University of
Southern Denmark

Scheduling Audits at Danish Gambling Facilities to Ensure Regulatory Compliance

Christoffer Mondrup Kramer
April 1, 2024

Cand.Scient Thesis in Data Science
Department of Mathematics and Computer Science
University of Southern Denmark

University Supervisor: Marco Chiarandini
Organization Supervisor: Jens Rasmussen

Spillemyndigheden er en myndighed i skatteministeriet, der regulerer det danske spillemarked. I forbindelse med deres reguleringsarbejde udfører spillemyndigheden regelmæssige tilsyn med alle spillesteder i Danmark. Som en del af myndighedens mål og resultats plan for 2023 vil organisationen inkorporere dataanalyse i planlægningsfasen for tilsyn. Derfor præsenterer denne specialeafhandling *The Auditor Scheduling and Routing Problem* (ASRP). For at forstå problemet benyttes kvalitative metoder i form af semistrukturerede interviews og observation. Databasedesign bruges til at udforme en database, som analyseres ved hjælp af eksplorative datavisualiseringer. Problemet er repræsenteret som en matematisk model, der er beskrevet ved hjælp af lineær og heltalsprogrammering. For at teste den matematiske model benyttes en *discrete-event simulation model*, der simulerer hver dag i 2022. Simuleringsmodellen introducerer to parametre (ϕ og ϵ), som kan bruges til at teste den matematiske model med forskellige datadistributioner. De første resultater fra simuleringsmodellen indikerer, at den matematiske formulering virker efter hensigten, men designer ruter, der er ineffektive med den simulerede datadistribution. Som videre arbejde med projektet præsenteres en række effektivitetsmål, der kan bruges til at sammenligne forskellige formuleringer af ASRP og forskellige datadistributioner.

Contents

List of Figures

List of Tables

Chapter 1

Acronyms

ASRP	Auditor Scheduling and Routing Problem
ADFJ	Aggregated Dantzig-Fulkerson-Johnson
BAG	The German Federal Office for Goods
BnB	Branch and Bound
CVRP	Capacitated Vehicle Routing Problem
DFJ	Dantzig-Fulkerson-Johnson
DGA	Danish Gambling Authority
ER-diagram	Entity-Relationship diagram
ILP	Integer Linear Programming
LP	Linear Programming
OR	Operations Research
MILP	Mixed-Integer Linear Programming
MPVRPD	Multi-Period Vehicle Routing Problem with Due dates
MSP	Machine Scheduling Problem
MTZ	Miller-Tucker-Zemlin
MVRP	Multi-depot Vehicle Routing Problem
PMSP	Parallel Machine Scheduling Problem
SEC	Subtour Elimination Constraint

SMSP	Single Machine Scheduling Problem
TEP	Toll Enforcement Problem
TISP	Technicians and Interventions Scheduling Problem
TSP	Traveling Salesman Problem
TSRP	Technician Scheduling and Routing Problem
VRP	Vehicle Routing Problem
VRP-RD	Vehicle Routing Problem with Release and Due dates

Chapter 2

Introduction

The Danish Gambling Authority (DGA) is an organization within the Danish Ministry of Taxation. Their main responsibility is to regulate the Danish gambling market and to ensure that both gamblers and gambling operators participate in a fair and well-regulated gambling market [46]. In practice, the organization has a wide range of duties, which includes, but is not limited to, auditing regulatory compliance at gambling establishments [46]. All facilities that provide any kind of on-site gambling products are considered gambling establishments. Hence, gambling premises are not only casinos and gambling halls, but also grocery stores, kiosks, bars, restaurants, and even local associations. When assessing regulatory compliance, DGA auditors will perform an inspection of the data provided by the facility and a physical audit of the premises [46]. When planning these audits the DGA rely on risk assessments where high-risk facilities receive more audits and scrutiny. The DGA's current approach to auditing relies heavily on manual planning, and they schedule the audits based on heuristics. This is in contrast to their future goals, which are stated in a 2023 public strategy report:

> "Our intention is to be able to measure the effectiveness of audits, through the increasing use of data and risk assessments in the planning phase" [47][1]

To aid the DGA in their goal of further incorporating data in the planning phase this thesis will suggest a revised auditing approach, which incorporates mathematical scheduling and route optimization. To do this, the *Auditor Scheduling and Routing Problem* (ASRP) is presented. The ASRP is a resource constrained routing and scheduling problem, which seeks to schedule as many audits as possible, by using a rolling time-horizon.

The ASRP was developed with qualitative methods, database design, exploratory data visualizations, simulation modeling, and Integer Linear Programming (ILP). Qualitative methods are used to structure and understand the problem, and exploratory data visualizations provide a preliminary analysis of the problem and the

[1]P. 11 - Translated from Danish

results. The data is compiled from various sources to one central database. This database is used to retrieve the input for a *discrete-event simulation model*. This simulation model is used to test the ILP formulation of the ASRP.

This thesis is only the first step towards a more data-driven auditing approach at the DGA. Therefore, the results are intended to be preliminary results, which can serve as the basis for further work on the project. Therefore, the thesis will present how the ASRP was developed and solved, but also provide a thorough discussion of future work on the project.

2.1 Problem Statement

This master thesis will address the following problem statement:

> ”How can the Danish Gambling Authority optimize in accordance with their regulatory strategy and goals when scheduling inspections at Danish gambling facilities?”

The intention of this problem statement is to, first and foremost, focus on the practical applications of the thesis. However, since the DGA is a regulatory agency, their aim cannot be assumed to be the same as that of a private profit-driven organization. Therefore, the goals and regulatory strategy of the DGA is included in the problem statement. This ensures that the unique organizational challenges and goals of the DGA are considered throughout this thesis.

2.2 Code and Data Sharing

Due to a non-disclosure agreement with the DGA, it is not possible to share the data used in this report. However, small excerpts of the data are provided in Chapter 6, but all sensitive information has been pseudonymized. All code, which is not related to the creation of the database, is shared under the open-source MIT license on the following GitHub repositories:

- `https://github.com/Chris-Kramer/Christoffer_Master_Thesis`
- `https://git.imada.sdu.dk/chkra21/christoffer_kramer_master_thesis`

At these repositories the output from the simulation model and interactive versions of the data visualizations are also provided. Note that the latitude and longitude columns from the output of the simulation model have been removed. Lastly, all code has been provided as Appendices in the thesis.

2.3 Report Structure

This thesis is divided into chapters. Chapter 3 outlines the methods used in this thesis. Chapter 4 provides a detailed overview of mathematical scheduling and routing.

This chapter also introduces the ASRP; the mathematical notation; and provides a review of similar problems. Chapter 5 describes the problem; how the DGA schedules audits; and details how the problem can be represented mathematically. Chapter 6 outlines how the data was collected, pre-processed, and structured. It also provides a description of all tables in the relational database. Chapter 7 provides a preliminary analysis of the data, the problem, and the auditors' scheduling and routing preferences. Chapter 8 presents the ILP formulation of the ASRP and presents alternative formulations. Chapter 9 introduces the simulation model and describes the input and output data. This chapter also introduces the hyper-parameters ϕ and ϵ and describes how these parameters affect the data distribution in the simulation model. Chapter 10 displays and discusses the preliminary results, while Chapter 11 presents and discusses potential evaluation metrics for the results. Lastly, Chapter 12 discusses future work on the project and provides a conclusion.

Chapter 3

Methodology

This thesis uses a wide range of methodologies that span multiple disciplines. However, most of the methods are related to *Operations research* (OR). This is a multidisciplinary field that uses mathematical techniques to aid in the decision-making process. This is usually done by using mathematical modeling, statistical analysis, data visualizations, and mathematical optimization to analyze and solve a problem [33]. Section 3.1 describes qualitative methods, also known as *soft OR* [37], while Section 3.2 introduces the concept of relational databases. Section 3.3 describes how data visualizations can be used for data analysis and exploration. Section 3.4 outlines the concept of linear programming problems and details how these problems can be solved with the simplex algorithm and the theory of duality. Section 3.5 demonstrates how linear programming problems can be extended to integer linear programming problems and how these problems can be solved with the branch and bounds method, heuristics, and cutting-plane algorithms. Lastly, Section 3.6 describes the purposes of simulation modeling.

3.1 Qualitative Methods

To properly understand how the DGA functions as an institution and how the auditors perform and schedule audits, qualitative methods were employed. There are various qualitative methods, most of which have an epistemological background within anthropology and ethnography. These fields seek to understand cultural phenomena by describing, observing, and participating in concrete activities related to the studied phenomena [42]. This thesis relied mainly on *semi-structured interviews* and *observation.*

Semi-structured interviews are characterized by the fact that it is the interviewer who structures the interview. This is done by having questions and/or themes to guide the interview [43]. This methodology is especially useful if the interviewer is studying narrow topics and questions [43]. If a semi-structured interview is too tightly structured, it can end up resembling an oral questionnaire [43]. Therefore, the semi-structured interview must be structured tightly enough to answer relevant research questions, while also being loose enough for the interviewee to come up with new and unexpected insights [43]. In this project, semi-structured interviews were conducted with auditors, managers, and project managers, and they form the basis of Chapter 5, which provides a detailed description of the problem and discusses how the problem can be represented mathematically. Moreover, to store the data in a standardized format, a relational database was created, which is described in Chapter 6. The schema of the database was created after having done extensive interviews with auditors and managers at the DGA.

Observation is characterized by paying special attention to a specific phenomenon or activity [27]. It is a common misconception that observation simply refers to "watching", and it should be noted that the point of observation is not to act as a human video recorder or a silent fly on the wall [27]. Observation must instead be thought of as an active method where the observer curiously, patiently, and carefully investigates how and why an activity is performed [27]. Therefore, this method requires that the observer actively asks questions, takes extensive notes, and plan what to observe and why [27]. This project mainly used observation for Section 5.5, where the observation of different audit types was used to determine how to calculate the duration of audits.

3.2 Database Design

For this thesis, a relational database was created to store the data. In 1970 the computer scientists Ted Codd introduced the concept of relational databases [53]. In a relational database, the data is organized in tables that represent relations between elements of different sets [8]. Each row in a relation is a tuple which can be manipulated using *relational algebra* [8]. This allows the user to perform operations, such as joining tables, selecting specific rows, and updating the data [53]. Most modern database systems use a *structured query language* (SQL) to incorporate relational algebra and perform operations on the data [53].

Each column in a relational table is an attribute and the schema of a relational database is the set of all relations (tables) in a database [53]. What makes a relational database an effective tool for ensuring data integrity is the ability to place constraints on relations [53]. The most fundamental of these constraints being *keys*. A key acts as a unique identifier for a tuple in a relation [53]. This allows the user to always identify a tuple and to create references between tables, which form relationships between the tables [53].

This ability to create relationships between tables is the basis for *normalizing* the data. There are different types of normal forms; however, the most common normalization is the *third normal form* [53]. Simply described, data is in third normal form if all attributes only provide information about the primary key(s) of the table [9]. By normalizing the data, the referential integrity of the data is ensured, since unnecessary redundancies and duplicate information is eliminated [53].

