r tourist group p is assigned to. Instead, all routes r without food are initially given a $RouteScore_{p,r}$ equal to 0 for all tourist groups p preferring food, and vice versa. Nevertheless, to ensure that by the selection of routes not satisfying the food preference, the routes including other preferred locations are weighted higher than those not, a score equal to 1 is added to $RouteScore_{p,r}$ per location preferred in the route.

In the same manner as routes without a location within the *Food* category provide a low $RouteScore_{p,r}$ for tourist groups wanting food and opposite, it is beneficial that routes based on a different port than the one the tourist group arrive at provide a score equal to 0. Still, this is not implemented in the $RouteScore_{p,r}$ because it is ensured by implementing a hard constraint, presented in Section 5.2.2.3. This constraint forces tourist group p to be assigned to a route r based on the port they arrived at. Equation 5.6 illustrates the composition of the $RouteScore_{p,r}$ parameter.

$$\begin{aligned}\text{RouteScore}_{p,r} = &\sum_{l \in L} (S_l + \text{prefScore} \times (\text{isPref}_{p,l} \times \text{inRoute}_{rl})) \\ &\times (1 - |\text{wantFood}_p - \text{haveFood}_r|) + \sum_{l \in L} isPref_{p,l}\end{aligned} \tag{5.6}$$

5.2.2.2 Variables

The decision variables are presented in Table 5.8. These are the values the model will seek to determine to maximize the objective.

Table 5.8: Variables

Symbol	Description
x_{p,r,ST_p}	Binary variable which is 1 if tourist group $p \in P$ is assigned to route $r \in R$ at starting time ST_p for tourist group p, 0 otherwise
$y_{p,r,l,t}$	Binary variable which is 1 if tourist group $p \in P$ is at location l $\in$ L belonging to route $r \in R$ at time $t \in T$, 0 otherwise
$z_{l,t}$	Integer variable larger than 0 which provides slack for the capacity at location $l \in L$ at time $t \in T$
q_p	Continuous variable larger than 0 which provides slack for the walking duration for tourist group $p \in P$

Decision variable x_{p,r,ST_p} is a binary variable indicating whether tourist group $p \in P$ is assigned to route $r \in R$ at the tourist group's starting time ST_p or not. If x_{p,r,ST_p}

equals 1, it indicates that tourist group p is assigned to route r at starting time ST_p, and conversely if x_{p,r,ST_p} equals 0. Decision variable $y_{p,r,l,t}$ is another binary variable. This variable indicates whether tourist group $p \in P$ is assigned to route $r \in R$ and is present at location $l \in L$ at time $t \in T$. $y_{p,r,l,t}$ equals 1 if person p is assigned to route r and is present at location l at time t, and 0 otherwise.

The model contains an integer variable $z_{l,t}$ larger than 0. This variable serves as a slack variable for the capacity at location $l \in L$ at time $t \in T$, allowing more than the capacity to be allocated to location l at time t in exchange for a penalty, *capPenalty*. Since there is no possibility of allowing half a person more at a location, this variable is required to be an integer number.

Further, the model contains a continuous slack variable q_p larger than 0, which allows tourist group p to be assigned to a route r that exceeds the maximum walking duration between two consecutive locations in exchange for a penalty, *walkPenalty*. q_p represents the minutes exceeding $walkMax$ a tourist group must walk between consecutive locations in the assigned route r.

5.2.2.3 Objective function and constraints

In the following, the objective function and the restricting constraints are elaborated.

Objective function

The objective function seeks to assign tourist groups to routes that maximize the total score achieved and at the same time prevent queues and long walking durations between consecutive locations. It does so by considering all possible pairings between tourist groups and routes and then assigning the tourist groups to the routes that provide the highest score. If too many tourists are assigned to the same location at the same time, exceeding the capacity, it results in a penalty to the total score as a result of queues. Furthermore, if the tourist groups are assigned to routes where the walking duration between two consecutive locations exceeds the maximum walking, a penalty is added to the total score.

$$\text{Maximize} \sum_{p\in P}\sum_{r\in R} \text{RouteScore}_{p,r} \times x_{p,r,\text{ST}_p} - \sum_{p\in P} q_p \times \text{walkPenalty} - \sum_{l\in L}\sum_{t\in T} z_{l,t} \times \text{capPenalty} \tag{5.7}$$

The objective is presented in equation (5.7). The objective sums $RouteScore_{p,r}$ multiplied by whether tourist group p is assigned to route r at starting time ST_p, x_{p,r,ST_p}, for all tourist groups $p \in P$ and routes $r \in R$. To allow for queues, but prevent them, the objective function subtracts the sum of all slack values for capacity, $z_{l,t}$, multiplied by the penalty for exceeding the capacity, $capPenalty$, for all locations $l \in L$ and all timeslots $t \in T$. Further, the objective subtracts the sum of all slack values for exceeding walking duration, q_p, multiplied by the penalty for exceeding the maximum walking, $walkPenalty$.

Route constraint

The model is subject to a hard constraint that ensures each tourist group p is assigned to one and only one route r. The constraint is structured to ensure that for every tourist group p within the set P, the sum of x_{p,r,ST_p} across all routes r within the set R must be equal to one. The mathematical formulation of this constraint is presented in equation (5.8).

$$\sum_{r\in R} x_{p,r,\text{ST}_p} = 1 \quad \forall p \in P \tag{5.8}$$

This constraint forces tourist group $p \in P$ to be assigned to a route $r \in R$ starting at time ST_p, which is the preference for starting time provided by tourist group p. This restricts the model from suggesting a route starting for instance 10 minutes later, despite this potentially providing a higher achieved score due to shorter queues and opening hours at preferred locations. This will be further discussed in Section 7.1.

Port constraint

Further, the objective function is subject to another hard constraint ensuring tourist group $p \in P$ must be assigned to a route $r \in R$ from the port they arrived at. This is presented in Equation (5.9). If the user input in $fromB_p$ is equal to one, indicating tourist group $p \in P$ arrived at Bontelabo cruise terminal, then the route $r \in R$ the tourist group is assigned to must be a route based on Bontelabo cruise terminal. Hence, $portB_r$ must equal $fromB_p$.

$$\sum_{r \in R} x_{p,r,\mathrm{ST}_p} \times \mathrm{portB}_r = \mathrm{fromB}_p \quad \forall p \in P \tag{5.9}$$

Time available constraint

The objective function is also subject to a hard constraint that ensures the time spent for each tourist group $p \in P$ does not exceed the time they have available. It is presented mathematically in Equation (5.10). The duration A_r on the assigned route $r \in R$ for tourist group $p \in P$ must be less than or equal to the time available, G_p for tourist group $p \in P$.

$$\sum_{r \in R} A_r \times x_{p,r,\mathrm{ST}_p} \leq G_p \quad \forall p \in P \tag{5.10}$$

Capacity constraint

A soft constraint for the capacity is added to the model to prevent congestion, illustrated in Equation (5.11). This constraint ensures that the sum of all tourists visiting location $l \in L$ at time $t \in T$ is within the visitor capacity at location l, or that a penalty is added to the score if the capacity is exceeded. The slack variable $z_{l,t}$ is added to the capacity to make it a soft constraint. People may wait in line if the capacity is reached, but the satisfaction will decline. For the mathematical formulation an indicator function $I(l \in r)$ is defined to determine whether location $l \in L$ is present in route $r \in R$. The indicator function will result in 1 if l is present in r and 0 otherwise.

$$\sum_{p \in P} \sum_{r \in R} y_{p,r,l,t} \times \mathrm{numP}_p \times I(l \in r) \leq \mathrm{openT}_{l,\mathrm{today},t} \times (\mathrm{Cap}_l + z_{l,t}) \quad \forall l \in L, \forall t \in T \tag{5.11}$$

By introducing a capacity constraint, the model will seek to allocate tourist groups to locations where the capacity is not exceeded. The slack variable permits the exceeding of capacities, but the subsequent penalty serves as a disincentive to such exceedings.

However, this requires including a significant number of predefined routes such that there are viable alternatives if the route providing the highest score leads to exceeding capacity at any location. The effect of the capacity penalty and the number of routes will be further elaborated in Section 6.

Walking duration constraint

To prevent tourist groups from being assigned to a route that contains walking durations between two consecutive locations exceeding *walkMax*, a soft constraint is added to the model. This constraint ensures that if tourist group p is assigned to route r, the travel time between all consecutive locations within route r must be within *walkMax*, or the slack variable is activated and results in a penalty. This is presented mathematically in Equation (5.12). The notation n_r represents the number of locations in route r, such that $travel_{i,i+1}$ represents the walking duration between two consecutive locations in route r.

$$x_{p,r,\mathrm{ST}_p} \times \mathrm{travel}_{i,i+1} \leq \mathrm{walkMax} + q_p \quad \forall p \in P, \forall r \in R, \forall i \in \{0, \ldots, n_r - 2\} \tag{5.12}$$

Logical relationship constraint

To ensure the correct relationship between x_{p,r,ST_p} and $y_{p,r,l,t}$, the objective is subject to two logical constraints. For the mathematical formulation of the logical relationship, there is an indicator function, $B(r, l, t, \mathrm{ST}_p)$ for whether time t falls within the time interval between the start time of location $l \in L$ and the end time of the same location l. The starting time of location l is calculated by the cumulative time for location l subtracting the travel time from the predecessor location. Furthermore, there is a placeholder for the time that has passed at in route $r \in R$ at location $l \in L$ for time $t \in T$ from the start time, ST_p for tourist group $p \in P$, denoted as *calcTime(r,l,t,*ST_p*)*, and presented in Equation 5.13.

$$calcTime = t - \mathrm{ST}_p \tag{5.13}$$

The indicator function $B(r, l, t, \mathrm{ST}_p)$ and the placeholder *calcTime(r,l,t,*ST_p*)* is then utilized in the formulation of the constraints, presented in Equation (5.14) and (5.15). These constraints ensure that when x_{p,r,ST_p} is equal to 1, then $y_{p,r,l,t}$ must be equal to 1 for the times t person p is at location l in route r. The indicator function ensures that these constraints are applicable when t is within the starting time and ending time of location l in route r for tourist group p.

$$y_{p,r,l,t} \leq x_{p,r,\mathrm{ST}_p} \times \mathrm{cumul}_{r,l,\mathrm{calcTime}(r,l,\mathrm{ST}_p)} \quad \forall p \in P, \forall r \in R, \forall t \in T, \forall l \in r, B(r,l,t,\mathrm{ST}_p) = 1 \tag{5.14}$$

$$y_{p,r,l,t} \geq x_{p,r,\mathrm{ST}_p} \times \mathrm{cumul}_{r,l,\mathrm{calcTime}(r,l,\mathrm{ST}_p)} - (1 - x_{p,r,\mathrm{ST}_p}) \quad \forall p \in P, \forall r \in R, \forall t \in T, \forall l \in r, B(r,l,t,\mathrm{ST}_p) = 1 \tag{5.15}$$

5.2.2.4 Problem classification

With the established elements, the optimization problem is interpreted as a scheduling problem. The model of this master's thesis may be considered to have similarities to JSSP if following the analogy of conceptualizing each of the tourist groups as *jobs* and locations as *machines*. Since the order of locations is fixed in already predefined routes, it can be interpreted as *operations*. Unlike the JSSP, the locations in our model can hold several tourists at the same time, while the machines in JSSP can only hold one job at a time. However, as in JSSP, each location can hold tourists at different timeslots. In a JSSP, a processing time at the machines is employed for the jobs, while in our optimization problem, the equivalent is the service times at the different locations.

5.3 Implementation

The mathematical model is implemented in Python programming language using the PuLP framework. PuLP serves as an LP modelling tool within the Python environment, facilitating the definition of sets, parameters, variables, objectives, and constraints in a mathematical format suitable for optimization problems (PyPI, n.d.). PuLP itself does not find the optimization solution. Instead, it formulates the LP problem, and the problem is further solved by an external solver. For this master's thesis, both the TSP for the predefined routes in Section 5.2.1.2 and the proposed optimization model in Section 5.2.2 are solved using the COIN-OR Branch and Cut (CBC) solver, an open-source mixed-integer linear programming solver (Forrest, n.d.).

The generating of predefined routes and the implemention of the model in Python is presented in Appendix Section E and F respectively.

6 Analysis

In this section, an analysis of the proposed model will be carried out. The objective is to examine the model's performance and to understand the impact of a selection of parameters' values. The model's functionality and responsiveness to the inputs will be evaluated by exploring five of the parameters; the number of predefined routes, the number of tourist groups, the size of the penalty for exceeding the capacity (*capPenalty*), the size of the penalty for exceeding the maximum walking duration between two consecutive locations (*walkPenalty*), and the size of the score added for individual preferences ($prefScore$).

To conduct such an analysis, an artificial test population is constructed. This is presented in Table 6.1. The test population seeks to mimic an arbitrary input the tourist groups provide through the application interface. A small number of groups equal to five tourist groups are chosen to be able to observe the effects more easily. It is important to mention that in real life, the number of tourist groups will be significantly larger than the test population in this analysis, and the same is true for the number of predefined routes. To put this in perspective, as mentioned in Section 1.1, the maximum number of cruise tourists visiting Bergen during a day is equal to 8,000 (Møllerup, 2023).

Table 6.1: Test population

Tourist Group	Number of People	Port	Start Time	Time Available
P0	5	Bontelabo	10:00	6h
P1	4	Jekteviken	11:00	5h
P2	2	Jekteviken	12:00	4h
P3	1	Bontelabo	10:00	7h
P4	8	Bontelabo	11:00	3h

Tourist Group	Food stop	Food category preference	Specific food preference
P0	Yes	Restaurants	Bjerck Bergen
P1	No	—	—
P2	Yes	Cafés	Blom Bergen
P3	Yes	Restaurants	—
P4	No	—	—

Tourist Group	Category preference	Specific location preference
P0	Museums & Galleries	Kunsthall 3,14
P1	Historical sites	Bryggen, Håkonshallen
P2	Museums & Galleries	KODE Lysverket
P3	Activities, Churches	Floibanen, Ulriken, Nykirken
P4	Family friendly	Floibanen

In addition to the test population, the weekday used as the base in the analysis is set to be Thursday throughout the analysis. The model initially retrieves the current day it is executed, but for the analysis, the day is manually set to Thursday as it is assumed that the majority of locations are open on Thursdays. Additionally, the analysis executed by Amland Reiselivsutvikling in 2021, shows that Thursday is the most common arrival day for the cruise ships both in 2019 and 2021 (Amland, 2021, p. 6). A set of 231 routes are established as the foundation for the analysis. Furthermore, in the analytical framework, a baseline for the other parameters are establised; *capPenalty* is set to 0.1, *walkPenalty* is set to 5, and *prefScore* is set to the approximate mean of all location scores, equal to 6. The model with the base parameters generates an optimal solution equal to 163.22 with a computational time equal to 146.6 seconds.

For all of the following parameter tunings, the model is solved for the five tourist groups at once, providing each tourist group with an optimal route suggestion while considering the other tourist groups as well.

6.1 Effect of the number of routes

This section will examine whether the number of predefined routes given as input to the model affects the objective, and in which direction and what scope. In addition, the effect on the computational time will be explored. As the model assigns the tourist groups to the predefined route providing the largest score, given the constraints, the number of predefined routes is assumed to positively affect the objective. Since the composition of the locations in the predefined routes is random, one might be lucky and receive exceptionally good route suggestions even if the number of routes generated is rather limited. To account for this when exploring the relationship, a set of new unique routes is added to the already existing set instead of generating a new set of routes in total when solving for the different number of routes.

When exploring the effect of the number of routes, everything else is held constant with the input presented in Table 6.1 and the other base values. The model is first solved with only 22 possible routes. This results in an objective of 20.41. By adding a new unique set of predefined routes to the initial set of routes, increasing the number of predefined routes to 46, the objective increases to 54.87. This represents a growth of approximately 169% from the initial run. As presented in Figure 6.1, when increasing the number of routes further, the objective keeps rising, but eventually, it stabilizes. From the figure it is observed that the increase in the objective is very steep for the initial qrowth in the number of routes, but after reaching approximately 140 routes the increase in the objective is minimal. This is because, with an increased number of routes, it is more likely that the tourist groups already are assigned to a good alternative. This implies a smaller increase in the total score. Further, more route alternatives imply less probability of exceeding the capacity constraint which would lead to a penalty in the objective value.

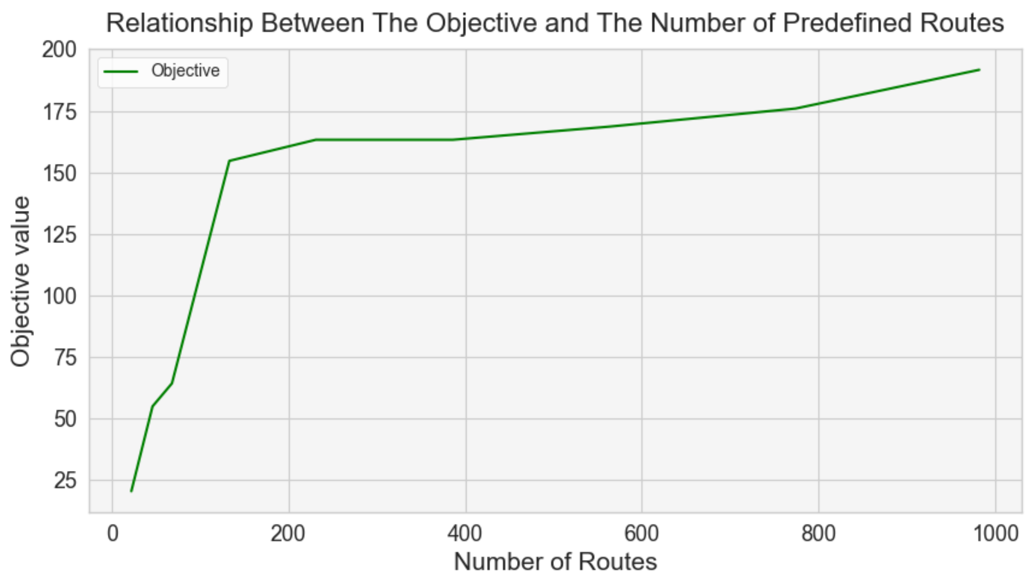


Figure 6.1: Objective vs the number of predefined routes

As the routes are randomly composed of locations, the relationship might look different if the model is solved for a new set of routes. However, the trend with a rising objective when increasing the number of routes but with an eventual stabilization will remain. Furthermore, it is important to emphasize that this example of the relationship between the number of routes and the objective value applies to five tourist groups. With an increased amount of tourist groups, the number of routes required to achieve a stabilized trend will likely be larger.

When considering the expansion of the model, understanding the impact of the number of routes on the computational time is essential. This insight is vital to establish a balance between the acceptable computational time and the size of the objective. Figure 6.2 illustrates the relationship between the computational time and the number of predefined routes given as input to the model.

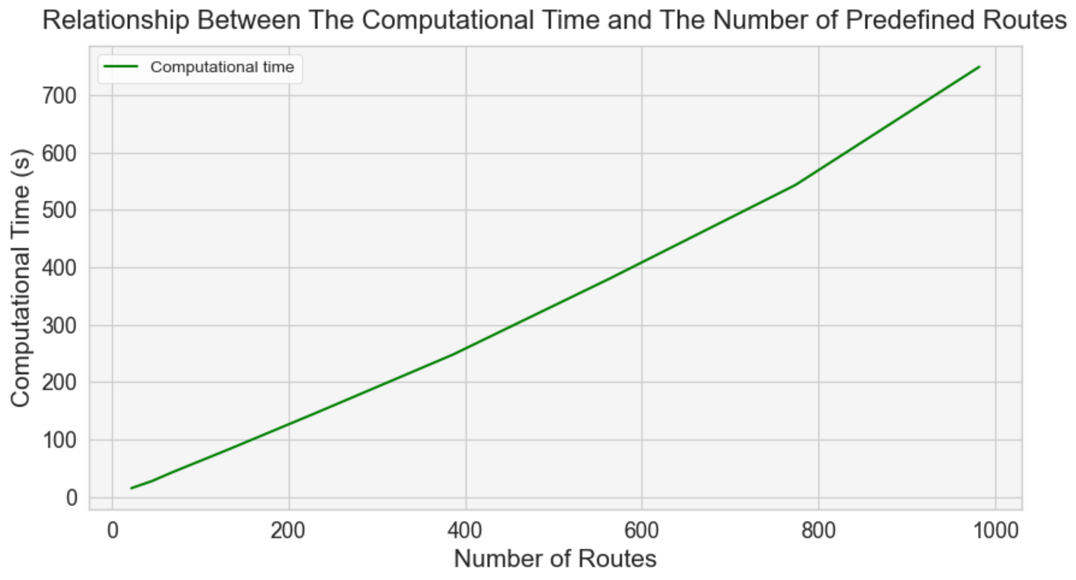


Figure 6.2: Computational time vs the number of predefined routes

An accelerating increase in the computational time when increasing the number of routes is observed for the relationship, but it is not immediately evident in Figure 6.2. To more effectively illustrate this, a simple linear regression is conducted and incorporated into the initial plot, resulting in a more evident visualization presented in Figure 6.3.

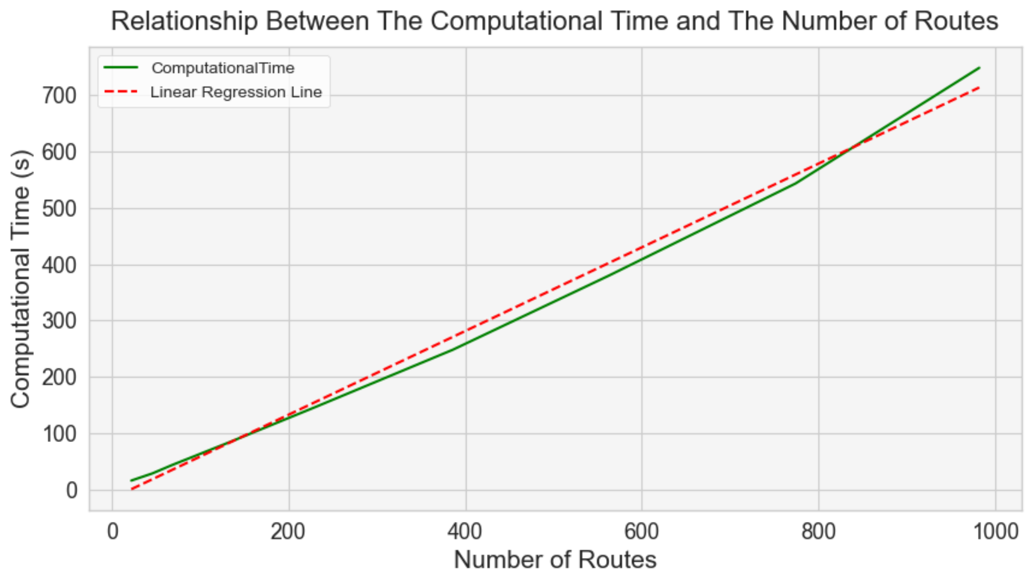


Figure 6.3: Computational time vs the number of predefined routes - Linear regression

The trend visualized in Figure 6.3 is characteristic of exponential growth, but a more detailed analysis would be necessary to determine this. Nevertheless, an exponential model is fitted to the existing data on computational time for the different numbers of routes. This is done to explore the predicted exponential growth in computational time for a further increase in the number of routes, presented in Figure 6.4.

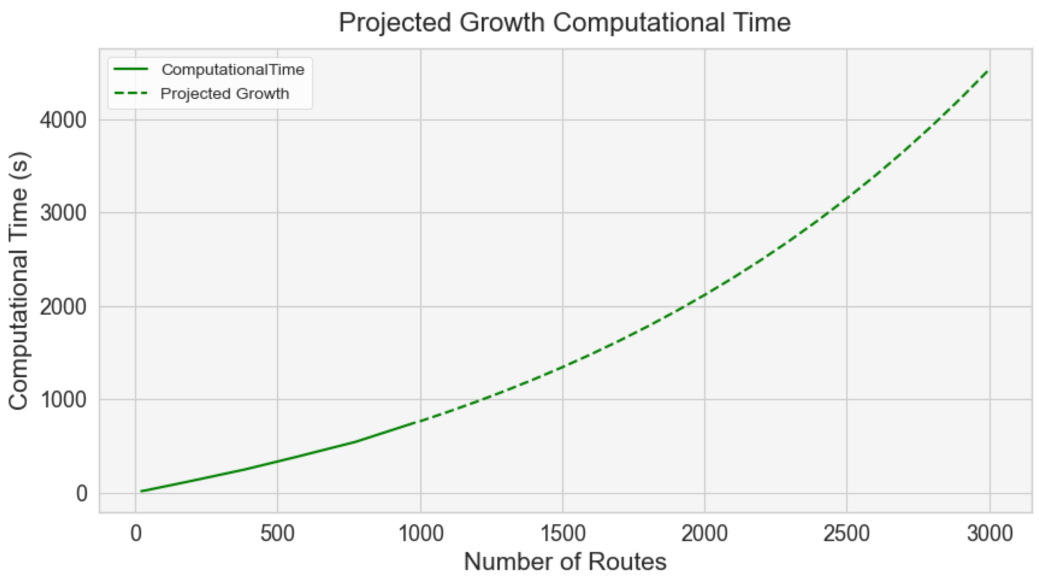


Figure 6.4: Computational time vs the number of predefined routes - Projected exponential growth

As explored, increasing the number of routes increases the computational time, and hence, there is a trade-off between these two. As mentioned, effectively allocating tourists in the city centre of Bergen is crucial for minimizing congestion. To achieve this, the model is reliant on a sufficient number and variety of routes. Contrary, while increasing the number of routes leads to better allocation and hence a larger objective value, it also results in a greater computational time, as illustrated in Figure 6.2. This underscores the observed trade-off; the choice between more efficient allocation with longer computations, or faster computations with a smaller objective and with the increased possibility of congestion.

6.2 Effect of the number of tourist groups

In this section, it will be examined whether increasing the number of tourist groups affects the computational time. When the number of tourist groups increases, the complexity of the problem also grows, directly affecting the computational time. To observe this relationship, the model is solved with different numbers of tourist groups.

The number of tourists in the group, the port, the start time, and the time available is randomly set equal to tourist group P3, presented in Table 6.1 for all the tourist groups in this part of the analysis. Given that the primary aim of this analysis is to observe the relationship between the number of tourist groups and the computational time required, other elements are not considered relevant. Consequently, no food preferences or other preferences are specified.

Table 6.2: Computational time for number of tourist groups

Number of tourist groups	Computational time (s)	Computational increase (s)
5	119	—
10	243	+124
15	379	+136
20	526	+147
25	706	+180

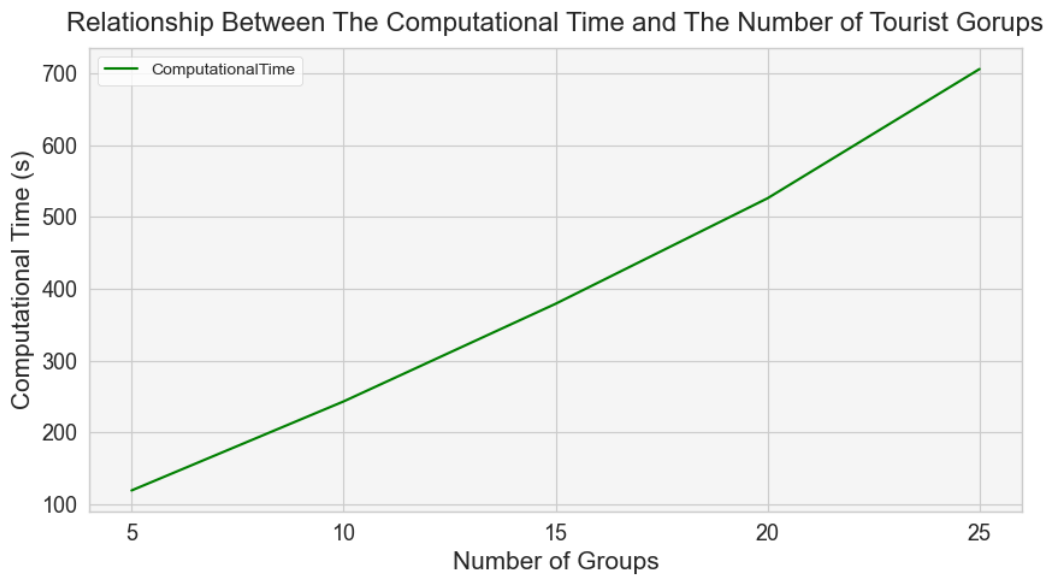


Figure 6.5: Computational time vs the number of Tourist Groups

Table 6.2 indicates an increasing growth in the computational time when the number of tourist groups increases. Similarly to the analysis conducted for the number of routes, the growth is not immediately evident in Figure 6.5. Hence, a simple linear regression is fitted and incorporated into the initial plot presented in Figure 6.6.