3.3 Data Visualizations

Data visualization is the act of drawing graphical displays to show and present data and is considered one of the most fundamental methods within data science [49]. Data visualizations can help enhance a narrative [22]; help in the exploratory data analysis process [49]; or make abstract information and data patterns comprehensible [45]. The principles of *graphical excellence*, from the analytical design theorist Edward Tufte, form the basis for how visualizations are utilized in this thesis. Graphical excellence is a set of nine principles which, according to Tufte, should apply to all graphics [48]:

- Graphics must show the data [48].
- Graphics must induce the viewer to think about substance rather than graphic design or methodology [48].
- Graphics must avoid distorting the data [48].
- Graphics must present many numbers in a small space [48].
- Graphics must make large data sets coherent [48].
- Graphics must encourage the eye to compare different pieces of data [48].
- Graphics must reveal the data at several levels of detail, from a broad overview to fine structure [48].
- Graphics must serve a clear purpose: description, exploration, tabulation, or decoration [48].
- Graphics must be closely integrated with the statistical and verbal descriptions of a data set [48].

All visualizations in this thesis follow these principles by keeping the visualizations simple and on-point. Moreover, all visualizations have detailed descriptions attached, which clearly explains the illustration and the presented data.

According to Antony Unwin, visualizations can be divided into either *exploratory* or *presentation* visualizations [49]. Exploratory visualizations are used to find new information, while presentation visualizations are intended to convey known information and results [49]. In this report, the visualizations presented in Chapter 5, 6, 9, 8, and 10 are intended to explain work routines; theoretical concepts; results; and the structure of the data, therefore, these illustrations are presentation visualizations. Meanwhile, the visualizations presented in Chapter 7 are intended to provide an exploratory analysis of the data and the problem at hand. Therefore, these illustrations are used as exploratory visualizations.

3.4 Linear Programming

The essential task of *linear programming* (LP) is to solve a system of inequalities of the form $Ax \leq b$ where A is a $m \times n$ matrix, x is a $n \times 1$ matrix, and b is a $m \times 1$ matrix [7]. These inequalities are referred to as *constraints* and they form a convex polyhedron [7]. The area within this polyhedron is known as the *feasible region*, since all feasible solutions lie within this region [50]. A simple example of a linear program and the feasible region is illustrated in Figure 3.1.

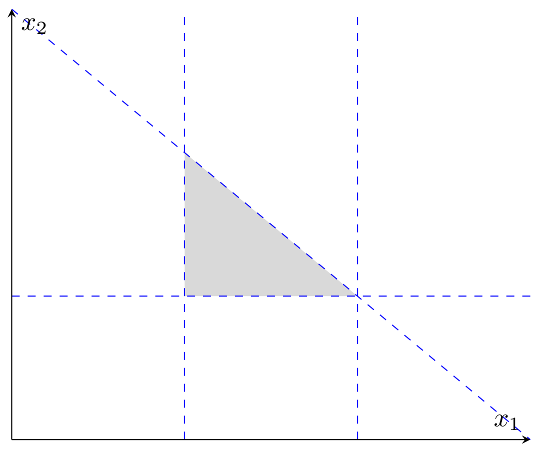


Figure 3.1: *How a LP problem forms a convex polyhedron.*

In Figure 3.1 the feasible region is colored gray and the constraints are represented as the dotted lines. The aim of LP is to find a solution to this system of inequalities such that an objective function is maximized or minimized. This function is generally denoted by $max/min\ c^T x$, where c is a $n \times 1$ matrix containing coefficients [7]. By using linear programming, two fundamental problems are addressed:

- The *solvability* of a problem. That is, can the system of linear inequalities be solved. Geometrically, this means checking if the convex polyhedron is non-empty [7].
- The *optimality* of a problem. That is, given a problem that can be solved, what is the optimal solution to the system of linear inequalities with respect to the objective function [7].

Linear programming can be traced back to 1827 when the mathematician Joseph Fourier formulated the first linear programming problem by solving a system of linear inequalities; however, it was not until the 1940s that it developed as a formal discipline [7]. This was a direct result of the second world war, where efficient resource allocation was needed for large-scale military planning [7].

3.4.1 Simplex Method

One of the most widely used algorithms to solve LP problems is the simplex method, which was proposed in 1947 by George B. Dantzig [7]. The simplex method utilizes the fact that the feasibility region of a linear program $\mathcal{K} = \{x \in \mathbb{R}^n : Ax \leq b, x \geq 0\}$ always forms a pointed convex polyhedron [7]. Therefore, if $\mathcal{K} \neq \emptyset$ then there will always be at least one extreme point which is referred to as a *basic feasible solution* or a vertex [50] [7]. An example of the basic feasible solutions to a simple LP problem can be seen on Figure 3.2.

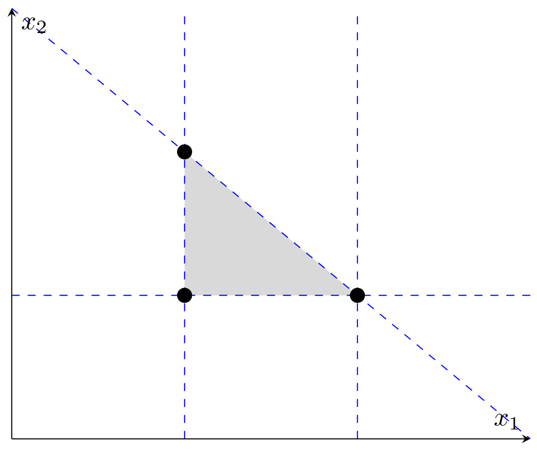


Figure 3.2: *The basic feasible solutions to a LP problem.*

The fundamental theorem of linear programming states that for any LP problem with a solution the optimal solution will be a basic feasible solution [50]. The simplex method utilizes the geometric shape of an LP problem and the fundamental theorem of linear programming by moving between vertices until it reaches the optimal solution [7]. A naive approach would be to check each vertex and then pick the vertex that maximizes / minimizes the objective function, as illustrated on Figure 3.3.

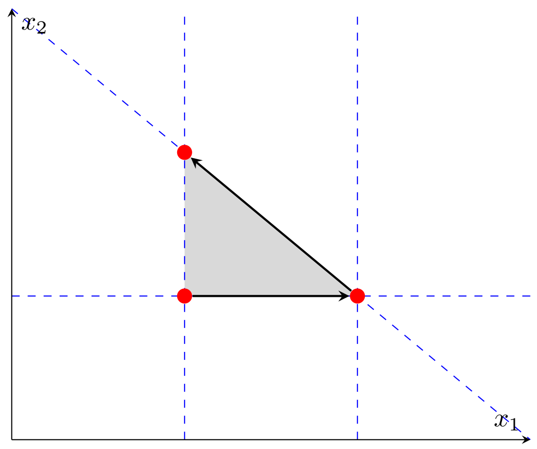


Figure 3.3: *A naive approach for solving LP problems.*

Figure 3.3 demonstrates a naive approach where all vertices are checked. However, this method is inefficient and would guarantee a complexity of 2^n [20]. Instead, the simplex method starts at a vertex and identifies neighboring vertices [50]. This is displayed on Figure 3.4.

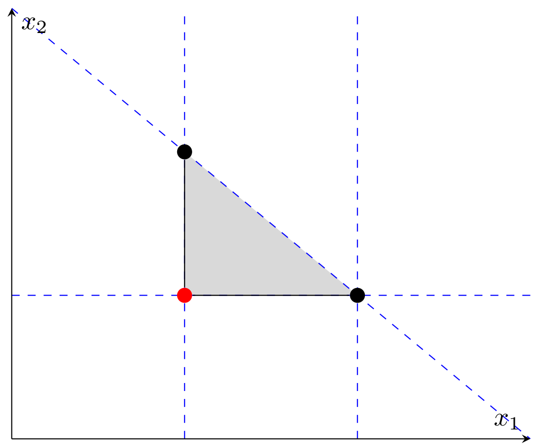


Figure 3.4: *How neighbouring vertices are identified.*

As Figure 3.4 displays, the neighboring vertices of the red vertex are identified. The algorithm then pivots to the vertex which provides the best solution [50], which is illustrated on Figure 3.5.

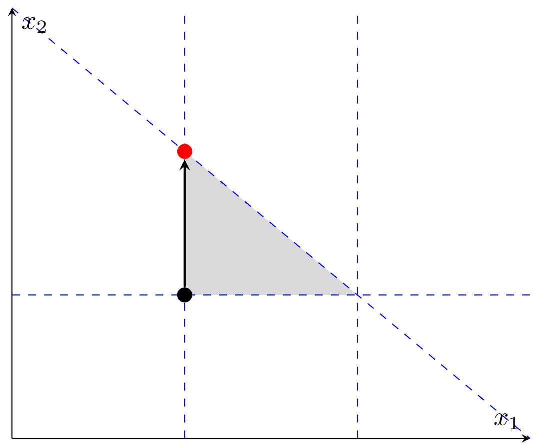


Figure 3.5: *How the simplex algorithm pivots to the vertex with the best solution.*

Figure 3.5 illustrates a maximization problem. In these types of problems, the simplex algorithm pivots to the vertex, which maximizes the objective function. This process is repeated until a local maxima / minima is reached and there are no neighboring vertices with a better solution [50].

Although the simplex method, in the worst case, has to visit all vertices, in practice its runtime tends to be substantially faster than 2^n [20]. Moreover, by using advanced rules to select where to pivot, known as *pivot rules*, even large instances of LP problems can be solved in a reasonable time [20].

3.4.2 Theory of Duality

In 1947, during a meeting between Dantzig and Jon Von Neumann, the theory of duality was conjectured [7]. The theory of duality states that for any LP problem, which is called the *primal* problem, a related *dual* problem can be derived [51]. The dual problem is essentially the same problem but from a different perspective, since any problem that seeks to maximize a function $\mathcal{Z}$, also has a related minimization problem that seeks to find the minimum value of a function $\mathcal{W}$ that bounds the primal problem [51]. Simply put, while the primal problem seeks to maximize $\mathcal{Z}$, until it reaches the optimal value, the dual problem seeks to minimize $\mathcal{W}$, until it reaches the upper bound of $\mathcal{Z}$ [51]. The relation between the primal and dual problem is visualized on Figure 3.6.

$$\begin{tikzcd}
\boxed{\textbf{Dual Problem}} \arrow[d, "\text{Minimize } \mathcal{W}"] \\
\boxed{\textbf{Optimal value}} \\
\boxed{\textbf{Primal Problem}} \arrow[u, "\text{Maximize } \mathcal{Z}"']
\end{tikzcd}$$

Figure 3.6: *How the dual problem bounds the primal problem.*

As Figure 3.6 demonstrates, the upper bounds of LP maximization problem represents the lower bound of the dual minimization problem. The dual problem have a range of interesting properties one of the most important being, that if the dual problem has a finite solution, the primal problem likewise has a finite solution such that the value of $\mathcal{Z}$ is equal to the value of $\mathcal{W}$ [51]. This is known as the *strong duality theorem* and it means, that from the optimal solution to the dual problem, the optimal solution to the primal problem can be derived and vice versa [51]. Therefore, it can sometimes be more computationally efficient to solve the dual problem, which might be simpler, and then derive the solution to the primal problem [51] [7].