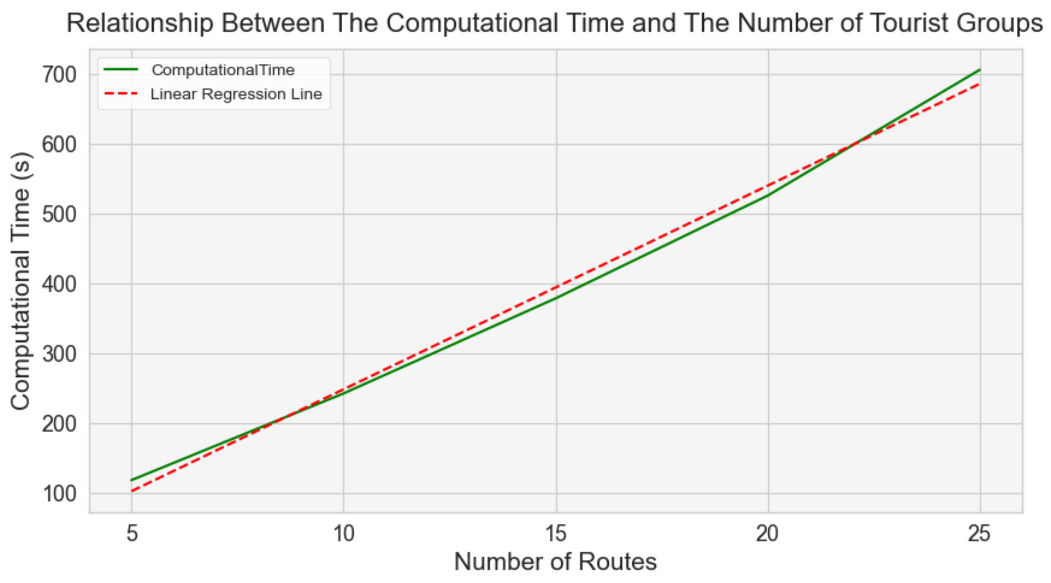


Figure 6.6: Computational time vs the number of tourist groups - Linear regression

The executed analysis focuses on a relatively small-scale instance involving a total of 25 tourist groups and 231 routes, as increasing the number of tourist groups further implies a too complex computation for the scope of this thesis. Consequently, the observed effects correspond to a smaller-scale scenario. However, the computational increase presented in Table 6.2 indicates an accelerating growth in the required computational time as the number of tourist groups increases. Therefore, as in Section 6.1, an exponential model is fitted to the existing data in Table 6.2 to observe the predicted exponential growth, presented in Figure 6.7.

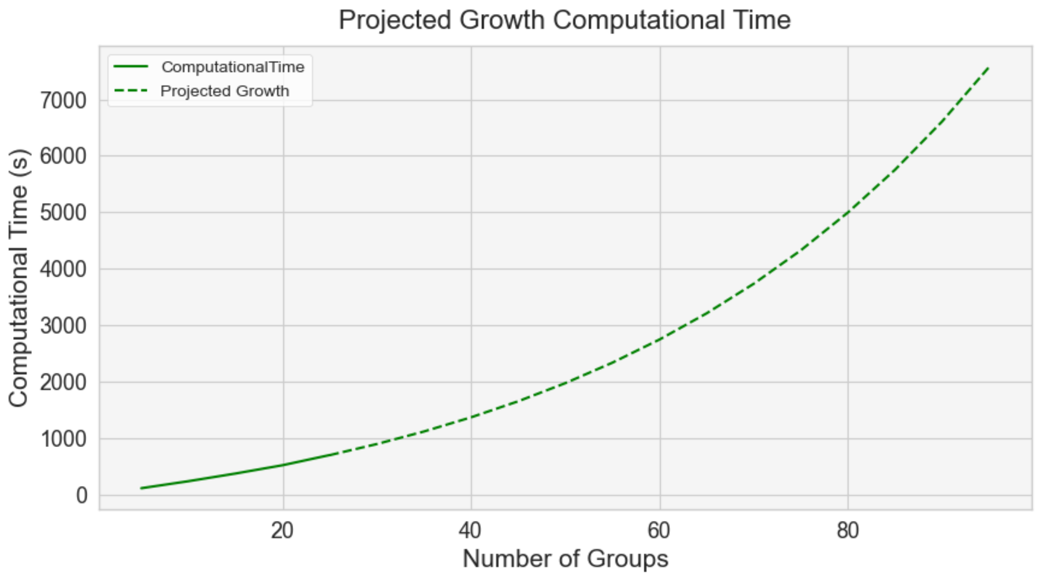


Figure 6.7: Computational time vs the number of tourist groups - Projected exponential growth

6.3 Effect of the capacity penalty

In this section, the effect of the penalty on capacity will be examined, while keeping everything else constant. To observe how the model responds to the value of the penalty parameter *capPenalty*, some of the initial input of the test population is modified while maintaining the base number of tourist groups and the corresponding number of people. To be able to observe the effect of the capacity penalty exclusively, the initial input data across all tourist groups is the same. All the groups start at Bontelabo cruise terminal at time 10:00 and have seven hours available. They do not want to include a food stop and no other preferences are provided.

The initial capacity at each location, except for the ports, is deliberately set to a minimal value equal to five to activate the slack variable, $z_{l,t}$, and consequently *capPenalty*. This is done to observe the model's redistribution of excess demand to alternative routes when faced with capacity constraints. Additionally, a selection of 68 routes from the base of 231 routes is utilized as input to restrict options and observe the effect more clearly.

The analysis commences by setting *capPenalty* to zero, which enables the model to assign tourists to routes in the absence of penalties for capacity violations. This is the initial condition which is based on the assumption that queues are nonexistent at the locations and that the allocated tourist groups do not contribute to any formation of queues. As the tourist groups are unaffected by each other and the capacity constraint this is an idealized setting. This result will serve as a benchmark for further comparison of the model's behaviour. Ideally, all five tourist groups are allocated to route R26 as seen in Table 6.3. Consequently, the slack variable $z_{l,t}$ is activated.

Table 6.3: Capacity penalty set to 0

Tourist Group	Route	Route Score
P0	R26	28.4885
P1	R26	28.4885
P2	R26	28.4885
P3	R26	28.4885
P4	R26	28.4885
Objective		**142.4427**

With the established benchmark, *capPenalty* is further set to 0.1 to observe how the introduction of the penalty affects the allocation. Tourist groups P0, P2 and P4 are then assigned to new routes R48, R30 and R29, which have lower route scores compared to the benchmark. This implies the score retrieved by remaining at the same route subtracting the penalty for exceeding the capacity, is lower than the score retrieved from the new route allocations. Furthermore, tourist group P4 has a route score of 0, indicating that this tourist group has been assigned to a route with a food location, which is not preferred.

Table 6.4: Capacity penalty set to 0.1

Tourist Group	Route	Route Score	Change in score
P0	R48	15.4930	-12.9955
P1	R26	28.4885	0
P2	R30	23.8044	-4.6841
P3	R26	28.4885	0
P4	R29	0	-28.4885
Objective			**83.3745**

Further, *capPenalty* is set to 1. Tourist groups P0, P1 and P4 are then assigned to routes R30, R48 and R29 respectively, with a further reduced objective value as a result of a higher *capPenalty*.

Table 6.5: Capacity penalty set to 1

Tourist Group	Route	Route Score	Change in score
P0	R30	23.8044	-4.6485
P1	R48	15.4930	-12.9955
P2	R26	28.4885	0
P3	R26	28.4885	0
P4	R29	0	-26.5481
Objective			**-32.7254**

It is observed that the principle of opportunity cost drives the reallocation of tourist groups to new routes. The model operates under the premise that the opportunity cost of tourist groups remaining on their initial allocation exceeds the cost associated with being allocated to alternative routes. It is favourable to reallocate tourist groups to new routes rather than to keep them on the initial route. This implies that *capPenalty* has a pivotal influence in determining the fraction of tourist groups reallocated as seen across Table 6.3, 6.4 and 6.5. When determining *capPenalty*, one must balance between contributing to congestion by aligning with individual preferences or assigning tourist groups to routes with locations providing less satisfaction.

Optimally, as mentioned in Section 2.1, the penalty for exceeding the capacity would be unique for the different locations, at least for the different location categories. Waiting in a line is likely more acceptable for locations providing a significant experience, such as visiting Ulriken or Akvariet. Waiting in line for a restaurant when the main purpose is to consume food, is likely less acceptable. Nevertheless, this is not entirely certain, as some might have a strong desire to visit a specific restaurant.

6.4 Effect of the walking penalty

The penalty for being assigned to a route where the walking duration between two consecutive locations is more than 25 minutes, affects whether the tourist groups are assigned to alternative routes instead. Long travel times between consecutive locations may lead to less satisfied tourists, and hence the penalty must be sufficiently large to prevent significant exceedings beyond 25 minutes. This section will study the effect of the walking penalty on the assigning of routes. As for the analysis of the capacity penalty, to be able to see the effects of the walking penalty more clearly, the selection of 68 routes from the base of 231 routes is utilized as input. Additionally, for the other parameters, the base values are established and maintained consistently throughout the analysis.

The *walkPenalty* elucidates the trade-off between satisfied preferences and walking duration for the tourist groups. Consequently, it can be argued that the value of the *walkPenalty* should vary according to the different tourist groups' willingness to walk. However, due to the scope of this thesis, this is not implemented. Furthermore, there are trade-offs among the different parameters. For instance, a large *prefScore* coupled with a small *walkPenalty* would result in the model favouring routes containing preferred locations even if this results in exceeding the walking constraint.

Initially, the model is solved with a *walkPenalty* equal to zero, which implies no consequences on the objective for assigning tourist groups to routes that include longer walking durations than 25 minutes between two consecutive locations. The results are shown in Table 6.6. Tourist groups P0, P2, and P3 are all assigned to routes that activate the slack variable for walking, q_p.

Table 6.6: Walking penalty set to 0

Tourist Group	Route	Slack (q)	Route Score
P0	R9	17.48	23.6435
P1	R1	0	9.9771
P2	R7	3.31	16.6012
P3	R9	17.48	23.6435
P4	R30	0	29.8044
Objective			**103.67**

Table 6.7 presents the results for the model solved with a penalty equal to 2.5. By adding a significant penalty, both P0 and P3 are assigned to new routes that do not exceed the walking constraint. This implies that the negative consequences of the walking penalty are larger than the loss in score when assigned to another route. The loss in score equals 17.0187 while the alternative penalty equals $17.48 \times 5 = 87.4$, making the switch of routes more profitable on the total score.

Table 6.7: Walking penalty set to 2.5

Tourist Group	Route	Slack (q)	Route Score	Change in score
P0	R51	0	6.6248	-17.0187
P1	R1	0	9.9771	0
P2	R7	3.31	16.6012	0
P3	R51	0	12.6249	-11.0186
P4	R30	0	29.8044	0
Objective				**67.36**

Lastly, the model is solved for a *walkPenalty* equal to 5. The results are presented in Table 6.8. A penalty equal to 5 causes P2 to switch routes, despite this implying a route that leads to a decrease of 11.4005 in the score for P2.

Table 6.8: Walking penalty set to 5

Tourist Group	Route	Slack (q)	Route Score	Change in score
P0	R51	0	6.6248	-17.0187
P1	R1	0	9.9771	0
P2	R3	0	5.2007	-11.4005
P3	R51	0	12.6249	-11.0186
P4	R30	0	29.8044	0
Objective				**64.23**

In this analysis, given a set of only 68 routes, the route alternatives are limited. Consequently, the model requires a larger $walkPenalty$ to prevent tourist groups from being assigned to routes exceeding the walking constraint relative to a model with a larger set of routes. As the number of routes expands, the probability of finding an almost as good alternative that satisfies the walking constraint becomes greater.

6.5 Effect of the preference score

The size of the preference score, $prefScore$, affects the weighting of individual preferences relative to reviews from previous visitors. Hence, the choice of preference score will affect whether the routes containing locations preferred or the routes containing locations with high reviews will be selected. To determine a value for the preference score, it is essential to establish what the primary goal of the model is. As mentioned, this thesis focuses on satisfying and making the most out of the cruise tourist's stay in Bergen, while reducing the congestion at the locations. Therefore, it is advantageous to find a suitable middle way. All parameters, except for the preference score, are set as the test population and held constant during the analysis.

To ensure no tourist groups are assigned to locations with low quality, it can be argued that reviews should weigh more than personal preferences. However, since the locations selected for the model are chosen based on suggestions from Bergen Havn and previous visitors' reviews, locations with bad ratings are excluded. The possibility of assigning tourists to low-quality locations is therefore significantly reduced, allowing for adding more weight to individual preferences without the concern of assigning the tourist groups to locations with bad quality.

The relationship between the size of the preference score and the penalty for exceeding the capacity illustrates the weighting between maximizing the cruise tourist's satisfaction and reducing congestion. A low preference score combined with a high penalty for exceeding capacity will result in the model prioritizing assigning the tourists to routes with locations where capacity is not exceeded rather than assigning them to routes with preferred locations. The opposite applies to a high preference score combined with a low penalty for exceeding capacity. The same relationship holds for the preference score and the walking penalty, as mentioned in the previous.

As the proposed model is simplified with predefined routes rather than generating individual routes, the preference score has a less significant effect. To enable tourist groups to be assigned to routes containing their preferred locations, it is necessary to have a large number of predefined routes representing as many route combinations as possible. To be able to run the model within reasonable computational time for this analysis, the model is solved with a set of 231 routes. The solution is therefore not able to fully illustrate the effect of a change in the preference score. However, it is still possible to observe the effect to a certain extent. Table 6.9 shows the number of satisfied location preferences for each of the five tourist groups when the model is solved with different preference scores. The table illustrates a change in the assigned route for tourist group P2, increasing the number of satisfied preferences to one, when *prefScore* is increased from 0 to 3. Furthermore, when *prefScore* is increased to 6, tourist group P4 also achieves one satisfied preference. However, the number of route alternatives is too small to give a clear representation of the relationship between the preference score and the number of satisfied preferences.

Table 6.9: Number of preferences satisfied with different values of *prefScore*

prefScore / **Tourist group**	0	3	6	12	18	24	30
P0	0	0	0	0	0	0	0
P1	0	0	0	0	0	0	0
P2	0	1	1	1	1	1	1
P3	1	1	1	1	1	1	1
P4	0	0	1	1	1	1	1
Sum satisfied	1	2	3	3	3	3	3

7 Discussion

In this section, the limitations of the proposed model, an extension of the model and elements to consider for further research are discussed.

7.1 Limitations

Despite the proposed model provides valueble insigths it is prone to certain limitations which will be discussed in the following.

Firstly, the model does not consider already present individuals at the different locations, such as other tourists and local inhabitants. The model therefore assumes full capacity at the different locations to disposal for allocations. This is an oversimplification of reality, and a way to mitigate this limitation is to assume that a certain percentage of the initial capacity is already in use. This is however a challenging estimate to determine and is therefore dismissed from the proposed model. Alternatively, Google provides an estimate of busy times during the day at the different locations that can be incorporated. This data is however not retrievable through the utilized Google API. A manual extraction is possible, but extensive to conduct for all 91 locations within a reasonable time frame. The estimate of busy times is therefore not included.

Another significant limitation which is elaborated in the previous sections, is the generating of predefined routes given as input to the model. This limits the possibility of customized routes per tourist group. When predefined routes are given to the model, the tourist groups are assigned to the route alternative providing the highest overall score rather than assigning them to individual locations based solely on the highest score per location. However, increasing the number of predefined routes will increase the probability of more suitable routes for each of the tourist groups. Consequently, the weight between the objective and the computational time must be considered.

As mentioned in Section 5, the proposed model requires all user input before it is solved, and therefore it is assumed that all the tourist groups provide this user input within a certain time. This limits the possibility for "late" tourists to get assigned to a route. If a tourist group does not provide user input in the application interface before the deadline, they are not considered, and hence will not be assigned to a route or accounted for regarding the capacity at each location. Further, this limits the possibility of changing preferences or other user input after the model is solved, making the model static.