3.5 Integer Linear Programming

For many real-world problems, it is required that the decision variables be integer values [52]. In *Mixed-Integer Linear Programming* (MILP) problems only some of the decision variables must be an integer value, while in pure *Integer Linear Programming* (ILP) problems all decision variables must be integer values [52]. Both MILP and ILP problems can model a wide range of real-world problems; however, they are also considerably harder to solve than classic LP problems [52].

3.5.1 Branch and Bounds

A common method to solve ILP and MILP problems is the *Branch and Bound* (BnB) method. In the BnB method, a relaxed version of the original problem is solved [54]. This relaxed problem is an LP problem where the integer constraints are removed [54]. If the integer constrained variables, in the relaxed problem, contain fractional values, the feasibility region S is partitioned into binary sets S_k where $S_1 = S \cap \{x : x_j \leq \lfloor x_j \rfloor\}$ and $S_2 = S \cap \{x : x_j \geq \lceil x_j \rceil\}$ [54]. This creates a partition tree with two new LP problems that must be solved [54]. Geometrically, the feasibility region is partitioned into smaller subproblems, as shown in Figure 3.7.

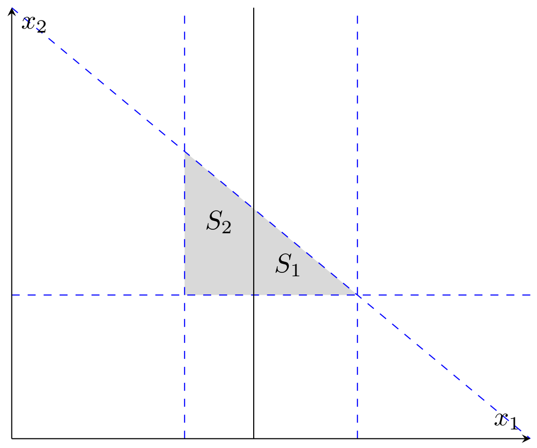


Figure 3.7: *How the BnB algorithm partitions the feasibility region.*

The aim of the BnB method is to find the subproblem, S_k, with the highest integer solution [54]. If S_k contains a fractional solution, a new branch of subproblems is created and the value of the optimal solution for S_k, which is denoted as $\mathcal{Z}_k$, is stored [54]. This means that the feasible region is repeatedly partitioned into smaller subproblems until the subproblem, which contains an optimal integer solution, is found [54]. This process is shown in Figure 3.8.

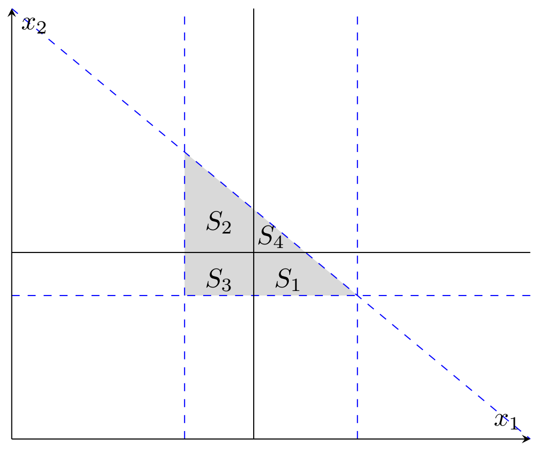


Figure 3.8: *How the BnB algorithm repeatedly partitions the feasibility region.*

In order to minimize the number of partitions, a pruning process is employed [54]. If one of the subproblems has a lower value of $\mathcal{Z}_k$, than the best known integer solution, the whole subproblem can be pruned [54]. When a subproblem is pruned no further partitions, within this subproblem, are created, which is known as *pruning by bound* [54]. Furthermore, if a solution is infeasible, the subproblem is likewise not partitioned any further, which is known as *pruning by infeasibility* [54]. Lastly, if no subproblems contain a higher value of $\mathcal{Z}_k$ than the current integer solution, no further partitions are created since the optimal integer solution has been found [54]. This is known as *pruning by optimality* [54].

3.5.2 Cutting-Plane Algorithms

Another method for solving ILP and MILP problems are *cutting-plane algorithms*. Like the BnB method, cutting-plane algorithms also relax the integer constraints and add cuts to the feasible region until the best integer solution is found [10]. The primary difference between a cutting-plane algorithm and the BnB method lies in how the cuts are generated and added to the problem. Where the BnB divides the feasible region into binary partitions, a cutting-plane algorithm repeatedly cuts off the fractional part of the feasible region [10]. This process is shown in Figure 3.9.

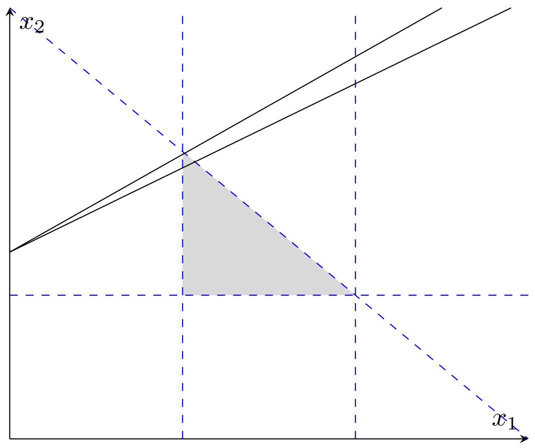


Figure 3.9: *How a cutting-plane algorithm repeatedly cuts off the fractional part of the feasibility region.*

The most famous type of cut is the *Gomory fractional* cut, which can be obtained from a solved simplex tableau by selecting row i, where b_i is a non-integer value, and then applying the following Formula [10]:

$$\sum_{j}^{n}(a_{ij} - \lfloor a_{ij} \rfloor)x_j \geq b_i - \lfloor b_i \rfloor$$

When the cut is added the problem is resolved and, if the solution still contains non-integer values, a new cut is generated and added [10]. This process is repeated until the solution only contains integer values [10].

3.5.3 Solvers and Heuristics

Rather than directly implementing the methods described above, most LP, MILP, and ILP, problems are solved using a *solver*, which is mathematical optimization software that finds the solution to a mathematical problem. Most modern solvers incorporate a mixture of the BnB method, cutting-plane algorithms, duality theory, and the simplex method [10]. In some cases, the problem can prove too challenging for even modern commercial solvers to solve [23]. In these cases, a common approach is to use *heuristics*, which find near optimal solutions to large-scale problems [23]. Another approach is to decompose the problem into simpler subproblems that can be solved optimally. When dealing with time-horizons this can be done by taking advantage of the underlying time structure to decompose the problem [17]. This strategy is referred to as a *rolling time-window* and is the method used in this report, which is outlined in Section 7.8.

3.6 Simulation Modeling

To test the performance of the ILP formulation of the ASRP a *simulation*, which simulated the year 2022, was used. A simulation imitates a *system*, which is a collection of entities that interact towards some common logical end [24]. The aim of a mathematical model is to represent this system, such that information about the system can be derived [24]. It may be possible to derive exact information through the use of mathematical methods, which is known as the *analytical solution* [24]. However, in most cases, the system and the mathematical model is too complex for these methods to be applicable [24]. Therefore, a simulation must be used to test how certain inputs affect the output of the mathematical model [24]. The combination of a simulation and a mathematical model is referred to as a *simulation model* [24].

Simulation models can be categorized into *discrete* and *continues* simulation models [24]. In a continuous simulation model, the state variables change continuously [24]. An example of this could be an airplane that flies through the air since its velocity and position continuously change [24]. In a discrete simulation model, the state variables change instantly at discrete points in time [24]. The number of customers in a store on a given day is an example of a discrete simulation model [24].

Simulation models can also be categorized into *static* and *dynamic* simulation models [24]. Static models are used to represent a system at a singular point in time or to represent systems where time is not a factor, such as coin flips or dice rolls [24]. Dynamic simulation models are used to represent systems that change over time [24]. An example of this is a conveyor belt system in which a factory's products are steadily assembled over time [24].

Lastly, simulation models can be categorized into *deterministic* and *stochastic* simulation models [24]. In a deterministic simulation model, there are no random components [24]. Therefore, the output is *determined* and can, theoretically, be computed manually [24]. A stochastic simulation model contains random components; therefore, its output will always contain some elements of randomness and should, therefore, be treated as an estimation of the system [24].

In this report, a discrete, dynamic, and deterministic simulation model was used. Since it simulates each day in the year 2022, it belongs to a class of simulation models named *discrete-event simulation models*. These types of simulation models are used to represent systems that change over time and they simulate discrete events that instantly change the state variables at each time step [24].

There exists a wide range of simulation software [25]. However, for this report, the simulation model was implemented directly in Python without any third-party packages dedicated to simulation modeling. There are many reasons for this. First and foremost, the simulation model is relatively simple, and it was, therefore, not necessary to have any advances features beyond Python's capabilities. Moreover, since the rest of the project was built in Python, it was more simple to integrate the data and mathematical model with a simulation model built in Python compared to using dedicated simulation software.

3.7 Conclusion

Semi-structured interviews were used to understand the problem and how it could be represented mathematically. Semi-structured interviews are characterized by the fact that they are structured by the interviewer. However, they cannot be so tightly structured that they resemble an oral questionnaire. Moreover, they were also used to create the relational database schema. Observation refers to the method of actively observing how an activity is performed, this method was used to determine the formula for calculating the duration of audits.

A relational database was used to store the data in a central location. A relational database represents relationships between tuples. These relationships are constrained by keys, which forms the basis of normalizing the data. When data is normalized in the third form, all attributes in a table provide information about the primary key(s) of that table. This eliminates redundancies and ensures referential integrity. Another useful feature of relational databases is the ability to perform relational algebraic operations on the data, making the data easy to query and manipulate.

All data visualizations in this report have been designed to follow Tufte's principles of graphical excellence. These principles ensure that the data visualizations are easily comprehensible and seek to avoid distorting the data. Data visualizations can be categorized into exploratory visualizations and presentation visualizations. The exploratory visualizations are intended to aid in the analysis of the problem, while the presentation visualizations present the results of the thesis.

This thesis uses Linear Programming (LP) to represent the ASRP. LP problems seek to solve a system of linear inequalities. LP addresses two fundamental problems in a system of linear inequalities: the solvability of a system, which involves checking if a convex polyhedron is non-empty; and the optimality of a problem, which means finding the optimal solution to the system of linear inequalities with respect to an objective function. One of the most important algorithms for solving LP problems is the simplex method, which pivots to neighboring vertices until it reaches a local minimum / maximum. The simplex method can be combined with the theory of duality when solving LP problems. The theory of duality states that for every primal LP problem, a related dual LP problem can be derived. If the primal LP problem seeks to maximize an objective function until it reaches its upper bound, the dual problem seeks to minimize an objective function until it reaches the upper bounds of the primal problem. From the strong duality theorem it is possible to derive the solution to the primal problem from the solution to the dual problem. Therefore, it can sometimes be more computationally efficient to solve the dual problem and then derive the solution to the primal problem.