The input for service times and walking times are generated as average values for all tourist groups. Holding these times equal for all groups, implies an assumption that all groups exhibit similar behaviours and preferences, as mentioned briefly in Section 2.5. This approach does not account for diversity in walking speed, different interests in locations, or different service times for serving groups of different sizes or with different preferences, making the model deterministic. This assumption might oversimplify the diverse needs and behaviours of the different tourist groups. In real-life situations, the tourist groups vary in for instance age and physical abilities, leading to diversity in walking speed. Further, these factors together with the size of the groups will lead to diversity in service time at the different locations. This implies that in reality, the congestion might happen at another time than what is expected, as well as the tourist groups might return late. An alternative to reduce the possibility of not returning on time is to add a security margin for the constraint in Equation (5.10), ensuring the tourist groups are assigned to routes within their time available. Moreover, detailed information on the tourist groups combined with a diverse distribution over the timeslots could give a more realistic approach. An alternative of using stochastic values will be discussed for further research in Section 7.3.

As mentioned in Section 5.2.2.3 where the route constraint is presented, the starting time for each tourist group is determined based on the user input from each tourist group. This implies no flexibility in altering the starting time despite this might provide a better score. For instance, a tourist group might lose the possibility of being assigned to a route with its preferred location because the preferred location opens for instance 30 minutes later. In addition, allowing for a more flexible starting time might reduce congestion as the model can allocate more tourist groups to the same route but at different starting times, as long as each tourist group reaches back to port on time. How to allow for a more flexible starting time is discussed in Section 7.3.

The proposed model does not account for invalid user input, such as a starting time outside the timeslots considered for the model or an unreasonably short available duration.

If users provide a value for available hours below the minimum route duration, the model will generate an infeasible solution. This is a limitation in the sense that none of the other tourist groups will be assigned to a route due to a single invalid input. The same is true for a tourist group providing a starting time equal to 05:00. However, it is envisioned that the application interface will exclude invalid options.

Lastly, as explained in Section 2, the selection of locations for this thesis is retrieved manually from a combination of VisitBergen and Tripadvisor. This methodology represents a static approach to the data gathering process, where the selection was fixed at the time of retrieval which was the 31st of October. Consequently, any changes or updates after retrieval will not be accounted for and this is therefore considered a limitation of the model's ability to be up to date.

7.2 Extending the application of the model

As mentioned in the introduction, the motivation behind the thesis and the development of the proposed model is to be able to allocate potentially 8,000 tourists divided into groups. When implementing a model of such scope, it is essential to understand the factors that can significantly increase both the computational time and the complexity of the model.

In the proposed model, two factors that significantly affect the complexity of the model are the number of tourist groups and the number of routes. As mentioned in Section 4, an increase in the number of jobs or the number of operations in a JSSP model will lead to an even larger increase in the solution scope. For the proposed model, an increase in the number of tourist groups or the number of routes will increase the complexity, as the possible combinations increase.

The complexity is additionally aggravated by the dependencies between constraints. The assignment of routes to a specific tourist group directly influences the available options for other groups considering constraints like location capacities. These dependencies create a dynamic solution space, where the decision of one variable affects the decision of other variables. Furthermore, the complexity is increased due to the nature of the binary and integer decision variables. Contrary to continuous variables that allow for a smooth range of solutions, binary and integer variables result in a solution space that is a set of discrete points. This discretization adds a significant complexity due to it restricting the solution process to consider numerous discrete combinations.

To account for such complexities, heuristics may be implemented. However, as mentioned in Section 4.2.4, with such an implementation the optimal solution is not necessarily achieved, but rather a feasible one. This means there might be combinations of tourist groups and routes that yield a higher objective beyond the one being suggested.

A possible method for reducing the computational complexity and hence the computational time is to increase the timeslots. In the proposed model, the timeslots are given in minutes, leading to a significantly large number of $y_{p,r,l,t}$ variables. Increasing the timeslots will result in reduced precision regarding the times the tourist groups are present at the different locations. Consequently, there is a trade-off between reduced computational time and precision of the $y_{p,r,l,t}$ variable that must be considered in an extension of the model.

7.3 Further research

In the following, different elements for further research will be elaborated.

7.3.1 Use of textual data analysis on reviews

As mentioned in Section 3.1.2, an alternative method to determine the satisfaction of locations implemented in the base score is through *textual data analysis* and *sentiment analysis*. Textual data analysis involves techniques that quantify semantic content from a text systematically (Harris, 2023). Such a technique is sentiment analysis, which evaluates text by identifying the sentiment expressed, then classifying its polarity as positive, neutral or negative (Medhat et al., 2014). In such an analysis, one could determine the relative satisfaction by analyzing the fraction of positive words in each of the reviews left on a location on Google. This relative satisfaction may be included in the measure of satisfaction for a more precise estimate, as one would have a more representative understanding of the wording of the different reviews.

7.3.2 External factors

Factors that are not implemented, but may be of interest and significance for the tourists, are the demand-oriented variables such as prices of the different locations and the current exchange rate. As denoted in Section 1.3, pricing may be of significant influence on the location choices of the tourist groups, but is held outside the model. A suggestion for further research is to implement prices in the scoring, such that high prices at locations may result in reduced scores for the locations in question.

Another external factor to consider for further research that may have significance for the distribution of tourists is weather and season. As mentioned in Section 2.1, approximately 80% of tourists disembark the ship during sunny weather to go to the city centre, contrary to 60-65% when it is raining (Møllerup, 2023). This implies that weather indeed serves as a pivotal determinant in allocating tourists on specific days. Additionally, it is conceivable that tourists may aim to minimize outdoor exposure during adverse weather conditions, resulting in a different demand for locations. Since the weather condition is an external factor that is not considered when generating predefined routes, tourists may be allocated to outdoor and weather dependent locations such as Floibanen and Ulriken in bad weather, resulting in a potential loss in satisfaction which is not reflected in the model's objective.

Tourists are assumed to move from one location to another by foot, the model does therefore not consider public transport like buses, taxis and other micro-mobility modes of transportation such as electric scooters. This is something that can be considered for further research, allowing for locations further away from the city centre to be included as well.

7.3.3 Slack variable for starting time

As discussed for limitations in Section 7.1, more flexibility in the starting time for each tourist group may lead to less probability of congestion and a higher total score. A way to implement a flexible starting time is to introduce a slack variable for the minutes past the starting time given as input from the user and a belonging penalty which is activated when the starting time is modified. Consequently, a new constraint setting the starting time equal to the input for starting time plus a slack variable allowing for modifications can be supplied to the model.

7.3.4 Machine learning: DBSCAN algorithm

An alternative way to supply the selection of locations, is by replicating a study done in Valencia, Spain (Moreno, 2021). The study uses unsupervised machine learning and density-based spatial clustering of applications with noise (DBSCAN) algorithms to discover the most photographed locations in Valencia. The study uses the Flickr photo library to identify clusters around the city of Valencia where photos are geotagged. For this thesis, such an analysis could have been conducted to identify clusters of photographs in Bergen. This is however difficult to execute in the case of Bergen currently. This is due to, as mentioned in Section 2.7.2, the issues that arise as a result of an attempt to distinguish photos from each other and the fact that the number of photos geotagged in Bergen is limited as of now.

7.3.5 Simulation

As mentioned in Section 7.1, assuming the same walking speed and service time for all tourist groups is a significant simplification of reality. The presumed time and location for congestion might not align with the actual occurrence. Further, the presumed time for returning to port might not align with the actual time returned. There are several factors of uncertainty in the model, and an approach to evaluate and test how the model behaves and handles uncertainty is to run simulations of different scenarios. As such, simulation introduces random variables and uncertain parameters in the model, mirroring real-world conditions more closely.

The proposed model is developed based on linear programming principles and under the assumption of certainty, making it deterministic. This is a common approach in classical optimization theory as simplification facilitates the possibility of finding an optimal solution to the problem. Therefore, fixed values across the tourist groups are determined for the walking and service times, as well as strict adherence to locations in the assigned route. In reality, these values are prone to significant uncertainty and variability due to human behaviour among other things.

For further research and in an expansion of the model, it is convenient to study the uncertainty using stochastic values for the parameters. This implies running simulations with random values for uncertain parameters, such as walking duration and service time. Several scenarios where the tourist groups are assigned different values for these parameters based on values that are drawn from distributions may be simulated. Further, a probabilistic approach may also be implemented to test how noncompliance to scheduled locations in the respective routes affects the model. As such, the model accounts for the randomness in human behaviour.

8 Conclusion

Considering Bergen is the largest cruise port in Norway and with the future plans of consolidating all cruise traffic to Bontelabo, it is advantageous to be able to effectively allocate the cruise tourists to prevent queues and overcrowding. Therefore, the purpose of this master's thesis was to develop an optimization model allocating cruise tourists in the city centre of Bergen, optimizing the tourist's satisfaction and at the same time preventing congestion to satisfy the municipality of Bergen.

The proposed model is based on the principles of linear programming and constitutes a mixed integer programming model. As the scope of the problem requires a complex and computational comprehensive model, simplifications were implemented. Initially, random collections of locations based on certain requirements were extracted. Further, the order of the locations in these collections was optimized utilizing a TSP model, to minimize the total walking duration. The results obtained from the TSP model were then provided as a set of routes in the proposed simplified optimization model. The objective of the proposed model was to assign tourist groups to the route in the set of predefined routes providing the largest total score. This included to prevent exceeding the capacity at the locations.

In the analysis of the model, an optimal solution was identified. However, this was achieved using a significantly smaller test population for the number of routes and the number of tourist groups than what is reality. The analysis revealed that as the number of tourist groups and the number of routes increase, the model becomes computationally exhaustive, requiring a longer computational time to identify an optimal solution. Hence, the model was identified as difficult to solve for Bergen as a whole despite the conducted simplifications. For real-life implementation considering up to 8,000 tourists, one would have to balance the trade-off between computational time and the objective. Implementing alternative methods, and a more comprehensive search is therefore mentioned to be considered for further research. Furthermore, the model was developed under the assumption of certainty, making it deterministic. Consequently, for further research, conducting simulations is recommended to examine how the model behaves with stochastic values, and to illuminate the associated uncertainties.

References

Amland, T. (2021). *Håndtering av cruisepassasjerer til bergen etter covid19 pandemien* [Amland Reiselivsutvikling for Bergen Havn AS og Visit Bergen, April 2021].

Boeing, G. (2017). OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Science Direct*, *65*, 126–139. https://doi.org/10.1016/j.compenvurbsys.2017.05.004

Bradley, S. P., Hax, A. C., & Magnanti, T. L. (1977). *Applied mathematical programming*. Massachusetts Institute of Technology. https://web.mit.edu/15.053/www/AppliedMathematicalProgramming.pdf

Brucker, P., Sotskov, Y. N., & Werner, F. (2007). Complexity of shop-scheduling problems with fixed number of jobs: A survey. *ResearchGate*, *65*, 461–481. https://doi.org/10.1007/s00186-006-0127-8

Du, P., Liu, N., Zhang, H., & Lu, J. (2021). An improved ant colony optimization based on an adaptive heuristic factor for the traveling salesman problem. *Hindawi*, *2021*. https://doi.org/10.1155/2021/6642009

Flickr. (n.d.). *About Flickr*. Retrieved December 19, 2023, from https://www.flickr.com/about

Forrest, J. (n.d.). CBC. Retrieved December 5, 2023, from https://coin-or.github.io/Cbc/intro.html

Giovanni, L. D., & Summa, M. D. (n.d.). Methods and models for combinatorial optimization: Exact methods for the traveling salesman problem [University of Padova]. Retrieved December 9, 2023, from https://www.math.unipd.it/~luigi/courses/metmodoc1718/m08.01.TSPexact.en.pdf

Google Maps. (n.d.). *Places API*. Retrieved December 12, 2023, from https://developers.google.com/maps/documentation/places/web-service

Harris, T. (2023). Literature review: What is textual analysis, and can we use it to measure corporate culture? *Springer*, 45–68. https://doi.org/10.1007/978-3-031-30156-8_3

Houston, M. B., Bettencourt, L. A., & Wenger, S. (1999). The relationship betweenwaiting in a service queueand evaluations of servicequality: A field theoryperspective. *Psychology Marketing*, *15*, 735–753. https://doi.org/10.1002/(SICI)1520-6793(199812)15:8<735::AID-MAR2>3.0.CO;2-9

Innovation Norway. (2019). Cruise report 2019 [Accessed: 2023-12-14]. https://assets.simpleviewcms.com/simpleview/image/upload/v1/clients/norway/Cruisereport_2019_Innovation_Norway_756af9e6-677f-47f6-837b-bcad3ce82b46.pdf

Jones, A. (2023). *(TSP) travelling salesman optimization problem* [Doctoral dissertation]. ResearchGate. https://doi.org/10.13140/RG.2.2.12029.54247

Kringstad, A. K. (2023). *Anbefaler forslag til arealstrategi for utvikling av dokken.* Retrieved December 19, 2023, from https://www.bergen.kommune.no/politikk/byradet/behandlede-saker/bymiljo/anbefaler-forslag-til-arealstrategi-for-utvikling-av-dokken

Kubiak, W. (2022). A book of open shop scheduling: Algorithms, complexity and applications (1st ed.). *325*. https://doi.org/10.1007/978-3-030-91025-9

Laporte, G., & Nobert, Y. (2008). Exact algorithms for the vehicle routing problem. *Science Direct*, *132*, 147–184. https://doi.org/10.1016/S0304-0208(08)73235-3

Li, W., Ding, Y., Yang, Y., Sherratt, R. S., Park, J. H., & Wang, J. (2020). Parameterized algorithms of fundamental NP-hard problems: A survey. *Springer Link*, *10*. https://doi.org/10.1186/s13673-020-00226-w

Medhat, W., Hassan, A., & Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. *Science Direct*, *5*, 1093–1113. https://doi.org/10.1016/j.asej.2014.04.011

Møllerup, N. (2023). *Cruiseutvikling bergen havn 2023* [Nils Møllerup for Bergen Havn].

NHO Reiseliv. (2019). *Reiselivets verdi: Destinasjonsanalyser bergen.* Retrieved November 8, 2023, from https://www.nhoreiseliv.no/tall-og-fakta/reiselivets-verdi-destinasjon/#part2

OpenRouteService. (n.d.). *Services.* Retrieved December 19, 2023, from https://openrouteservice.org/services/

PyPI. (n.d.). *Pulp 2.7.0.* Retrieved December 5, 2023, from https://pypi.org/project/PuLP/#:~:text=PuLP%20is%20an%20LP%20modeler,GUROBI%20to%20solve%20linear%20problems.

Ragsdale, C. T. (2022). *Spreadsheet modeling and decision analysis: A practical introduction to business analytics* (9th ed.). Cengage.