LP problems can be extended to Integer Linear Programming (ILP) and Mixed-Integer Linear Programming (MILP) problems. In these problems, some or all of the decision variables must take an integer value. These problems can represent a wide range of real-world problems, but they are substantially harder to solve. The most common approach is to use either the Branch and Bound (BnB) method or cutting-plane algorithms. The BnB method relaxes the integer constraints and then repeatedly partitions the feasible region into smaller subproblems until it finds the optimal integer solution. To limit the number of partitions, a pruning procedure, where subproblems are removed, is employed. Subproblems can be pruned by optimality, bound, and infeasibility. Cutting-plane algorithms also relax the integer constraints and apply a cut to the feasible region. However, unlike the BnB method, which creates binary partitions, cutting-plane algorithms repeatedly cut off the fractional value from the feasible region until the optimal integer solution is found.

Rather than manually implementing these algorithms, a solver, which is mathematical optimization software, is usually used to solve LP, ILP, and MILP problems. However, some problems can be too difficult for even modern solvers to solve. In these cases, it is necessary to either use heuristics, which finds the near-optimal solution, or to decompose the problem into smaller subproblems, which can be solved to optimality. In this report, the second option was selected by using a rolling time-horizon.

To test the ILP formulation of the ASRP a simulation model, which is a method that mimics a system of real-world processes, was employed. A simulation model is useful when modeling complex systems where the analytical solution is unattainable. The aim of using a simulation model is to understand a mathematical model by studying what effect certain inputs have on the output of the mathematical model. A simulation model can be categorized into many different types, but the one used in this report is a discrete, dynamic, and deterministic simulation model. Discrete refers to the fact that the state variables change instantaneously; dynamic denotes the fact that it models a system which changes over time. Lastly, deterministic represents the fact that there is no element of randomness in the model. The simulation model in this thesis belongs to a class of models called discrete event simulation models, which simulate discrete events where the state variables change instantly at each time step. It was modeled using Python without any third-party packages dedicated to simulation modeling.

Chapter 4

Mathematical Scheduling and Routing

This chapter presents the theory behind mathematical scheduling and routing. Section 4.1 outlines scheduling problems, while Section 4.2 describes the Traveling Salesman Problem (TSP). Section 4.2.1 details the VRP and how this problem can be extended to a wide variety of problems called rich VRPs. These problems can incorporate multiple depots, time-horizons, vehicle capacities, and a heterogeneous fleet. Section 4.2.2 describes how a new class of VRPs, which deals with resource-constrained routing and scheduling, has emerged. It then presents the Technician Scheduling and Routing Problem (TSRP) which seeks to schedule technicians for requests. Section 4.3 outlines the Auditor Scheduling and Routing Problem (ASRP) and introduces the mathematical notation used in this thesis. Lastly, Section 4.4 provides a literature review by comparing the ASRP with the Toll Enforcement Problem (TEP); The Technicians and Interventions Scheduling Problem (TISP); and The Multi-Period Vehicle Routing Problem with Due dates (MPVRPD).

4.1 Scheduling Problems

Scheduling is concerned with the allocation of resources to tasks over a given time period [40]. In its most basic form, a scheduling problem can be defined as a set of J jobs that must be processed by M machines [28]. Each job $j \in J$ has a processing time of p_j and must be processed on one of the machines [28]. The goal is to find a schedule that optimizes a performance measure [28]. These classes of scheduling problems are called *Machine scheduling problems* (MSP). If there is only one machine, the problem is referred to as a *Single Machine Scheduling Problem* (SMSP) [28]. In the SMSP the only decision is the *sequencing* of the jobs, which refers to the order the jobs are performed [28]. If there are multiple machines, which is known as a *Parallel Machine Scheduling Problem* (PMSP), both the sequencing and the *assignment* must be decided upon [28]. The assignment refers to what jobs are allocated to which machines [28].

MSPs can be extended to include release and due dates where no job $j \in J$ can be performed before r_j and all jobs $j \in J$ must be performed before d_j [28]. In this scenario, the completion time of job j, which is denoted by C_j, must be $r_j \leq C_j \leq d_j \quad \forall j \in J$ [28].

In some cases, there can be *sequence-dependent setup times*, which means that there is a setup time when going from job i to j [41]. This setup time can be denoted as s_{ij} [41]. In this case s_{0j} denotes the setup time when j is the starting job and s_{j0} denotes the cleanup time when j is the last job [41]. A SMSP with sequence-dependent setup times, where the goal is to complete the last job as early as possible (also known as the make-span), is identical to the *Traveling Salesman Problem* (TSP) [41].

4.2 Routing Problems

In the TSP a salesman needs to visit a set of n cities [52]. He starts in his home city and visits each city once before returning to his home city [52]. The cost of traveling between each pair of cities is known in advance and can be distance, travel time, money, or any other appropriate cost [52]. The goal of the TSP is to visit all cities and return home such that the total travel cost is minimized [52]. An example of the TSP is displayed on Figure 4.1.

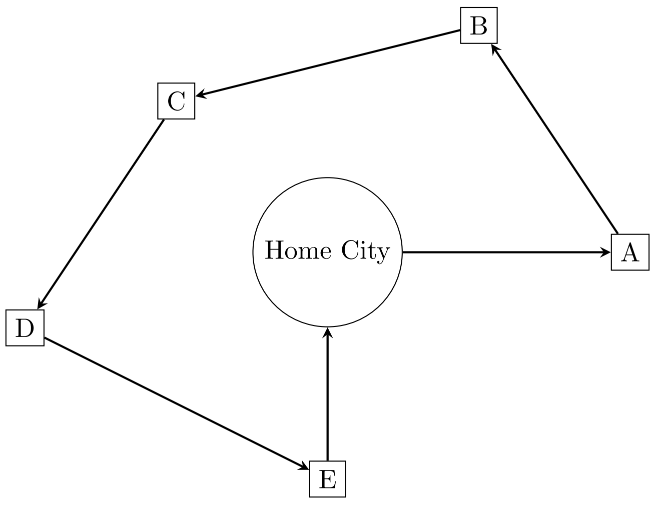


Figure 4.1: *An example of the TSP.*

A surprisingly number of problems can be modelled in terms of the TSP and the TSP can be extended and generalized to a wide range of routing problems [41]. One of the most famous of these is the *Vehicle Routing Problem* (VRP) [18].

4.2.1 The Vehicle Routing Problem

The VRP was originally proposed by Dantzig and Ramser in 1959 as the *Truck Dispatching Problem* [18]. In this problem, the aim was to design optimal routes for a fleet of gasoline trucks to service a set of customers [18]. Today, the VRP is one of the most studied problems within OR and has found applications in almost every industry involved with routing [18].

The VRP can be represented by a network graph $N = (V, A)$ where V is a set of vertices, which represents locations that must be visited, and vertex 0 represents the depot [12]. A is a set of directed arches that represents feasible roadways between the vertices [12]. Each arch $(i, j) \in A$ has an associated travel cost denoted by c_{ij} [12]. If arch $(i, j) \in A$ is traversed, then the decision variable $x_{ij} = 1$ otherwise $x_{ij} = 0$ [12]. The purpose of the VRP is to design routes for K available vehicles such that the total travel cost is minimized and all vertices are visited [12]. The vehicles can only leave and return to the depot once [12]. A simple example of the VRP is illustrated on Figure 4.2.

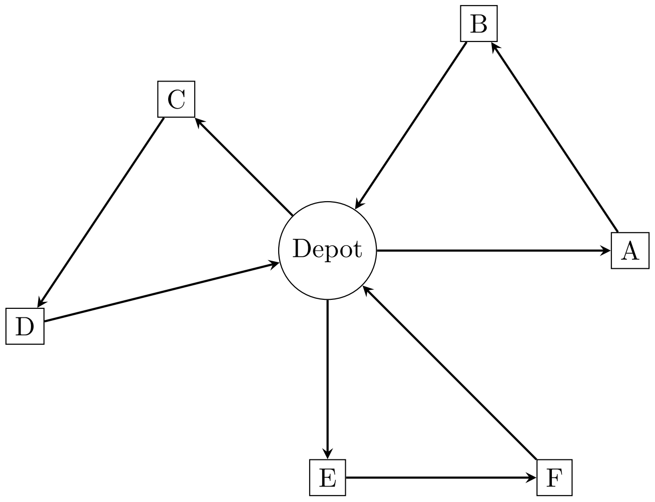


Figure 4.2: *An example of the VRP.*

The basic decisions in the VRP is the assignment of vehicles to vertices and the sequencing of these vertices [30]. Since the first VRP was proposed in 1959, a variety of extensions and variations, known as *rich VRPs*, have been proposed to solve a wide range of real-world problems [30]. A common extension is the inclusion of multiple depots which is referred to as the *Multi-depot Vehicle Routing Problem* (MVRP) [29]. An example of the MVRP is displayed on Figure 4.3

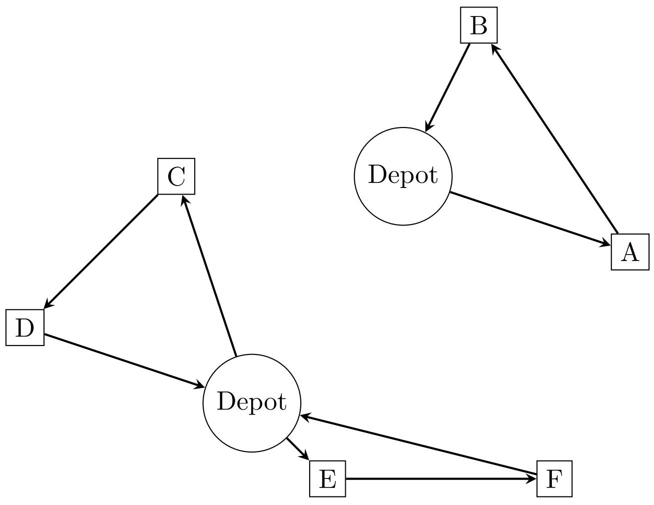


Figure 4.3: *An example of the MVRP.*

As illustrated on Figure 4.3 the MVRP incorporate multiple depots all of which have a limited number of vehicles available [29]. Each vehicle must leave from its designated depot and return to its designated depot [29].

A classic rich VRPs is the *Capacitated Vehicle Routing Problem* (CVRP) [30]. In this variation, all vertices $i \in V$ have a demand p_i for a given resource and all vehicles have a maximum capacity Q of this resource. The aim is to visit all vertices without violating the capacity constraints of the vehicles [30]. In the classic VRP, the fleet of vehicles is assumed to be *homogeneous* which means that all vehicles are identical [30]. However, the VRP can also be extended to a *heterogeneous* fleet of vehicles where each vehicle has different characteristics [4]. With a homogeneous fleet, vehicles are usually denoted implicitly by a circuit [4]. However, with a heterogeneous fleet, it is necessary to explicitly model the vehicles with a set of vehicles K such that when vehicle $k \in K$ traverses arch $(i, j) \in A$ then $x_{ijk} = 1$ otherwise $x_{ijk} = 0$ [4]. A classic example are vehicles with different capacities. In these cases, the capacity of each vehicle $k \in K$ can be denoted by q_k [4].

Another common extension is a class of VRPs named *Vehicle Routing Problems over Time* [30]. These problems usually incorporate a time-horizon T and the vehicles must perform multiple routes during this time-horizon [30]. One example of this is the *Vehicle Routing Problem with Release and Due Dates* (VRP-RD) [30]. In this problem, all vertices $i \in V$ have a release date r_i and a due date d_i and all vertices must be visited after their release date but before their due date [30]. These dates can be *soft dates*, which means that it is possible to violate these constraints, but it comes at a penalty in the objective function, or *hard dates* where it is not allowed to violate these constraints [30].

4.2.2 The Technician Scheduling and Routing Problem

Recently a new type of rich VRPs, dealing with *resource-constrained routing and scheduling*, has emerged [35]. The distinctive feature of these problems is that they incorporate various resources to meet the demands of customers [35]. A common resource-constrained routing and scheduling problem is the *Technician Scheduling and Routing Problem* (TSRP) [38]. This is an extension of the VRP-RD and is commonly used within the telecommunications industry to schedule repairs and interventions [38]. In the TSRP, each technician has a set of skills, tools, and spare-parts and must serve a set of requests that require a subset of these resources [38]. Each request has a soft time-window and a priority that determines the penalty for violating the release and due dates [38]. The problem consists of designing tours for technicians so that skill levels are respected; the technicians have the required tools and spare parts for the request; technicians' working capacity is respected; and requests are served within their time window [38]. Compared to most VRPs, the TSRP differs in its objective, as it usually does not seek to minimize the total travel cost [39]. Instead, the TSRP seeks to optimize a quality of service such as minimizing earliness and lateness; maximizing the fairness of the work-schedule; or maximizing the priorities of scheduled jobs [39]. For this thesis, a variation of the TSRP is presented, named the *Auditor Scheduling and Routing Problem* (ASRP).

4.3 The Auditor Scheduling and Routing Problem

The ASRP is designed to solve the problem the DGA faces when auditing gambling facilities. The ASRP is a variant of the TSRP; however, unlike the traditional TSRP, the ASRP only considers skills, not tools or spare-parts. Moreover, the ASRP is solved on a daily basis, and the time-windows consists of hard due dates. Furthermore, the ASRP differentiates between on-site audits, which must be performed at a facility, and desk audits, which must be performed at a depot. Since on-site audits require an available vehicle, the ASRP limits the number of auditors who can leave the depot for on-site audits. This is unlike the classic TSRP where it is assumed that there are enough vehicles for all technicians.

In the ASRP it is assumed that there might be more audits available than the auditors have the capacity to perform. Therefore, the goal is not to perform all audits on a given day but to determine which audits to perform and which audits to postpone until a later date. The aim of the ASRP is to design tours for the auditors so that the vehicle capacity of each depot is respected; the required skill levels are respected; the audits are performed before their due date; and that as many audits are scheduled as possible.

The ASRP is organized as a daily problem where T denotes the set of days and $t \in T$ denotes the current date. It is represented as a network graph $N = (V_t, A)$ where V_t is a set of vertices that represents the audits available on day t. All $i \in V_t$ have a release date r_i such that $t \geq r_i$. The ASRP is a multi-depot problem and the set of depots is represented by L. Each depot $l \in L$ has a limited number of vehicles available on day t, which is denoted by k_{lt}. On-site audits, which are available on day t, are represented by the set O_t, which is a subset of V_t. A denotes a set of directed arches that represents feasible roadways between the vertices in $\{O_t \cup L\}$. Each arch $(i, j) \in A$ has an associated travel time in hours which is denoted as c_{ij}.

All audits $i \in V_t$ have a due date d_i such that $d_i \geq t$. Furthermore, the value $u_i = d_i - t \quad \forall i \in V_t$ denotes the difference between the due date and the current date. All audits $i \in V_t$ have a duration that is represented by p_i. The auditors are represented by the set E and each auditor $e \in E$ has a capacity of q_{et}, which denotes how many hours auditor e is available to work on day t. All auditors are attached exclusively to one depot, and if auditor $e \in E$ belongs to depot $l \in L$, then $b_{el} = 1$ otherwise $b_{el} = 0$. Not all auditors are qualified to perform an audit; therefore, $g_{ie} = 1$, if auditor $e \in E$ can perform audit $i \in V_t$, otherwise $g_{ie} = 0$.

If arch $(i, j) \in A$ is traversed by auditor $e \in E$, then the decision variable $x_{ije} = 1$ otherwise $x_{ije} = 0$. Furthermore, if audit $i \in V_t$ is performed by auditor $e \in E$, the decision variable $y_{ie} = 1$ otherwise $y_{ie} = 0$. Lastly, if auditor $e \in E$ is assigned to an on-site audit the decision variable $a_e = 1$ otherwise $a_e = 0$. All audits $i \in V_t$ have an audit date, which is denoted by $\mathcal{C}_i$, and if $\sum\limits_{e \in E} y_{ie} = 1$ on day t, then $\mathcal{C}_i = t$ otherwise $\mathcal{C}_i = -1$.

Unscheduled audits will be postponed to a later date, but must be performed before d_i. All auditors $e \in E$ can only leave and return to their depot b_{el} once and must be able to perform the audits to which they are assigned. Furthermore, each auditor $e \in E$ cannot spend more than q_{et} hours traveling and auditing on day t and all depots $l \in L$ cannot have more than k_{lt} auditors leaving for on-site audits on day t.

4.4 Literature Review

The DGA is not the only regulatory agency in which a resource-constrained routing and scheduling problem is utilized. Gamrath et al. proposes the *Toll Enforcement Problem* (TEP) where inspectors from the German Federal Office for Goods Transport (BAG) must ensure that tolls on German highways are paid [15]. The TEP is, like the ASRP, a multi-depot problem where inspectors leave and return to the same depot [15]. However, in the TEP, the inspectors are scheduled to *tasks* which are sections of the highway and a tour is a combination of these tasks [15]. Unlike the ASRP, which considers skill levels, the TEP assumes that inspectors can perform all tasks and tasks are assigned exclusively to inspectors from a designated depot [15].

A significant commonality between the ASRP and the TEP is that they both utilize a rolling time-horizon. The ASRP is organized as a daily problem, and the TEP is organized as a monthly problem [15]. This decomposition is probably the result of the scale of the problems, since it reduces them to a more manageable size [17]. Considering that the ASRP deals with around 6500 facilities and the TEP tackles a problem with around 50k kilometers of highway [15], this decomposition seems necessary.

In most VRPs and TSRPs, all vertices must be visited or a penalty is incurred. The ASRP and TEP relaxes this constraint since it cannot be assumed that inspectors or auditors have the capacity to perform all tasks or audits within a month or a day. Unlike the TEP, which does not incorporate release and due dates, the ASRP uses due dates in the objective function and as a constraint. The TEP and ASRP also differ in their objectives. The ASRP seeks to schedule as many audits as possible, while the TEP seeks to find a compromise between providing fair schedules and maximizing the number of tasks performed [15].

Another problem, similar to ASRP, is the *Technicians and Interventions Scheduling Problem* (TISP) [14]. This problem was proposed by Dutot et al. in the 2007 ROADEF challenge, where a group of technicians, with the correct combination of domain skills, must be scheduled to perform interventions before their due date [14]. As will be clear from Section 5.4, the concept of domain skills is closely related to the ASRP, where auditors must have a high enough skill level within an audit type to perform an audit. However, in the ASRP, only a single auditor needs to perform an audit, while the TISP usually requires multiple qualified auditors to perform an intervention [14]. In both the ASRP and the TISP the audits and interventions have an associated priority; however, they differ in how these are incorporated. In the TISP the priority serves as an important evaluation metric [14], whereas they are only used to determine due dates and release dates in the ASRP. Both the TISP and the ASRP incorporate due dates; however, the ASRP incorporates hard due dates while the TISP incorporates soft due dates [14].

A significant difference between the TISP and the ASRP is that the TISP does not have capacity constraints in terms of work hours [14]. Instead, the only capacity constraint is how many auditors are available for interventions on a given day [14]. Furthermore, the TISP does not incorporate routing and is mainly concerned with assembling the right technicians roster on any given day [14].

In terms of routing, the ASRP shares some features with the *Multi-Period Vehicle Routing Problem with Due dates* (MPVRPD) which is proposed by Architte et al. [2]. In this problem, a set of customers must be served within a set of days T [2]. All customers have a release date and a due date, and customers who have a release date within T but the due date is beyond T can remain unvisited at a cost [2]. The aim of the MPVRPD is to design routes for each day in T such that travel costs and penalty costs are minimized [2]. Like the MPVRPD, the ASRP incorporates hard due dates, and the routes are designed on a daily basis. However, the MPVRPD explicitly models the entire time-horizon [2], while the ASRP decomposes the problem into a daily problem. Moreover, since the ASRP solves the problem on a daily basis, the release dates are handled in pre-processing by only selecting audits where $r_i \geq t$. This is unlike the MPVRPD where the release dates are explicitly represented in the model [2].

Originally, the ASRP was intended to be modeled as an extension of the MPVRPD which incorporated skill levels like the TISP. The idea was to have a planning horizon where all released audits had to be visited before their due date by a qualified auditor. However, the problem was too large to be solved with this approach, and a rolling time-horizon was implemented instead. This results in the model outlined in Chapter 8.

An interesting aspect of the ASRP is the differentiation between on-site and desk audits, since it means that some audits must be performed at a depot. This is unlike most VRPs and TSRPs where the depot only serves as a start and end point. If the ASRP was a single-depot problem, this would not be an issue since the desk audits could be modeled as vertices with the same location as the depot. However, considering this is a multi-depot problem, it is difficult to determine the correct travel cost between an on-site audit and a desk audit, since a desk audit can have the same location as any of the depots. To overcome this, the ASRP incorporates a disconnected network graph where there are no arches going to and from the desk audits. This is unlike most VRPs and TSRPs, which tend to employ a fully connected network graph.

4.5 Conclusion

This chapter has outlined the basic theory behind mathematical routing and scheduling. A scheduling problem is concerned with the allocation of resources to tasks over a given time period and generally takes the form of a set of jobs, which must be processed by a set of machines, where the aim is to optimize some performance measure. These problems can be extended to include release dates, due dates, and sequence-dependent setup times. A Single Machine Scheduling Problem (SMSP) with sequence-dependent setup times is identical to the Traveling Salesman Problem (TSP).