Rossen, E., & Nätt, T. H. (2023). *API.* Retrieved December 4, 2023, from https://snl.no/API

Tripadvisor. (n.d.). *About tripadvisor.* Retrieved October 29, 2023, from https://tripadvisor.mediaroom.com/US-about-us

Tripadvisor. (2023). *Cafes in bergen.* Retrieved October 31, 2023, from https://www.tripadvisor.co.uk/Restaurants-g190502-c8-Bergen_Hordaland_Western_Norway.html

Tsirlin, A., & Balunov, A. (2022). Job-shop scheduling problem and its solution method. *Research Square*, *1*. https://doi.org/10.21203/rs.3.rs-2294412/v1

VisitBergen. (n.d.-a). *About us.* Retrieved October 31, 2023, from https://en.visitbergen.com/about

VisitBergen. (n.d.-b). *Attractions in bergen.* Retrieved October 31, 2023, from https://en.visitbergen.com/things-to-do/attractions

VisitBergen. (n.d.-c). *Reiselivet i bergen: Tall 2019.* Retrieved October 30, 2023, from http://2019.visitbergen.com/2.html#:~:text=Reiseliv%20er%20en%20viktig%20n%C3%A6ring,p%C3%A5%205%2C1%20milliarder%20kroner.

Woeginger, G. J. (2018). The open shop scheduling problem. *4*, 4:1–4:12. https://doi.org/10.4230/LIPIcs.STACS.2018.4

Appendices

A Selection of locations

Table A.1: Table of locations and categories

Location	Category
Bontelabo cruise terminal	Port
Jekteviksterminalen	Port
Bryggen	Historical sites, Family Friendly
Bergen Kunsthall	Museums & Galleries
Bergenhus Festning	Historical sites
Bergenhus Festningsmuseum	Museums & Galleries
Bergen Maritime Museum	Museums & Galleries
Bryggen Museum	Museums & Galleries
Det Hanseatiske Museum & Schøtstuene	Museums and Galleries
Bergen Tekniske Museum	Museums & Galleries, Family Friendly
Gestapomuseet	Museums & Galleries
Haakonshallen	Historical sites
KODE Lysverket	Museums & Galleries
KODE Rasmus Meyer	Museums & Galleries
KODE Stenersen	Museums & Galleries
KODE Permanenten	Museums & Galleries
Kunsthall 3,14	Museums & Galleries
Muséhagen	Historical sites
Norges Fiskerimuseum	Museums & Galleries, Family Friendly
Rosenkrantztårnet	Historical sites, Family Friendly
Skolemuseet	Museums & Galleries
Storeblå visningssenter for havbruk	Museums & Galleries, Nature
Universitetsmuseet	Museums & Galleries, Family Friendly
VilVite	Museums & Galleries, Family Friendly, Activities
Akvariet	Family Friendly, Activities
Floibanen	Family Friendly, Activities, Nature
Nordnes sjøbad	Family Friendly, Activities, Nature
Bus stop for Ulriken express	Family Friendly, Activities, Nature
Mariakirken	Churches
Bergen Domkirke	Churches
Johanneskirken	Churches
St.Jørgens kirke	Churches
St.Paul kirke	Churches
Nykirken	Churches
AdO Arena	Family Friendly, Activities
Fisketorget	Family Friendly, Activities

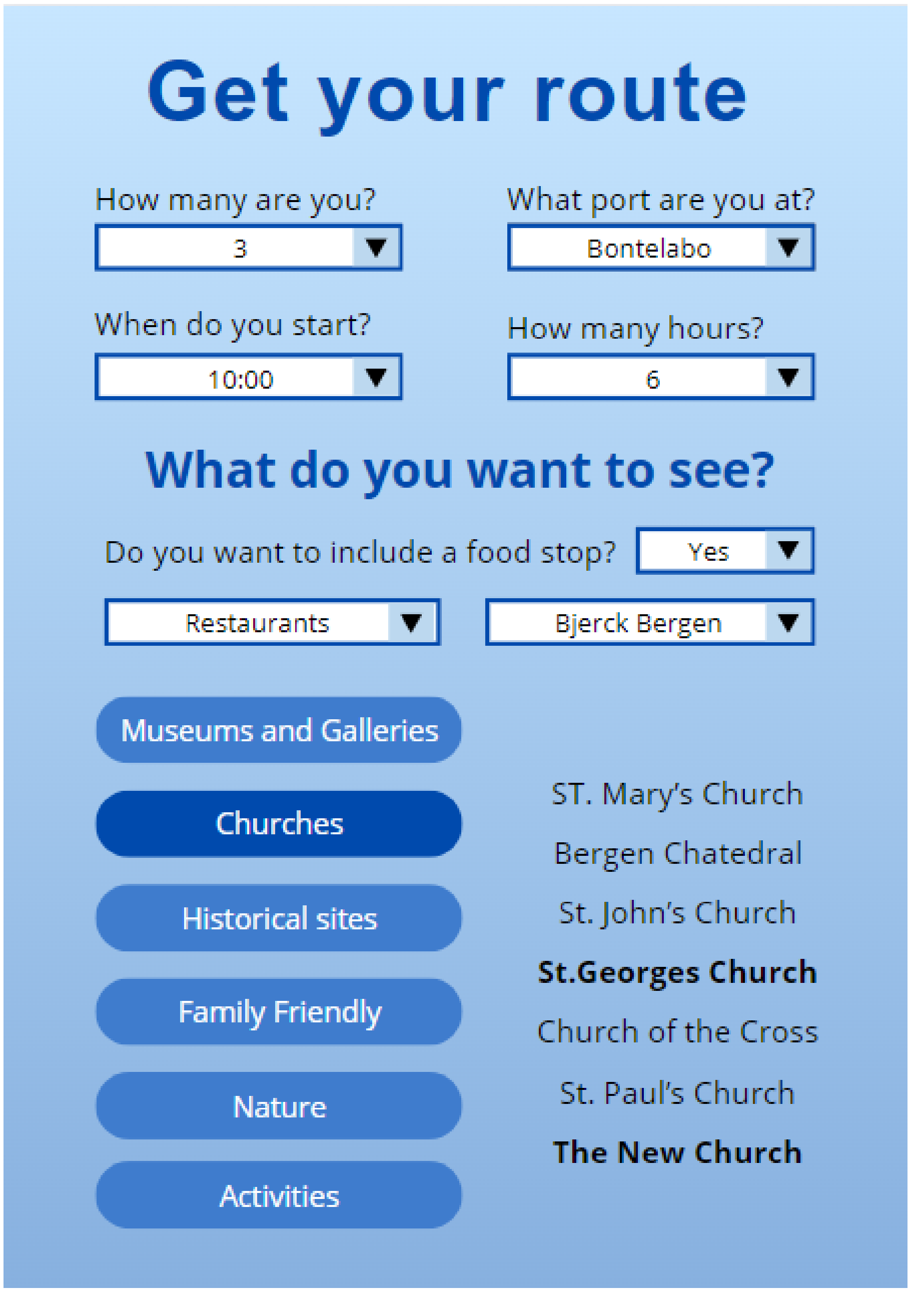
Table A.2: Table of locations in subcategory Restaurants within category Food

Location	Subcategory
Colonialen Kranen	Restaurants
Brasilia Bergen	Restaurants
Lola Bistro Bergen	Restaurants
Rebel Bergen	Restaurants
Pingvinen Bergen	Restaurants
Le Mathis Bistro Bergen	Restaurants
Namastey Bergen	Restaurants
FG Restaurant & Bar	Restaurants
Bryggen Tracteursted	Restaurants
Såpas Bergen	Restaurants
Enhjorningen fiskerestaurant	Restaurants
Fjellskaal Sjomatrestaurant	Restaurants
Banzha Bergen	Restaurants
Hekkan Burger Bergen	Restaurants
Restaurant 1877	Restaurants
Colonialen 44	Restaurants
To kokker Bergen	Restaurants
Bien Basar Bergen	Restaurants
Bien Centro Bergen	Restaurants
Indian Gate Bergen	Restaurants
Escalón Mundo Floien	Restaurants
Allmuen Bistro Bergen	Restaurants
Bryggeloftet & Stuene	Restaurants
Casa del Toro Bergen	Restaurants
Royal Gourmetburger & Gin Bergen	Restaurants
Bjerck Bergen	Restaurants
Colonialen Brasseriet Bergen	Restaurants

Table A.3: Table of locations in subcategory Cafés within category Food

Location	Subcategory
Cafe Opera Bergen	Cafés
Løvetann Café & Bistro	Cafés
Godt Brød Marken	Cafés
Solros Bergen	Cafés
Godt Brød Fløyen	Cafés
Det Lille Kaffekompaniet	Cafés
Vågal Kaffe & Vin	Cafés
We are Brgn Bistro	Cafés
Dromedar Kaffebar	Cafés
Fjaak Chocolate Bergen	Cafés
Kaffemisjonen Bergen	Cafés
Baker Brun Torget	Cafés
Pygmalion Bergen	Cafés
Blom Bergen	Cafés
Smakverket Bergen	Cafés
Espresso House Strømgaten	Cafés
Godt Brød Muséplassen	Cafés
Starbucks Kjøttbasaren	Cafés
Baker Brun Bryggen	Cafés
Kaffelade Bergen	Cafés
Godt Brød Korskirken	Cafés
Baker Brun Strømgaten	Cafés
Bergen Kaffebrenneri	Cafés
Starbucks Neumanns gate	Cafés
Lie Nielsen	Cafés
Baker Brun Telegrafen	Cafés
Trikken 106	Cafés

B Layout for Application Interface


Get your route

How many are you? 3

What port are you at? Bontelabo

When do you start? 10:00

How many hours? 6

What do you want to see?

Do you want to include a food stop? Yes

Restaurants

Bjerck Bergen

ST. Mary's Church

Bergen Chatedral

St. John's Church

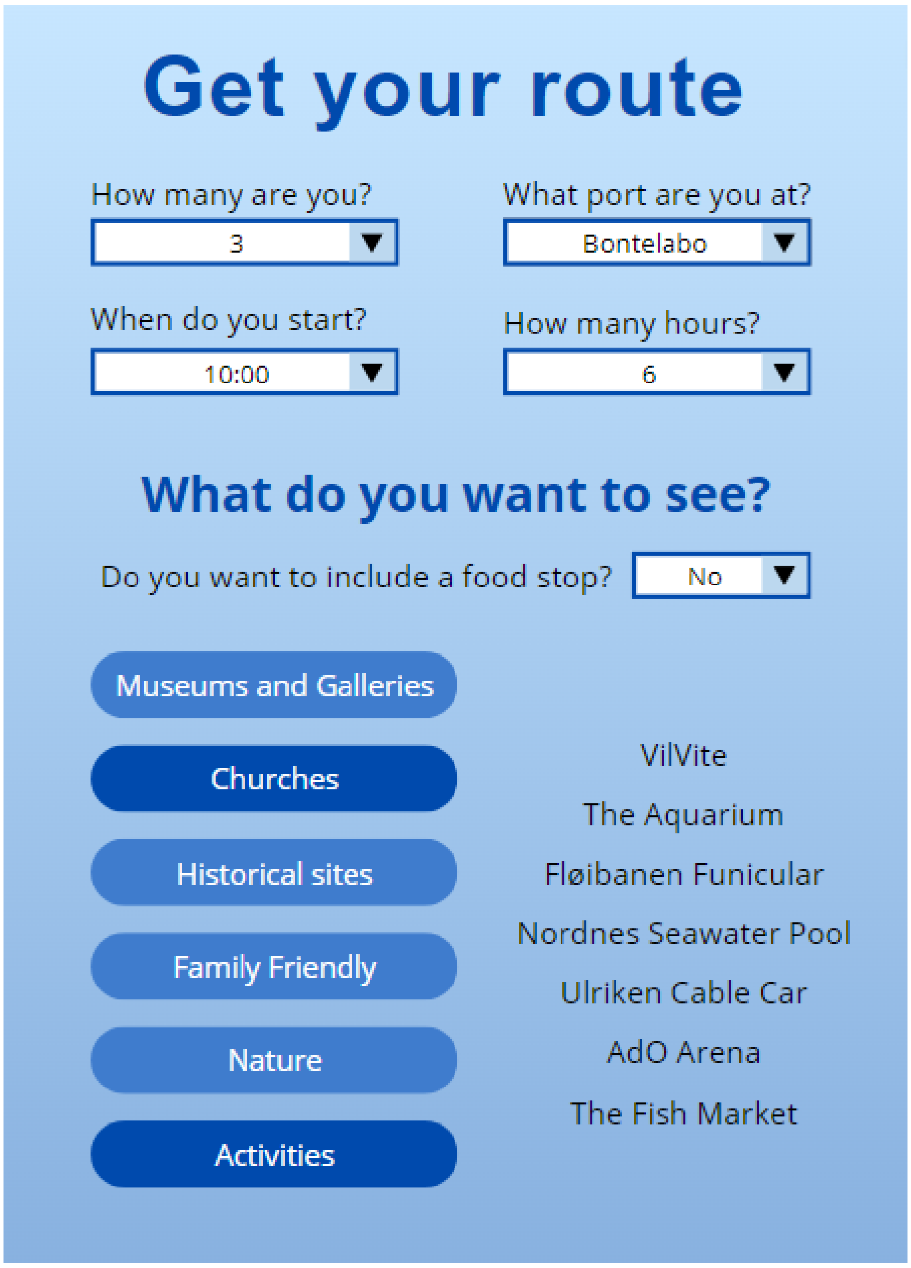
St.Georges Church

Church of the Cross

St. Paul's Church

The New Church

Figure B.1: Example from Application Interface where the tourist group want to include food and have preffered "Bjerck Bergen". In addition the user have selected "Churches" and two specific churches.



Get your route

How many are you? 3

What port are you at? Bontelabo

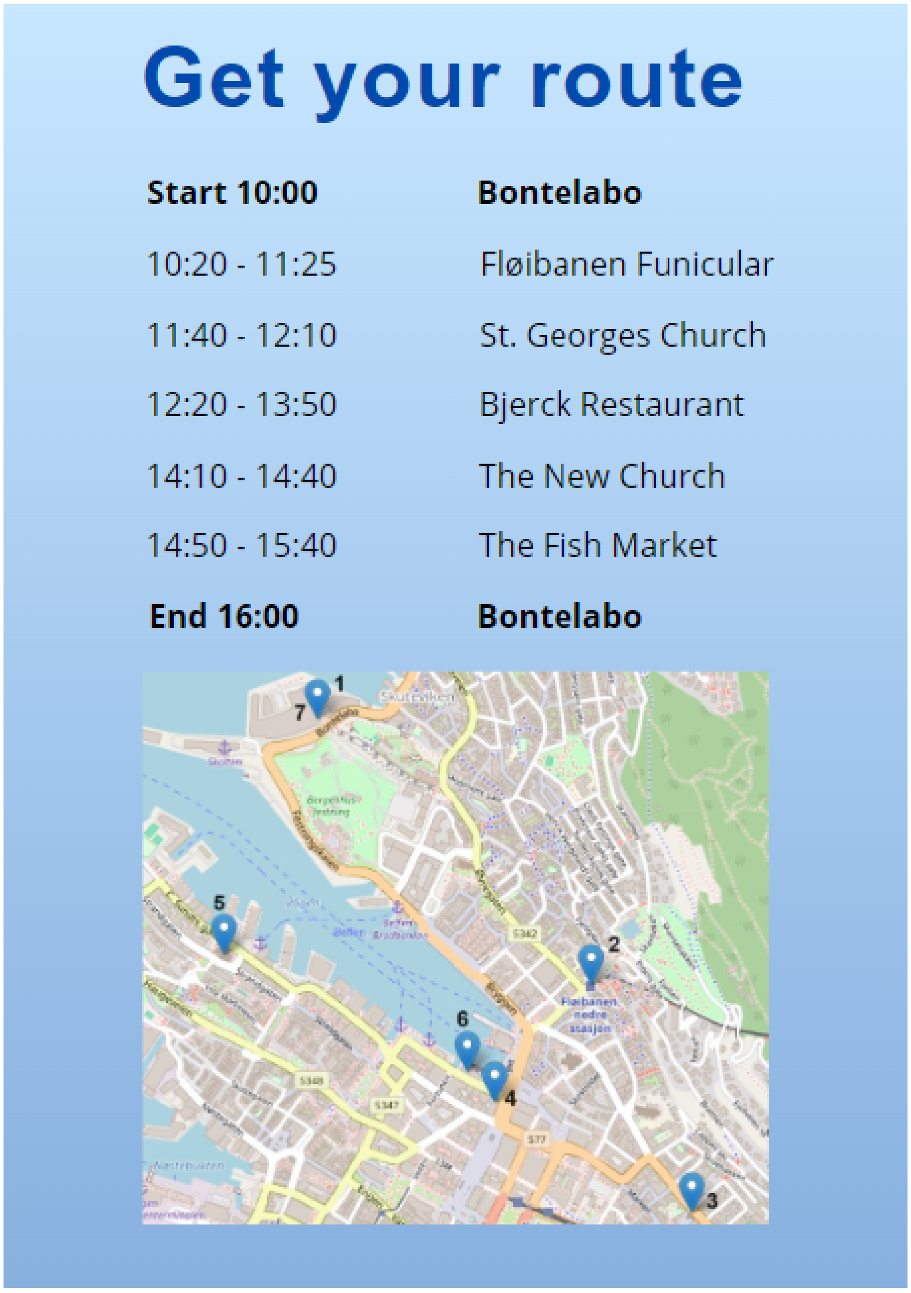
When do you start? 10:00

How many hours? 6

What do you want to see?

Do you want to include a food stop? No

Figure B.2: Example from Application Interface where the tourist group does not want to include food, but have selected two specific "Churches" and category "Activities" as its whole.

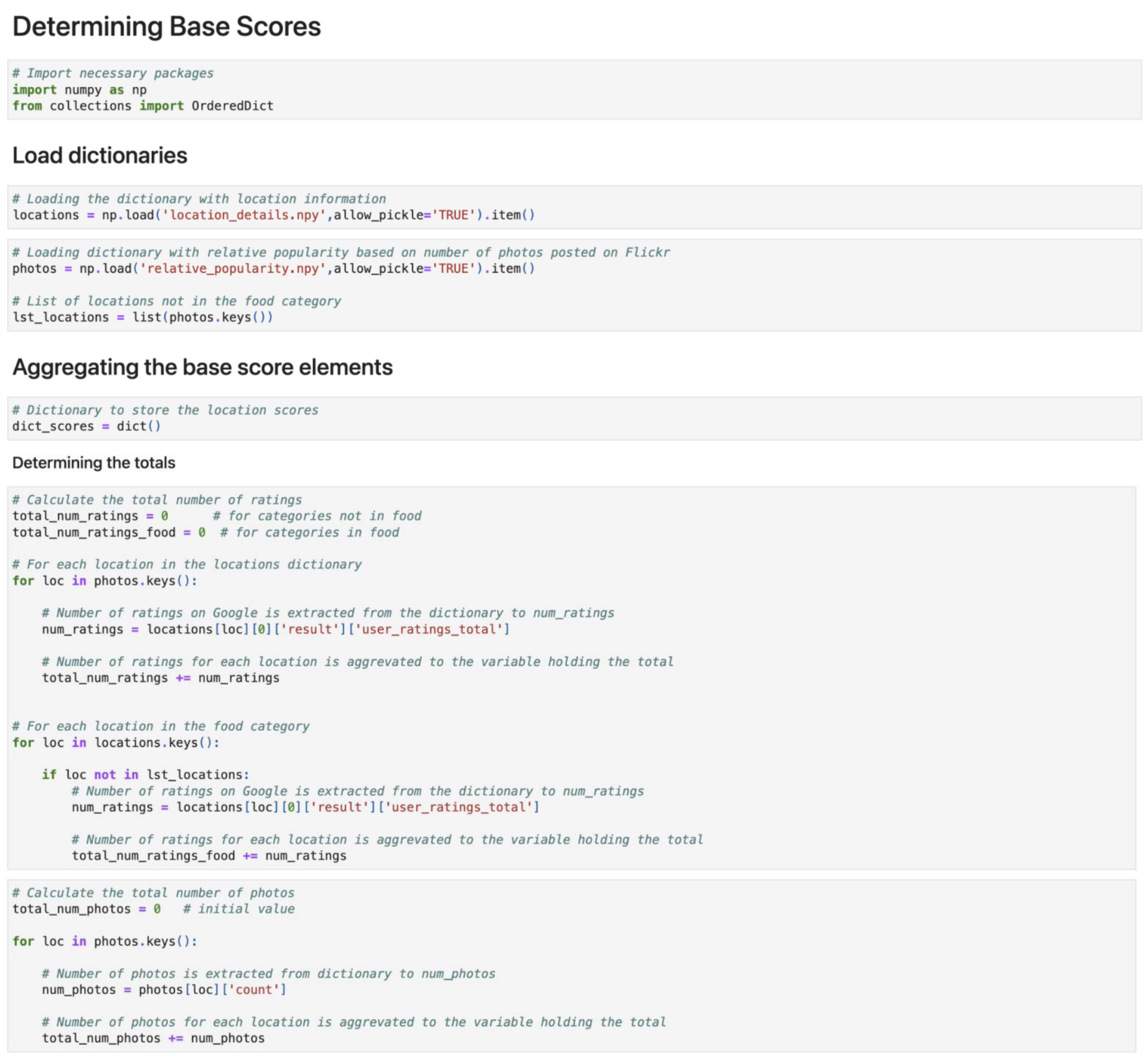


Get your route

Start 10:00	**Bontelabo**
10:20 - 11:25	Fløibanen Funicular
11:40 - 12:10	St. Georges Church
12:20 - 13:50	Bjerck Restaurant
14:10 - 14:40	The New Church
14:50 - 15:40	The Fish Market
End 16:00	**Bontelabo**

Figure B.3: Example from Application Interface output

C Generating base scores in Python



Determining Base Scores

```python
# Import necessary packages
import numpy as np
from collections import OrderedDict
```

Load dictionaries

```python
# Loading the dictionary with location information
locations = np.load('location_details.npy',allow_pickle='TRUE').item()
```

```python
# Loading dictionary with relative popularity based on number of photos posted on Flickr
photos = np.load('relative_popularity.npy',allow_pickle='TRUE').item()

# List of locations not in the food category
lst_locations = list(photos.keys())
```

Aggregating the base score elements

```python
# Dictionary to store the location scores
dict_scores = dict()
```

Determining the totals

```python
# Calculate the total number of ratings
total_num_ratings = 0      # for categories not in food
total_num_ratings_food = 0  # for categories in food

# For each location in the locations dictionary
for loc in photos.keys():

    # Number of ratings on Google is extracted from the dictionary to num_ratings
    num_ratings = locations[loc][0]['result']['user_ratings_total']

    # Number of ratings for each location is aggrevated to the variable holding the total
    total_num_ratings += num_ratings


# For each location in the food category
for loc in locations.keys():

    if loc not in lst_locations:
        # Number of ratings on Google is extracted from the dictionary to num_ratings
        num_ratings = locations[loc][0]['result']['user_ratings_total']

        # Number of ratings for each location is aggrevated to the variable holding the total
        total_num_ratings_food += num_ratings
```

```python
# Calculate the total number of photos
total_num_photos = 0   # initial value

for loc in photos.keys():

    # Number of photos is extracted from dictionary to num_photos
    num_photos = photos[loc]['count']

    # Number of photos for each location is aggrevated to the variable holding the total
    total_num_photos += num_photos
```

Adding the elements togheter

```
# Add score to dictionary of scores
for loc in locations.keys():

    # If location has present number of photos from Flickr
    if loc in photos:

        # Rating from Google
        ratings = locations[loc][0]['result']['rating']

        # Number of reviews from Google
        num_ratings = locations[loc][0]['result']['user_ratings_total']
        # Calculating the relative number of reviews and multipluying it with 100
        relative_ratings = (num_ratings / total_num_ratings) * 100

        # Number of photos from Flickr
        num_photos = photos[loc]['count']
        # Calculating the relative number of photos
        relative_popuarity = (num_photos / total_num_photos) * 100

        # Consolidating the the components
        dict_scores[loc] = ratings + relative_ratings + relative_popuarity

    # For the locations with absent number of photos from Flickr
    else:

        # Number of reviews from Google
        num_ratings = locations[loc][0]['result']['user_ratings_total']
        # Calculating the relative number of reviews and multipluying it with 100
        relative_ratings = (num_ratings / total_num_ratings_food) * 100

        # Consolidating the the components
        dict_scores[loc] = ratings + relative_ratings


# Set score for the two ports equal to zero
dict_scores["Bontelabo cruise terminal"] = 0
dict_scores["Jekteviksterminalen"] = 0
```

Save dictionary with scores

```
# Save dictionary with scores
np.save('location_scores.npy', dict_scores)
```

D Base Score dictionary

Location	**Score**
Bontelabo cruise terminal	0.00
Jekteviksterminalen	0.00
Bryggen	7.39
Bergen Kunsthall	8.29
Bergenhus Festning	20.88
Bergenhus Festningsmuseum	5.23
Bergen maritime museum	7.56
Bryggen museum	8.50
Det Hanseatic Museum og Schøtstuene	4.52
Bergen tekniske museum	4.90
Gestapomuseet Bergen	4.53
Haakonshallen	5.12
KODE Lysverket	7.07
KODE Rasmus Meyer	5.55
KODE Stenersen	4.66
KODE Permanenten	4.69
Kunsthall 3,14	4.64
Muséhagen	4.91
Norges Fiskerimuseum	8.39
Rosenkrantztårnet Bergen	8.64
Skolemuseet Bergen	4.32
Storeblå visningssenter for havbruk	4.64
The University Museum of Bergen	4.94
VilVite	8.14
Akvariet i Bergen	19.45
Floibanen i Bergen	14.74
Nordnes sjobad	5.77
Buss stopp for Ulriken express	4.73
Mariakirken Bergen	9.03
Bergen Domkirken	5.06
Johanneskirken Bergen	8.92
St.Jorgens kirke Bergen	4.52
St.Pauls kirke	4.84
Nykirken Bergen	5.66
AdO Arena	9.81
Fisketorget i Bergen	28.49

Table D.1: Base scores for locations not in food category

Location	**Score**
Colonialen Kranen	4.62
Brasilia Bergen	6.90
Lola Bistro Bergen	4.26
Rebel Bergen	4.28
Pingvinen Bergen	7.20
Le Mathis Bistro Bergen	4.25
Namastey Bergen	4.80
FG Restaurant & Bar	4.38
Bryggen Tracteursted	4.80
Såpas Bergen	4.40
Enhjorningen fiskerestaurant	5.05
Fjellskaal Sjomatrestaurant	6.62
Banzha Bergen	4.27
Hekkan Burger Bergen	4.41
Restaurant 1877	4.68
Colonialen 44	4.17
To kokker Bergen	4.34
Bien Basar Bergen	4.31
Bien Centro Bergen	5.04
Indian Gate Bergen	4.75
Escalón Mundo Floien	4.61
Allmuen Bistro Bergen	4.73
Bryggeloftet & Stuene	9.03
Casa del Toro Bergen	4.60
Royal Gourmetburger & Gin Bergen	5.38
Bjerck Bergen	5.20
Colonialen Brasseriet Bergen	4.69

Table D.2: Base scores for Restaurants

Location	**Score**
Cafe Opera Bergen	5.76
Løvetann Café & Bistro	4.97
Godt Brød Marken	4.83
Solros Bergen	4.43
Godt Brød Fløyen	4.94
Det Lille Kaffekompaniet	5.50
Vågal Kaffe & Vin	4.68
We are Brgn Bistro	4.11
Dromedar Kaffebar	4.58
Fjaak Chocolate Bergen	4.53
Kaffemisjonen Bergen	5.16
Baker Brun Torget	4.28
Pygmalion Bergen	4.91
Blom Bergen	4.54
Smakverket Bergen	4.44
Chillout Travel Store	4.27
Espresso House Strømgaten	5.11
Godt Brød Muséplassen	4.53
Starbucks Kjøttbasaren	5.56
Baker Brun Bryggen	4.91
Kaffelade Bergen	4.69
Godt Brød Korskirken	4.52
Baker Brun Strømgaten	4.28
Bergen Kaffebrenneri	4.95
Starbucks Neumanns gate	4.95
Lie Nielsen	4.17
Baker Brun Telegrafen	4.21
Trikken 106	4.39

Table D.3: Base scores for Cafés

E Generating routes in Python

Generating random predefined routes

```python
# Import necessary packages
import random
import numpy as np
import pulp
```

Loading inputdata

List of locations:

```python
# Loading the dictionary of the locations with information
dict_locations = np.load('Data/location_details.npy', allow_pickle = 'TRUE').item()

lst_locations = list(dict_locations.keys())           # List of location names
lst_attractions = list(dict_locations.keys())[:36]    # List of locations outside the food-category
lst_food = list(dict_locations.keys())[36:]           # List of locations within the food-category
```

Dictionary of travel distances:

```python
# Loading the matrix of walking durations
duration_matrix = np.loadtxt('Data/durations_min.txt', delimiter='|', dtype='<U35')
durations = duration_matrix[1:]  # Removes the first row consisting of location names
```

```python
# Creating a dictionary with dictionaries for walking durations
travel_times = dict()

# For all locations, add the walking duration from that start location to all other locations
for i in range(len(lst_locations)):

    start_location = lst_locations[i]  # Start location
    times_temp = dict()                # Dictionary with walking duration from start_location to the other locations

    # For each location and duration in the list of locations and the matrix-row for the base
    # location, add the location as key and the duration as value to the dictionary
    for loc,dur in zip(lst_locations, durations[i]):
        times_temp[loc] = dur.astype(float)

    # Add dictionary with walking durations from start_location to the other locations
    travel_times[start_location] = times_temp

np.save("Data/travel_times.npy", travel_times)
```