In the TSP, a traveling salesman must leave his home city and visit n vertices once before returning to his home city. This problem can be generalized to the Vehicle Routing Problem (VRP), which seeks to design optimal routes for a fleet of k vehicles. The VRP can be made up of a homogeneous or a heterogeneous fleet. If the fleet is homogeneous, all vehicles have the same characteristics and can be represented implicitly as a circuit. If the fleet is heterogeneous, the vehicles have differing characteristics, such as capacities, and must be modeled explicitly. The VRP is one of the most commonly applied problems within OR, and it has been extended to address a wide variety of real-world problems. These extensions are referred to as rich VRPs.

A common rich VRP is the Multi-depot Vehicle Routing Problem (MVRP) where there are multiple depots which have a set of designated vehicles attached. Another classic rich VRP is the Capacitated Vehicle Routing Problem (CVRP), where all vertices have a demand for a given resource and all vehicles have a maximum capacity, which cannot be exceeded, when visiting the vertices.

A broad class of rich VRPs is Vehicle Routing Problems over time. In these classes of problems a time-horizon is incorporated and the vehicles must perform multiple routes over this time-horizon. A common VRP over time is the Vehicle Routing Problem with Release dates and Due dates (VRP-RD), where all vertices have a release date and a due date, which represent the earliest and latest time they can be visited within the time-horizon.

Recently, a new type of rich VRPs has emerged which deals with resource-constrained routing and scheduling. These classes of problems incorporate resources that must be used to serve the vertices. A common type is the Technician Scheduling and Routing Problem (TSRP). In this problem a set of technicians must serve a set requests such that they have the correct skill levels, spare-parts, tools, and that the requests' release and due dates are respected. The aim of the TSRP is usually to optimize some sort of quality of service, such as the work-schedule, the job priorities, or minimizing earliness and lateness.

This thesis presents the Auditor Scheduling and Routing Problem (ASRP), which is a variation of the TSRP. The ASRP is organized as a daily problem, where the aim is to schedule as many audits as possible. The audits are categorized into desk and on-site audits, and all audits have hard due dates which must be respected. A qualified auditor must be assigned to the audit, and all audits have a duration it takes to perform the audit. All auditors have limited work capacity and belong to a designated depot. All depots have a limited vehicle capacity, which constrains the number of auditors that can leave for on-site audits. It is assumed that there might be more audits than there are available auditors; therefore, the critical decision is not only the assignment and sequencing of vertices, but also which audits to perform and which ones to postpone.

The Technicians and Interventions Scheduling Problem (TISP); the Multi-Period Vehicle Routing Problem with Due dates (MPVRPD); and the Toll Enforcement Problem (TEP) was presented and compared with ASRP. In the MPVRPD, a set of customers must be visited before a due date and after a release date during a planning horizon. In the TISP, a group of technicians with different skill levels must be scheduled to interventions within the telecommunications industry. In the TEP, inspectors must patrol the German highways for toll payments.

The ASRP was originally intended to be modeled as an extension of the MPVRPD with skill levels like the TISP. However, due to the scale of the ASRP, this was unattainable. Instead, a rolling time-horizon, which is likewise used in the TEP, was incorporated. But, unlike the TEP, which is organized as a monthly problem, the ASRP is organized as a daily problem.

A notable difference between the ASRP and other related problems is the distinction between on-site and desk audits. Desk audits must be performed at a depot, and on-site audits must be performed at a physical location. This is unlike most VRPs and TSRPs where the depot only serves as a start and end point for the vehicles. Since there are multiple depots, it is difficult to determine the travel cost between on-site and desk audits, since desk audits can be performed at any depot. Therefore, the ASRP employs a disconnected graph, which is unusual for these types of problems.

Chapter 5

Problem Description

This chapter provides a detailed description of the problem. Section 5.1 details how audits are prioritized and Section 5.2 describes how due dates and release dates form an audit window. Section 5.3 outlines how time is discretized; how this discretization is used to represent auditor and vehicle availability; and how the availability is used to generate daily capacity constraints. Section 5.4 describes how audits are categorized into different audit types with required skill levels, and how these are used to create an accomplice matrix. Section 5.5 details how the duration of each audit is calculated. Section 5.6 outlines how an auditor-depot matrix is used to assign auditors to depots, while Section 5.7 details how the cost of traveling between facilities is represented by a travel-time matrix. Lastly, Sections 5.8 and 5.9 describe how the DGA currently assigns on-site and desk audits to auditors.

5.1 Audit Priorities

Going into and coming out of an audit there is a risk assessment which can be either *White*, *Green*, *Yellow*, or *Red*. This assessment describes how important the audit is and is visualized in Figure 5.1.

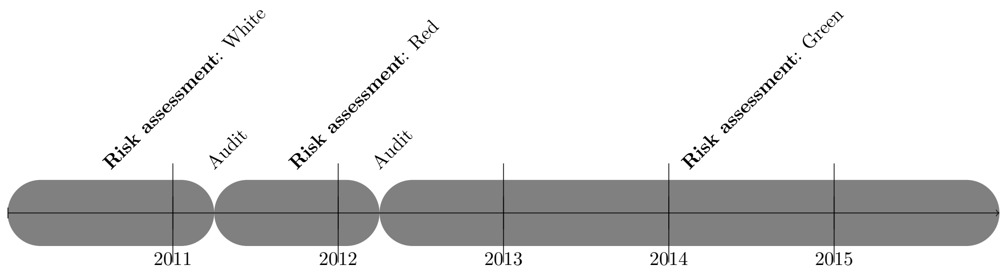


Figure 5.1: *Risk assessments before and after audits.*

To model this, the assessments are converted to priorities with the following numerical values: 1 (Green), 2 (Yellow), 3 (White), and 4 (Red). This is demonstrated in Figure 5.2.

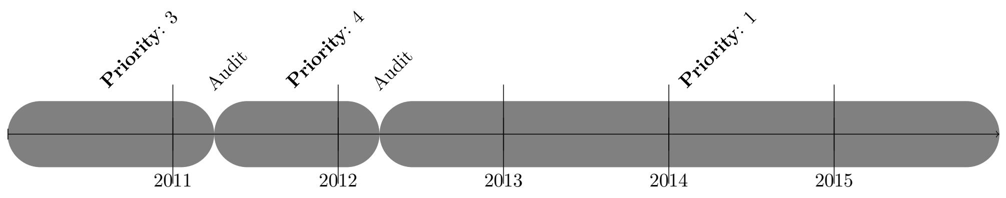


Figure 5.2: *Risk assessments converted to priorities.*

By converting the risk assessments to numerical priorities they can easily be used as potential weights in a model. Furthermore, since priorities can be any R_0^+, they can serve as a more accurate way for the DGA to prioritize audits. However, for the time being, they serve as a way to determine when to schedule the next audit, which is described in Section 5.2

5.2 Due Dates, Release Dates, and Audit Dates

All audits must be performed before a given deadline, which is referred to as the *due date*, this date is usually determined by the priority. However, since a facility needs time to correct a potential infraction, they cannot be visited too early. The earliest audit date is called the audit's *release date.* Together, the due date and the release date form an *audit window* which represents the time an audit can be scheduled. This is demonstrated on Figure 5.3.

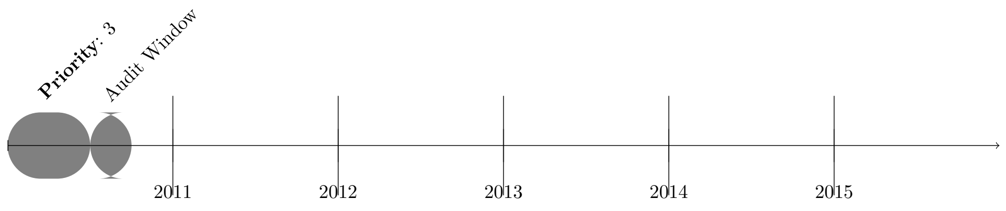


Figure 5.3: *An example of an audit window.*

Figure 5.3 illustrates an example of an audit window where there is a priority 3 prior to the audit. Since this audit has not been performed yet, there is no priority going out of the audit. After an audit has been performed, the next audit window is manually scheduled by an auditor. The time between audit windows is determined by the priority; audit type (see Section 5.4); and the type of facility (gambling hall, kiosk, casino, etc.). This is demonstrated in Figure 5.9.

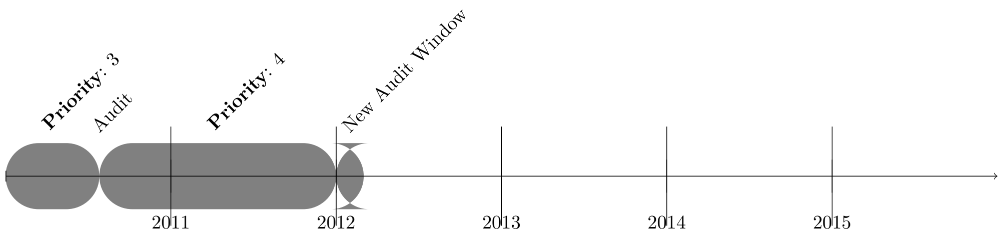


Figure 5.4: *Creation of a new audit window.*

The creation of due dates, release dates, and priorities will be handled externally from the model. That is, it will remain the responsibility of the auditor who performed the audit to insert a new audit with an associated release date, due date, and priority.

5.3 Time Discretization

The timeline has been discretized by splitting each day from 01-01-2000 to 12-31-2030 into 24 hours. Each hour represents a *time slot* which is shown on Figure 5.5.

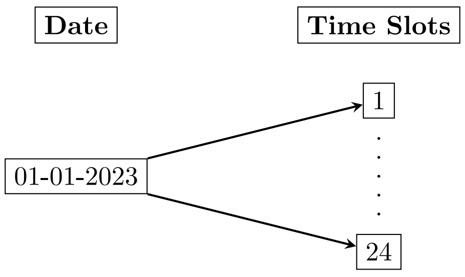


Figure 5.5: *How dates are paired with time slots.*

This offers the opportunity to model what hours auditors and vehicles are available and it forms the basis for calculating q_{et} and k_{lt}.

5.3.1 Auditor and Vehicle Availability

Auditors and vehicles have a public calendar that shows when they are available. For auditors, this means that holidays, vacations, and private appointments, within working hours, are posted on their calendar. Vehicles also have a calendar where employees, within the tax ministry, can reserve a vehicle. To model this, each auditor and vehicle have an associated availability within each time slot which is represented as a binary value of either 1 (available) or 0 (unavailable). This is demonstrated on Figure 5.6.

Date: 01-03-2022
Time Slot: 12
Employee: 15
Available: 1

Figure 5.6: *The available of employee 15 on time slot 12 on the third of January 2022.*

This approach opens up the future possibility of modeling the opening hours of a facility and implementing daily time-windows. This can be done by also giving facilities a binary availability value on a given time slot. However, to keep the scope of this project limited, the opening hours of the facilities have not been implemented. Moreover, all vehicles have currently been set as available 24 hours each day and the auditors are unavailable on weekends and holidays. On workdays, the auditors have the following available hours:

- **Monday**: 8 - 18
- **Tuesday**: 8 - 17
- **Wednesday**: 8 - 15
- **Thursday**: 8 - 15
- **Friday**: 8 - 13

The reasoning behind the differing hours of availability is to, first and foremost, mimic a normal work week as closely as possible. Moreover, by having at least two days where the auditors have more than eight available hours, it is possible to include on-site audits where the total travel and audit time might be above 8 hours. However, for future work on this project, it would be necessary to link auditor and vehicle calendars directly to the database.