Dictionary of service times:

```python
# Loading the dictionary which consists of service times
locations_service_time = np.load('Data/ServiceTimes.npy', allow_pickle = 'TRUE').item()
```

```python
# Creating dictionary with all locations and belonging service time
service_times = dict()

# For all locations add the belonging service time to the dictionary
for loc in locations_service_time.keys():
    service_times[loc] = locations_service_time[loc]["Service time"]
```

Functions for generating random collections of locations

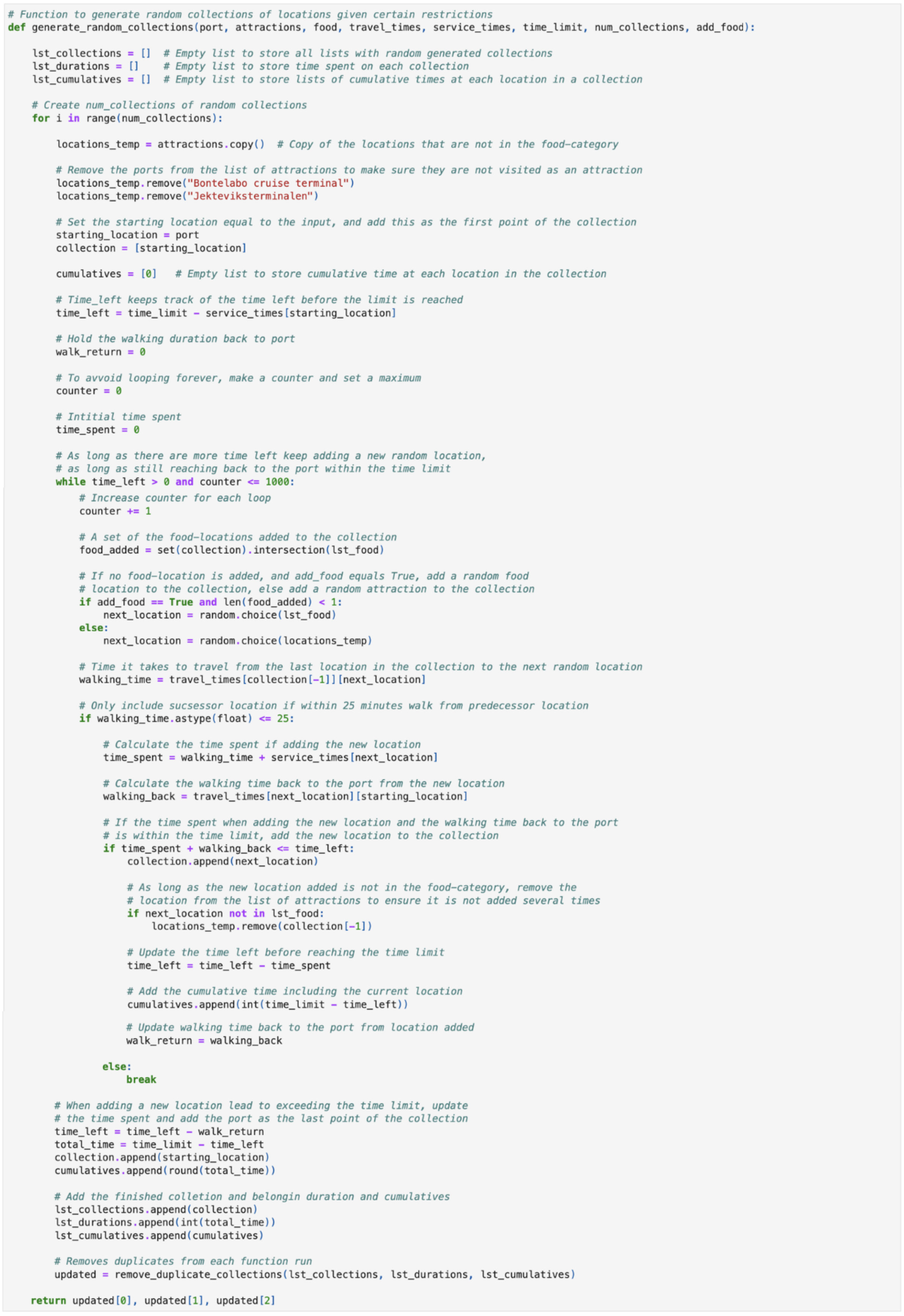
```python
# Function to remove duplicate collections of locations, and belonging durations and cumulative-times
def remove_duplicate_collections(lst_collections, lst_durations, lst_cumulatives):

    # Remove duplicate collections
    unique_collections = []
    unique_durations = []
    unique_cumulatives = []

    # For all collections, check if the collection is already added to the list of unique collections,
    # if not, add the collection, the belongig duration and cumulative-times to the relevant lists
    for i in range(len(lst_collections)):

        if lst_collections[i] not in unique_collections:
            unique_collections.append(lst_collections[i])
            unique_durations.append(lst_durations[i])
            unique_cumulatives.append(lst_cumulatives[i])

    return unique_collections, unique_durations, unique_cumulatives
```



```python
# Function to generate random collections of locations given certain restrictions
def generate_random_collections(port, attractions, food, travel_times, service_times, time_limit, num_collections, add_food):

    lst_collections = []  # Empty list to store all lists with random generated collections
    lst_durations = []    # Empty list to store time spent on each collection
    lst_cumulatives = []  # Empty list to store lists of cumulative times at each location in a collection

    # Create num_collections of random collections
    for i in range(num_collections):

        locations_temp = attractions.copy()  # Copy of the locations that are not in the food-category

        # Remove the ports from the list of attractions to make sure they are not visited as an attraction
        locations_temp.remove("Bontelabo cruise terminal")
        locations_temp.remove("Jekteviksterminalen")

        # Set the starting location equal to the input, and add this as the first point of the collection
        starting_location = port
        collection = [starting_location]

        cumulatives = [0]   # Empty list to store cumulative time at each location in the collection

        # Time_left keeps track of the time left before the limit is reached
        time_left = time_limit - service_times[starting_location]

        # Hold the walking duration back to port
        walk_return = 0

        # To avvoid looping forever, make a counter and set a maximum
        counter = 0

        # Intitial time spent
        time_spent = 0

        # As long as there are more time left keep adding a new random location,
        # as long as still reaching back to the port within the time limit
        while time_left > 0 and counter <= 1000:
            # Increase counter for each loop
            counter += 1

            # A set of the food-locations added to the collection
            food_added = set(collection).intersection(lst_food)

            # If no food-location is added, and add_food equals True, add a random food
            # location to the collection, else add a random attraction to the collection
            if add_food == True and len(food_added) < 1:
                next_location = random.choice(lst_food)
            else:
                next_location = random.choice(locations_temp)

            # Time it takes to travel from the last location in the collection to the next random location
            walking_time = travel_times[collection[-1]][next_location]

            # Only include sucsessor location if within 25 minutes walk from predecessor location
            if walking_time.astype(float) <= 25:

                # Calculate the time spent if adding the new location
                time_spent = walking_time + service_times[next_location]

                # Calculate the walking time back to the port from the new location
                walking_back = travel_times[next_location][starting_location]

                # If the time spent when adding the new location and the walking time back to the port
                # is within the time limit, add the new location to the collection
                if time_spent + walking_back <= time_left:
                    collection.append(next_location)

                    # As long as the new location added is not in the food-category, remove the
                    # location from the list of attractions to ensure it is not added several times
                    if next_location not in lst_food:
                        locations_temp.remove(collection[-1])

                    # Update the time left before reaching the time limit
                    time_left = time_left - time_spent

                    # Add the cumulative time including the current location
                    cumulatives.append(int(time_limit - time_left))

                    # Update walking time back to the port from location added
                    walk_return = walking_back

                else:
                    break

        # When adding a new location lead to exceeding the time limit, update
        # the time spent and add the port as the last point of the collection
        time_left = time_left - walk_return
        total_time = time_limit - time_left
        collection.append(starting_location)
        cumulatives.append(round(total_time))

        # Add the finished colletion and belongin duration and cumulatives
        lst_collections.append(collection)
        lst_durations.append(int(total_time))
        lst_cumulatives.append(cumulatives)

        # Removes duplicates from each function run
        updated = remove_duplicate_collections(lst_collections, lst_durations, lst_cumulatives)

    return updated[0], updated[1], updated[2]
```

Generate the routes

```
# Route specification
times = [150, 200, 300, 400, 500, 600]
ports = ["Jekteviksterminalen", "Bontelabo cruise terminal"]
haveFood = [True, False]
num_collections = 12
```

```
lst_collections = []
lst_durations = []
lst_cumulatives = []

# Add all generated collection to the list of collections and all belonging time spent to the list of times
for time in times:

    for port in ports:

        for bol in haveFood:

            collections = generate_random_collections(port, lst_attractions, lst_food, travel_times, service_times, time, num_collections, bol)

            for collection,duration,cumulative in zip(collections[0], collections[1], collections[2]):
                lst_collections.append(collection)
                lst_durations.append(duration)
                lst_cumulatives.append(cumulative)
```

```
# Add all collections with belongig duration and cumulatice times to dictionary
dict_collections = dict()

# Remove duplicates from the total set of generated collections
no_duplicates = remove_duplicate_collections(lst_collections, lst_durations, lst_cumulatives)

# Add unique routes and belonging time to separate dictionaries
for collection,duration,cumulative in zip(no_duplicates[0], no_duplicates[1], no_duplicates[2]):
    if duration != 0:
        dict_collections["R" + str(len(dict_collections) + 1)] = {"Route": collection,
                                                                  "Duration": duration,
                                                                  "Cumulative": cumulative}
```

Using TSP to optimize the predefined routes

```python
# Function to create routes minimizing the walking duration based on
# the random collection of location using a TSP model
def optimizeRouteTSP(dict_collections):

    # Dictionary to store optimal routes
    optimal_routes = {}

    # Run a TSP model on each collection in the dictionary
    for route_id, value in dict_collections.items():

        # Initialize the start/end location for the current route
        port = value["Route"][0]

        # Set of lists with locations (random routes)
        V = value["Route"] # Set of lists with locations

        # Parameter with travel times (walking duration) for all links
        D = travel_times

        # Initialize the problem
        problem = pulp.LpProblem(f"route_TSP_{route_id}", pulp.LpMinimize)

        # Decision variables
        x = pulp.LpVariable.dicts("x",
                                  [(i, j) for i in V for j in V],
                                  cat = 'Binary')
        u = pulp.LpVariable.dicts("u",
                                  value["Route"],
                                  lowBound = 0,
                                  upBound = len(value["Route"])-1,
                                  cat = 'Continuous')

        ### Objective function ###
        problem += pulp.lpSum([D[i][j] * x[(i, j)] for i in V for j in V if i != j])
        ### Constraints ###

        # Visit each location exactly once (excluding start/end location)
        for loc in V[1:-1]:
            problem += pulp.lpSum(x[(i, loc)] for i in V[:-1] if i != loc) == 1
            problem += pulp.lpSum(x[(loc, j)] for j in V[:-1] if j != loc) == 1

        # Start and end at the port
        problem += pulp.lpSum(x[(port, j)] for j in V if j != port) == 1
        problem += pulp.lpSum(x[(i, port)] for i in V if i != port) == 1

        # Sub-tour elimination constraints
        for i in V:
            for j in V:
                if i != j and i != port and j != port:
                    problem += u[i] - u[j] + len(V) * x[(i, j)] <= len(V) - 1

        ### Solve the problem ###
        problem.solve()


        #### Add the optimized route to dictionary of the routes ####

        # Check if an optimal solution was found
        if pulp.LpStatus[problem.status] == 'Optimal':

            current_location = port

            optimised_route = [current_location]
            total_duration = 0
            cumulative_times = [0]

            # For all the locations except the port at the end of the route find the next location
            # in the route and define the travel time to the next location and the service time at the
            # next location to add to the total duration and the cumulative time for the next location
            for _ in range(len(V) - 1):

                next_location = next(j for j in V if j != current_location and pulp.value(x[current_location, j]) == 1)

                walking = D[current_location][next_location]
                service = service_times[next_location]

                total_duration += walking + service
                cumulative_times.append(round(total_duration))

                # Update the current location to be equal to the next location
                current_location = next_location

                optimised_route.append(current_location)

            # Return to start location if not already there
            if current_location != port:

                # Add the walking backt to port to the total route duration
                total_duration += D[current_location][port]

                # Add the port as the last stop of the route
                optimised_route.append(port)

            # Save the optimized route and duration in the dictionary
            optimal_routes[route_id] = {'Route': optimised_route,
                                        'Duration': total_duration,
                                        "Cumulatives": cumulative_times}

        else:
            print(f"No optimal route found for {route_id}")

    return(optimal_routes)
```

```
# Create optimized routes of the collections (minimize walking)
optimal_routes = optimizeRouteTSP(dict_collections)
```

```
# Dictionary with unique routes
dict_unique_routes = dict()

# Lists to store the list of routes, the beloning route durations
# and the belongig cumulative times for each location in the route
lst_optimized_routes = []
lst_optimized_durations = []
lst_optimized_cumulatives = []

# Add all routes, durations and cumulative times from the dictionary to the lists
for route in optimal_routes:
    lst_optimized_routes.append(optimal_routes[route]["Route"])
    lst_optimized_durations.append(optimal_routes[route]["Duration"])
    lst_optimized_cumulatives.append(optimal_routes[route]["Cumulatives"])

# Remove potential duplicates
no_duplicates = remove_duplicate_collections(lst_optimized_routes, lst_optimized_durations, lst_optimized_cumulatives)

# Add the unique routes, belonging route duration and cumulative times to the new
# updated dictionary as long as the route has a longer duration than zero
for route,route_duration,cumulatives in zip(no_duplicates[0], no_duplicates[1], no_duplicates[2]):

    if route_duration != 0:
        dict_unique_routes["R" + str(len(dict_unique_routes) + 1)] = {"Route": route,
                                                                       "Duration": route_duration,
                                                                       "Cumulatives": cumulatives}
```

```
# Saving dictionary with routes and dictionary with belonging time spent
np.save("Data/PredefinedRoutes.npy", dict_optimized_routes)
```

F Proposed model in Python

Optimization using PuLP with CBC solver

```python
# Import necessary packages
import pulp
import numpy as np
from datetime import datetime,timedelta
import random
import statistics as st
```

Initialize the problem

```python
allocation = pulp.LpProblem("OptimizationProblem", pulp.LpMaximize)
```

Load necessary data

```python
# Load the dictionary of predifed routes
routes = np.load("Data/Analysis/PredefinedRoutes.npy", allow_pickle = "TRUE").item()
```

```python
# Load the dictionary of the locations with details
locations = np.load("Data/location_details.npy", allow_pickle = "TRUE").item()

# List of the locations within the food category
food_locations = list(locations.keys())[36:]
```

```python
# Load dictionary with categories and the belonging locations
categories = np.load("Data/categories_locations.npy", allow_pickle = "TRUE").item()
```

```python
# Load dictionary with the base score for the locations
dict_scores = np.load("Data/location_scores.npy", allow_pickle = "TRUE").item()
```

```python
# Load dictionary with walking durations between the locations
dict_travel = np.load("Data/travel_times.npy", allow_pickle = "TRUE").item()
```

```python
# Load dictionary with opening hours for the locations
opening_hours = np.load("Data/OpeningHours.npy", allow_pickle = "TRUE").item()
```

```python
# Load dictionary with capacity for the locations
capacity = np.load("Data/LocationCapacity.npy", allow_pickle = "TRUE").item()
```

Define the sets

```python
# Define sets
L = locations.keys()                        # Set of locations
R = routes.keys()                           # Set of routes
P = ["P" + str(i) for i in range(5)]       # Set of tourist groups

# Set of days
D = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

# Define timeslots
hours_day = 24
timeslots = [f"{hour:02d}:{minute:02d}" for hour in range(hours_day) for minute in range(60)]
T = timeslots[480:1321]    # Set of timeslots in minutes from 08:00 to 22:00

# Set of cumulative time (continiuous integer from 0 to the max length of a route)
C = range(601)
```

Define parameters

User-input parameters

```python
# Parameter with the number of tourists in tourist group p in P
numP = dict()

# Retrieve number of tourists in each tourist group
for p in P:
    num = input("How many are you walking together? ")

    numP[p] = int(num)
```

```python
# Binary parameter which is 1 if tourist p in P arrives at Bontelabo cruise terminal,
# 0 if tourist group p in P arrives at Jekteviksterminalen
fromB = dict()

# Retrieve the arrival port for each tourist group
for p in P:
    port = input("What port do you come from? ")

    if port.lower() == "Bontelabo cruise terminal".lower():
        fromB[(p)] = 1
    else:
        fromB[(p)] = 0
```

```python
G = dict()    # Parameter with available minutes for tourist group p in P
ST = dict()   # Parameter with starting time for tourist group p in P

# Retreive data on preferred starting time and number of
# available hours for each tourist group
for p in P:
    start = input("When do you want to start? ('00:00' format)")
    hours = input("How many hours do you have? ")

    G[p] = float(hours) * 60  # Convert to minutes before adding to dictionary
    ST[p] = start
```

```python
# Binary parameter which is 1 if tourist group p in P wants to have a food location in the route, 0 otherwise
wantFood = dict()

# Binary parameter which is 1 if tourist group p in P prefers location l in L, 0 otherwise
isPref = dict()

# Set initial preferences for all tourist groups to 0
for p in P:
    for loc in L:
        isPref[(p,loc)] = 0

# For each tourist group, retrieve data on whether to include food or not,
# and if yes, retrieve data on category or specific preferences
for p in P:

    pref = input("Do you want to include a food-stop? (yes/no)  ")

    # If food is preferred, set wantFood equal to 1 for the current group
    if pref.lower() == "yes":   # Ignore lower/upper-case
        wantFood[p] = 1

        category = input("Restaurants or cafés? ")

        # If restaurants are preferred, check if any specific are preferred and if not
        # set all restaurants as preferred (isPref equal to 1 for all restaurants)
        if category == "Restaurants":

            for restaurant in categories["Food"]["Restaurants"]:
                isPref[(p,restaurant)] = 1

            spec_restaurants = input("Any specific? (Seperate by '|') ").split("|")

            for specR in spec_restaurants:

                if specR in categories["Food"]["Restaurants"]:

                    for res in categories["Food"]["Restaurants"]:
                        if res not in spec_restaurants:
                            isPref[(p,res)] = 0   # Set isPref for all not-preferred restaurants to 0

        # If cafés are preferred, check if any specifics are preferred and if not
        # set all cafés as preferred (isPref equal to 1 for all cafés)
        elif category == "Cafés":

            for cafe in categories["Food"]["Cafés"]:
                isPref[(p,cafe)] = 1

            spec_cafes = input("Any specific? (Seperate by '|') ").split("|")

            for specC in spec_cafes:

                if specC in categories["Food"]["Cafés"]:

                    for caf in categories["Food"]["Cafés"]:
                        if caf not in spec_cafes:
                            isPref[(p,caf)] = 0   # Set isPref for all not-preferred cafés to 0

        # If no specified food category set all locations within the food
        # category as preferred (isPref equal to 1)
        else:

            for subcat in categories["Food"]:
                for loc in categories["Food"][subcat]:
                    isPref[(p,loc)] = 1

    # If food is not wanted, set wantFood equal to 0 for tourist group p
    else:
        wantFood[p] = 0
```

```python
# Add preferences to parameter isPref for other categories than food

# Print the possible categories to choose from
print("Categories to choose from: ")
for cat in list(categories.keys())[1:7]:
    print(cat)

# For each tourist group, retrieve data no preferred categories and specific locations
for p in P:

    # User input for category prefences
    category_pref = input("\nWhat categories do you prefer? (Seperate by '|') ").split("|")

    for cat in category_pref:

        # If a category is preferred, set isPref for all
        # locations within the category equal to 1 initially
        if cat in categories.keys():

            # Print locations within category
            print("\n" + cat + ":")
            for loc in categories[cat]:
                print(loc)
                isPref[(p,loc)] = 1

            # User input for specific location preferences
            locations_pref = input("Any specific " + cat.lower() + " you would like to visit? (Seperate by '|') ").split("|")

            # If specific locations are preferred, set isPref for all other locations not preferred
            # in the category equal to 0 while isPref for the preferred locations remains equal to 1
            for prefloc in locations_pref:

                if prefloc in categories[cat]:

                    # When specific locations are preferred, set all other
                    # locations (if not preferred as well) equal to 0
                    for l in categories[cat]:
                        if l not in locations_pref:
                            isPref[(p,l)] = 0
```

Historical data parameters

```python
# Parameter with the route-locations (list of locations) for route r in R
routeLocations = dict()

for r in R:
    routeLocations[r] = routes[r]["Route"]
```

```python
# Parameter with the cumulative times at the locations in route r in R
locCumulatives = dict()

for r in R:
    locCumulatives[r] = routes[r]["Cumulatives"]
```

```python
# Binary parameter inRoute is one if location l in L is in route r in R, 0 otherwise
inRoute = dict()

for r in R:
    for l in L:

        if l in routeLocations[r]:
            inRoute[(r,l)] = 1
        else:
            inRoute[(r,l)] = 0
```

```python
# Binary parameter which is 1 if route r in R starts and ends at Bontelabo
# cruise terminal, 0 if it starts and ends at Jekteviksterminalen
portB = dict()

for r in R:

    if "Bontelabo cruise terminal" in routeLocations[r]:
        portB[r] = 1
    else:
        portB[r] = 0
```

```
# Binary parameter which is if route r in R includes a location
# within the food category, and 0 otherwise
haveFood = dict()

for r in R:

    # Initially set haveFood equal to 0
    haveFood[r] = 0

    for l in routeLocations[r]:

        # Checks if locations are present in the list of food locations,
        # if yes, set haveFood equal to 1 for route r
        if l in food_locations:
            haveFood[r] = 1
```

```
# Parameter with the route duration for route r in R
A = dict()

for r in R:
    A[r] = routes[r]["Duration"]
```

```
# Parameter with the capacity at location l in L
Cap = dict()

for l in L:

    Cap[l] = capacity[l]["Capacity"]
```

```
# Parameter with walking durations between locations i in L and j in L
travel = dict_travel
```

```
# Binary parameter which is 1 if location l in L is in route r in R at
# cumulative time c in C from start at port at cumulative time 0, 0 otherwise
cumul = dict()

# Initially, set all values for cumul equal to 0
for r in R:
    for loc in routeLocations[r]:
        for c in C:
            cumul[(r,loc,c)] = 0

# For each route and each location in the route, set cumul equal
# to 1 for all cumulative minutes c where the location is present
for r in R:

    # List of cumulative times for each location (time right before
    # walking to the next location)
    route_cumulatives = locCumulatives[r]

    for i, l in enumerate(routeLocations[r]):

        if i > 0:

            # Set start for moving to location l equal to the end of the previous location
            start = route_cumulatives[i-1]

            # Define the walking duration between the previous location and location l
            walk = int(travel[routeLocations[r][i-1]][routeLocations[r][i]])

            # The cumulative time for arriving at location l
            end_walking = start + walk

            # The cumulative time for location l equals the minutes from the
            # start of the route to the finished service time at location l
            cumul_time = route_cumulatives[i]

            # Set cumul equal to 1 for the timespan the location is
            # present at location l in route r
            for i in range(end_walking, cumul_time, 1):
                cumul[(r, l, i + 1)] = 1
```

```
# Binary parameter for opening hours, which is 1 if location l in L
# is open at time t in T, 0 otherwise
openT = dict()

for l in L:

    for day in D:

        for t in T:

            # If the location is not "Closed" the whole day set openT
            # at location l on day d equal to 1 for all t between
            # and including the opening time and the closing time
            if opening_hours[l][day] != "Closed":

                if t >= opening_hours[l][day]["open"] and t <= opening_hours[l][day]["close"]:
                    openT[(l,day,t)] = 1
                else:
                    openT[(l,day,t)] = 0

            else:
                openT[(l,day,t)] = 0

# For all days and times of the day set Bontelabo cruise
# terminal and Jekteviksterminalen to be open
for day in D:
    for t in T:
        openT[("Bontelabo cruise terminal",day,t)] = 1
        openT[("Jekteviksterminalen",day,t)] = 1
```

```
# Today's day
date_today = datetime.now()
today = D[date_today.weekday()]

day_today = "Thursday"   # For analysis
```

```
# Fixed parameter with the penalty for exceeding the location capacity (queue penalty)
capPenalty = 0.1
```

```
# Fixed parameter with the penalty for exceeding the maximum walking duration
walkPenalty = 5
```

```
# Fixed parameter with the maximum walking duration between two conscutive locations in minutes
walkMax = 25
```

```
# Parameter with the base score for location l in L
S = dict_scores
```

```
# Fixed parameter for preference score
prefScore = int(st.mean(dict_scores.values()))
```

```
# Parameter with the route score tourist group p in P recieves from route r in R (preferences included)
RouteScore = dict()

for p in P:

    for r in R:

        # Initial score for tourist group p for route r equal to zero
        score = 0

        # If tourist group p wants food and route r includes food add the the base score for
        # each location in route r and preference scores if relevant to "score", and the same
        # opposite if tourist group p do not want food and route r do not include food
        if haveFood[r] == wantFood[p]:

            # Add the base score for all locations in route r, and if route r consists of
            # locations preferred by tourist group p, add the preference score "prefScore"
            for l in routeLocations[r]:
                score += S[loc]

                if isPref[(p,l)] == 1:
                    score += prefScore
        else:

            # If route r consists of locations preferred by tourist group p, add 1 to the score
            # for each preferred location present in route r to distinguish between routes including
            # preferred locations relative to those not when the food preference is not fullfilled
            for loc in routeLocations[r]:

                if isPref[(p,l)] == 1:
                    score += 1

        # Add the score route r provides tourist group p to the dictionary
        RouteScore[(p,r)] = score
```

Decision variables

```
# Binary variable which is 1 if tourist group p in P is assigned to route r in R, 0 otherwise
x = pulp.LpVariable.dicts("x", [(p,r,ST[p]) for p in P for r in R], 0, 1, pulp.LpBinary)
```

```
# Binary variable which is 1 if tourist group p in P is present at location l in L during timeslot t in T, 0 otherwise
y = pulp.LpVariable.dicts("y", [(p,r,l,t) for p in P for r in R for l in routeLocations[r] for t in T], 0, 1, pulp.LpBinary)
```

```
# Integer variable larger than zero representing capacity slack
z = pulp.LpVariable.dicts("z", [(l,t) for l in L for t in T], lowBound = 0, cat = "Integer")
```

```
# Continuous variable larger than zero representing walking slack
q = pulp.LpVariable.dicts("q", [(p) for p in P], lowBound = 0)
```

Objective function

```
# Maximize the sum of scores generated from tourist groups assigned to routes subtracting the penalty for exceeding maximum walking between consecutive locations and capacity at locations
allocation += pulp.lpSum([RouteScore[(p,r)] * x[(p,r,ST[p])] for p in P for r in R]) - (pulp.lpSum(q[p] for p in P) * walkPenalty) - (pulp.lpSum(z[(l,t)] for l in L for t in T) * capPenalty)
```

Constraints

```
# Each tourist group must be assigned to one and only one route
for p in P:
    allocation += pulp.lpSum([x[(p,r,ST[p])] for r in R]) == 1
```

```
# Each tourist group must be assigned to a route based on the port they come from
for p in P:
    allocation += pulp.lpSum([x[(p,r,ST[p])] * portB[r] for r in R]) == fromB[p]
```

```
# Each tourist group must be assigned to a route whitin their time available
for p in P:
    allocation += pulp.lpSum([A[(r)] * x[(p,r,ST[p])] for r in R]) <= G[p]
```

```
# Capacity constraint
for l in L:
    for t in T:
        # The number of tourists at location l must be within the capacity at location l, otherwise slack variable z is activated
        # If location l is closed at time t, the capacity equals 0 and no slack is possible
        allocation += pulp.lpSum([y[(p,r,l,t)] * numP[p] for p in P for r in R if l in routeLocations[r]]) <= openT[(l,today,t)] * (Cap[l] + z[(l,t)])
```

```
# Walking constraint
for p in P:
    for r in R:
        for i in range(len(routeLocations[r]) - 1):

            current = routeLocations[r][i]
            next_loc = routeLocations[r][i+1]

            # The travel time between two consecutive locations must be within
            # 25 minutes, otherwise slack variable q is activated
            allocation += x[(p,r,ST[p])] * travel[current][next_loc] <= 25 + q[p]
```

```
# Logical relationship between x and y
for p in P:
    start_time = datetime.strptime(ST[p], "%H:%M")

    for r in R:

        for i, l in enumerate(routeLocations[r][1:]):

            if i + 1 < len(routeLocations[r]):

                # Start time for location
                loc_start_time = start_time + timedelta(minutes = locCumulatives[r][i] + int(travel[routeLocations[r][i]][routeLocations[r][i+1]]))

                # End time for location
                loc_end_time = start_time + timedelta(minutes = locCumulatives[r][i+1])

                for t in T:
                    t_datetime = datetime.strptime(t, "%H:%M")

                    if loc_start_time < t_datetime <= loc_end_time:

                        # Cumulative time from start time for tourist group p to time t
                        calcTime = int((t_datetime - start_time).total_seconds() / 60)

                        # y must equal 1 if tourist group p is assigned to
                        # route r and is present at location l at time t
                        allocation += y[(p, r, l, t)] <= x[(p, r, ST[p])] * cumul[(r, l, calcTime)]
                        allocation += y[(p, r, l, t)] >= x[(p, r,  ST[p])] * cumul[(r, l, calcTime)] - (1 - x[(p, r,  ST[p])])
```

Solve problem

```
allocation.solve()
```

```
# Print status of the solution
print("Status: ", pulp.LpStatus[allocation.status])
```

```
# Print the solution
print("Objective: " + str(round(pulp.value(allocation.objective),2)) + "\n")
print("Allocation of the tourist groups: ")

for v in allocation.variables():

    if v.varValue != 0:
        print(v.name, "=", v.varValue)
```