5.3.2 Daily Depot and Auditor Capacity

The time slots are used as the basis for calculating the daily capacity of each auditor and depot. The auditor capacity q_{et} simply refers to how many hours auditor $e \in E$ is available on day t. To define it, let $m_{etn} = 1$ if auditor $e \in E$ is available on time slot n on day $t \in T$ otherwise $m_{etn} = 0$. The auditor capacity is then defined as:

$$q_{et} = \sum_{n=1}^{24} m_{etn} \quad \forall e \in E, \forall t \in T \tag{5.1}$$

An example of how Formula 5.1 calculates auditor capacity is shown on Figure 5.7.

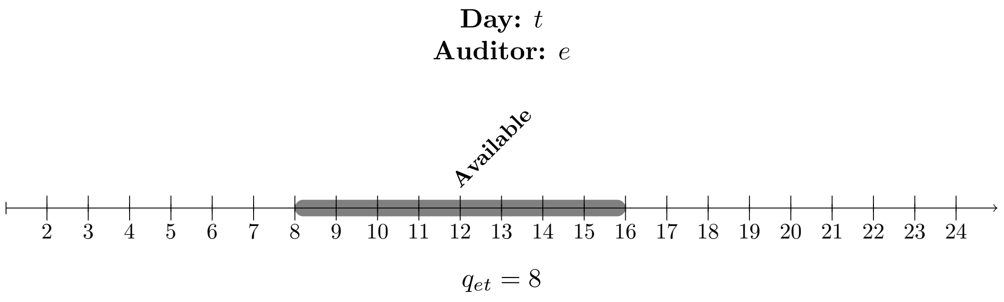


Figure 5.7: *How auditor e's capacity on day t is calculated.*

This method was chosen since it provides a simple and intuitive way for calculating the capacity. For each depot $l \in L$, the daily capacity k_{lt} refers to how many vehicles are available for driving on day t. To define it, let K_l denote the set of vehicles in depot $l \in L$, and let $o_{ktn} = 1$ if vehicle $k \in K_l$ is available on time slot n on day $t \in T$ otherwise $o_{ktn} = 0$. The depot capacity is then defined as:

$$k_{lt} = |\{k \mid k \in K_l \wedge \sum_{n=6}^{18} o_{ktn} = 12\}| \quad \forall t \in T, \forall l \in L \tag{5.2}$$

What Formula 5.2 calculates is, simply, the number of vehicles available on all time slots from 6 to 18 at depot l on day t. This calculation is illustrated on Figure 5.8.

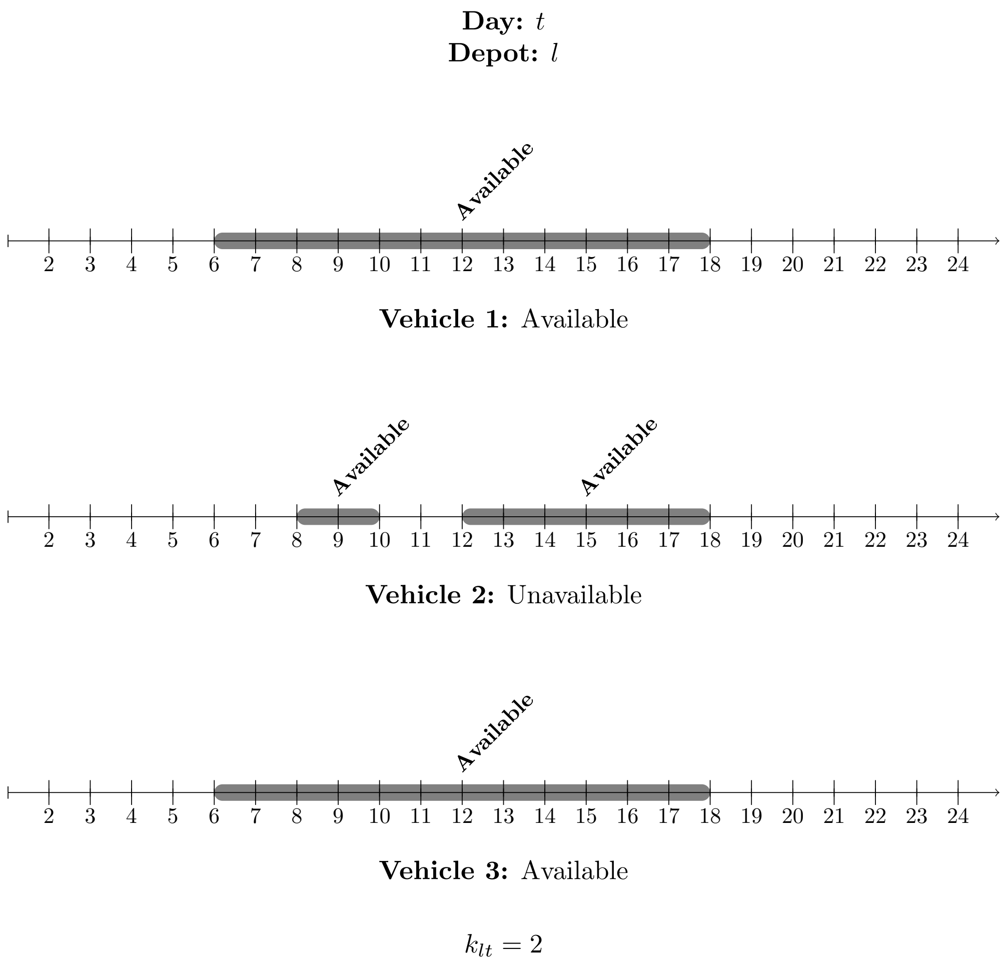


Figure 5.8: *How a depot's vehicle capacity is calculated.*

In Figure 5.8 there are only two vehicles, which are available at all hours from 6 to 18. Therefore, the vehicle capacity of the depot is 2. This definition of depot capacity was chosen because it ensures that an auditor uses a vehicle, which is unassigned most of the day. Moreover, by only counting vehicles, which are available for 12 hours, auditors should have plenty of room for delays without affecting another employee's work schedule.

5.4 Skill Levels and Audit Types

There are 16 different audit types. Not all auditors can perform all audit types, and some audit must be performed by an experienced auditor. To represent this, each audit has an audit type and a required skill level within this audit type. This is demonstrated in Figure 5.9.

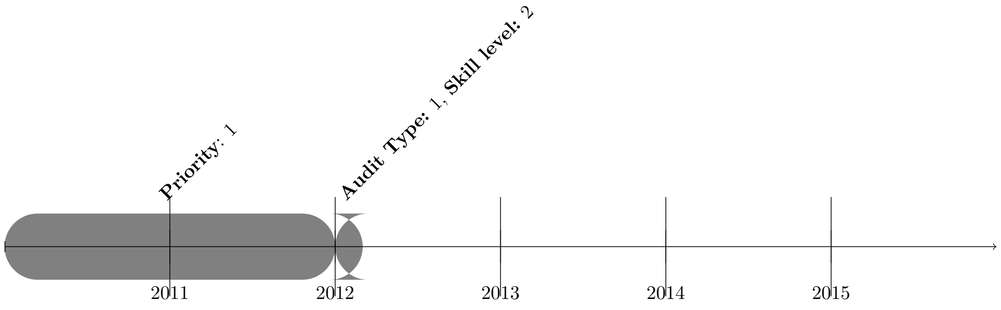


Figure 5.9: *An audit of type 1 with a required skill level of 2.*

There are currently 4 skill levels: 0, 1, 2, and 3. To perform an audit, the auditor must have a skill level, within that audit type, which is higher than the required skill level for the audit. Rather than dealing with skill levels directly in the model, the required skill levels for audits, and the auditors' skill levels, are used to create the binary matrix g_{ie}. This matrix is called the *accomplice matrix* and an example of this matrix is shown in Table 5.1.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
60941	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
60942	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
60944	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
60945	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
60947	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1

Table 5.1: *The accomplice matrix.*

In the table above all auditors are listed along the columns, while all audits are listed along the rows. As Table 5.1 above shows, all auditors are capable of performing all audits, except audit 60947, which cannot be performed by auditor 7. Since the accomplice matrix is created in pre-processing, the mathematical model does not have to consider skill levels and audit types. This is easier to represent mathematically, and it is suspected that it will be more computationally efficient than explicitly modeling skill levels and audit types in the ILP formulation.

5.5 Audit Durations

Each audit $i \in V_t$ has a duration p_i, which denotes how many hours it takes to perform the audit. For this project the durations are calculated in pre-processing with the following formula:

$$p_i = \frac{\varphi_1}{60} + \frac{\varphi_2}{60} i_b + \frac{\varphi_3}{60} i_m \qquad \forall i \in V_t \tag{5.3}$$

φ_1 is the preparation time needed for the audit in terms of minutes. On-site audits generally have a longer preparation time since it often takes time to find the manager at a facility or a parking space.

i_b represents how many types of betting facility $i \in V_t$ provides, and φ_2 represents the duration in minutes that it takes to handle each type of betting provided. For some audit types φ_2 has been set to 0, since the auditor does not pay attention to betting, while other audit types require 6 minutes for each type of betting provided since the auditor must fill out a form.

i_m represents the number of slot machines at facility $i \in V_t$ and φ_3 represents the duration it takes to handle each slot machine. Again, for some audit types, it takes a long time to process a slot machine, since the auditor must disassemble it, while other audit types only require that the auditor does a superficial check.

The values for the parameters described above were selected by interviewing the auditors and letting them demonstrate how they perform different audit types. The auditors were then asked to access the duration of the fastest audits and the duration of the longest audits. Lastly, the parameters' values were tested until the longest and shortest audit durations aligned with the auditors' assessments.

5.6 Auditor-Depot Matrix

In order model, which auditor belongs to which depot, a binary *auditor-depot matrix*, denoted by b_{el}, was created. This matrix is shown in Table 5.2.

	1	2	3	4	5
1	0	0	1	0	0
2	0	0	1	0	0
3	0	0	1	0	0
4	0	0	0	0	1
5	0	0	0	0	1
6	0	0	0	0	1
7	0	0	0	0	1
8	0	0	0	1	0
9	0	0	0	1	0
10	0	0	0	1	0
11	1	0	0	0	0
12	1	0	0	0	0
13	0	1	0	0	0
14	0	1	0	0	0
15	0	1	0	0	0

Table 5.2: *The auditor-depot matrix.*

In the auditor-depot matrix each row represent an auditor and each column represents a depot. If depot l is the designated depot of auditor e then $b_{el} = 1$ otherwise $b_{el} = 0$.

5.7 Travel-Time Matrix

The cost of traveling between facilities was represented by a distance matrix. A distance matrix is created by applying a distance measure to every pairwise combination of data points [3]. This creates a matrix that summarizes the distance between all points in the data set [3]. When applying a symmetrical distance measure, such as an euclidean or a haversine distance measure, the resulting distance matrix will also be symmetric. The matrix will also be square and the distance between two identical points will always be zero, which results in the diagonal only containing zeros [3].

This method of representing travel costs was also used by G. Dantzig, R. Fulkerson, and S. Johnson when they in 1954 showed how linear programming could be used to solve the TSP [11]. Dantzig, Fulkerson, and Johnson used road distances from an atlas to represent the cost of traveling between cities with the objective being to minimize the distance travelled [11].

Since the capacity of the auditors is given in available hours each day, the ASRP is primarily concerned with the time spent traveling between the facilities, rather than the road distances itself. Therefore, the travel distance between facilities is only relevant insofar as it can be used to calculate the travel time between these facilities. However, getting the road distances between all facilities was unattainable for this report. Instead, the latitude, denoted by i_{lat}, and longitude, denoted by i_{long}, of each facility $i \in V_t$ was retrieved using the geolocator geopy [16]. The haversine distance between all pairwise coordinates was then calculated using the scientific computing Python package Scikit-Learn [36]. The haversine distance is a distance measure that measures the distance between two points in a sphere, and is commonly used within navigation systems to provide an approximate distance between two points on Earth [44]. To define the haversine distance, let *arcsin* refer to the inverse sine function and let *cos* and *sin* refer to the cosine and sine functions, respectively. The haversine distance function, between two facilities $i \in V_t$ and $j \in V_t$, is then given as [44]:

$$haversine(i, j) =$$

$$2 \cdot arcsin\left(\sqrt{sin^2\left(\frac{i_{lat} - j_{lat}}{2}\right) + cos(i_{lat}) \cdot cos(j_{lat}) \cdot sin^2\left(\frac{i_{long} - j_{long}}{2}\right)}\right) \tag{5.4}$$

To get the distance in kilometers the haversine distance is then multiplied with earth's radius [44]. This gives the following distance function between two facilities:

$$distance(i, j) = haversine(i, j) \cdot 6371 \tag{5.5}$$

To get an estimate of the travel time in hours between two facilities, the distance is then divided by the speed per hour which is denoted as $\mathcal{V}$. This results in the following formula for calculating the cost of traveling between facilities:

$$cost(i, j) = \frac{distance(i, j)}{\mathcal{V}} \qquad \forall i \in V_t, \forall j \in V_t \tag{5.6}$$

For this thesis $\mathcal{V}$ was set to 80 km/h. Applying this cost function with $\mathcal{V} = 80$ to all facilities will then give the travel-time matrix c_{ij}. An example of this matrix can be seen on Table 5.3.

	1	2	3	4	5	60941	60942	60944	60945	60947	60943	60946
1	0.0	2.5	2.9	2.3	1.0	1.9	2.2	2.7	2.5	2.0	2.2	1.9
2	2.5	0.0	0.6	3.6	2.8	4.2	4.5	5.1	4.7	4.2	4.5	4.2
3	2.9	0.6	0.0	3.7	3.0	4.5	4.8	5.4	5.0	4.5	4.8	4.5
4	2.3	3.6	3.7	0.0	1.4	1.9	1.8	2.5	2.0	1.6	1.8	1.9
5	1.0	2.8	3.0	1.4	0.0	1.5	1.7	2.4	2.0	1.5	1.7	1.5
60941	1.9	4.2	4.5	1.9	1.5	0.0	0.3	0.9	0.6	0.3	0.4	0.0
60942	2.2	4.5	4.8	1.8	1.7	0.3	0.0	0.7	0.3	0.3	0.0	0.3
60944	2.7	5.1	5.4	2.5	2.4	0.9	0.7	0.0	0.5	1.0	0.7	0.9
60945	2.5	4.7	5.0	2.0	2.0	0.6	0.3	0.5	0.0	0.6	0.3	0.6
60947	2.0	4.2	4.5	1.6	1.5	0.3	0.3	1.0	0.6	0.0	0.3	0.3
60943	2.2	4.5	4.8	1.8	1.7	0.4	0.0	0.7	0.3	0.3	0.0	0.4
60946	1.9	4.2	4.5	1.9	1.5	0.0	0.3	0.9	0.6	0.3	0.4	0.0

Table 5.3: *The travel-time matrix.*

The travel-time matrix measures the travel time in hours between two facilities. So if facility i has a travel time travel of 2.5 to facility j then $c_{ij} = 2.5$. Moreover, since the matrix is symmetrical, $c_{ji} = 2.5$. Furthermore, since the diagonal only contains zeros, $c_{ii} = 0$.

Although this method only provides a rough estimation of travel times, it was, upon manual inspection, deemed appropriate. However, since $\mathcal{V} = 80$, there is a tendency for longer routes to be overestimated, since higher speeds on the highway are not taken into account, and underestimated in cities, since slower speed limits and heavier traffic is unaccounted for.

5.8 On-site Audits

Auditors are assigned to on-site audits according to *districts*. Most depots define a district as all facilities within a zip-code. This means that each auditor has a list of zip-codes and it is the auditor's responsibility to perform on-site audits on facilities within these zip-codes. This is visualized in Figure 5.10.

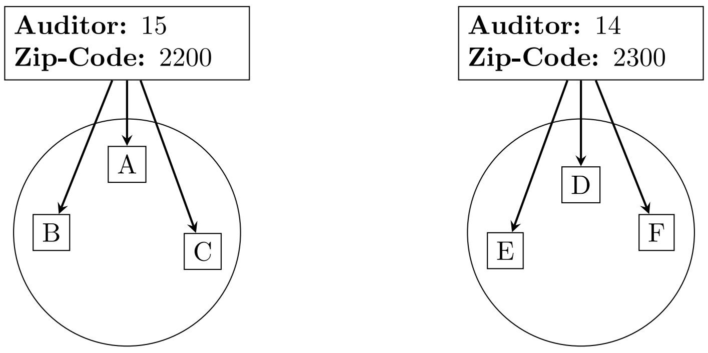


Figure 5.10: *Auditors assigned to audits within their designated zip-code.*

However, one of the depots defines a district as a combination of a zip-code and a facility type. This means that two auditors might have overlapping zip codes, but they perform audits on different types of facilities within these zip-codes. This is demonstrated on Figure 5.11.

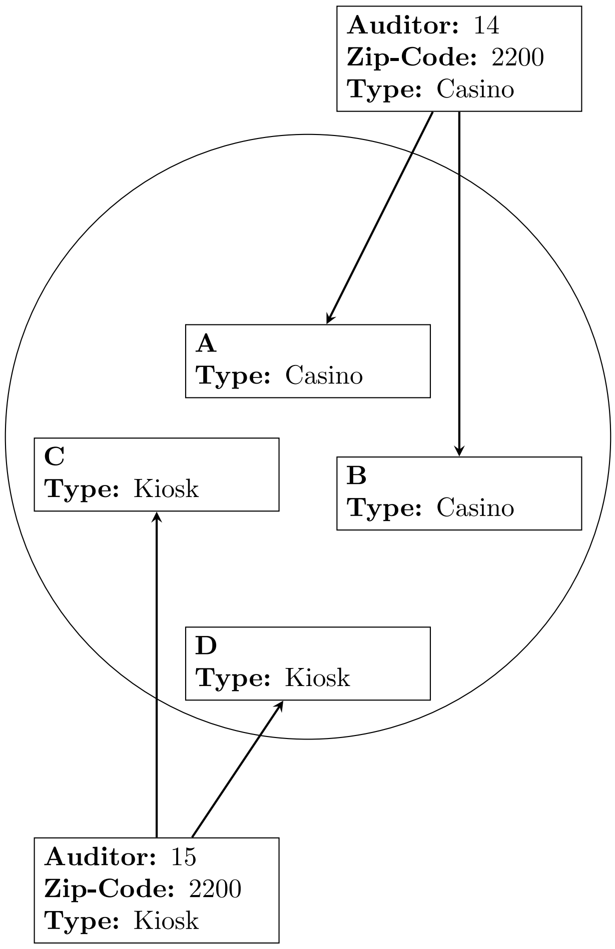


Figure 5.11: *Auditors assigned to audits within their designated zip-code and facility type.*

To create a consistent model, a district have been modeled as being composed of a facility type and a zip code. Each district has one assigned auditor, and the algorithm, for assigning auditors to on-site audits, is visualized on Figure 5.12.

$$\begin{tikzcd}
\textbf{Facility Type} \arrow[dr] & & \\
& \textbf{District} \arrow[r] & \textbf{Employee} \\
\textbf{Facility Zip Code} \arrow[ur] & &
\end{tikzcd}$$

Figure 5.12: *Algorithm for assigning an auditor to an on-site audit.*

The current algorithm, for assigning an auditor to an on-site audit, therefore, involves looking at the facility type and zip-code to get the district. The district then has a designated auditor who will be assigned to the audit.

5.8.1 Yearly On-site Projects

Yearly on-site projects are a type of on-site audits that are also assigned to auditors according to districts. However, each year around 1200 - 1500 facilities are released as potential audits. Each auditor must then choose between 25 and 50 facilities, within their district, to audit before the end of the year.

5.9 Desk Audits

Before 2023, desk audits were assigned to auditors based on districts. Some desk audits were performed as part of the on-site audit, while others were performed on an ad hoc basis. However, since 2023, each type of desk audit is now divided among teams of 3 - 4 auditors. Most desk audits are released in batches, usually at the beginning of the year, and they tend to have a year long audit window. All audits, in this batch, have identical release dates, due dates, and priorities, which is demonstrated on Figure 5.13.

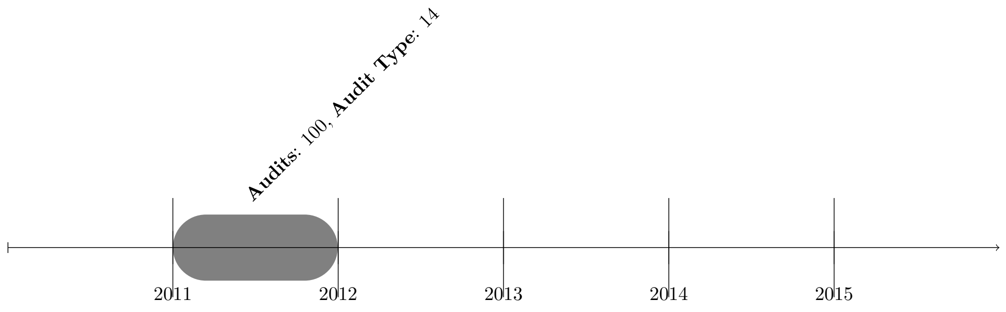


Figure 5.13: *Batch of 100 desk audits with a year long audit window.*

In Figure 5.13 there is a batch of 100 desk audits released. All 100 audits are of the same type and have identical release and due dates. The auditors, assigned to the batch of desk audits, perform the audits following a shared spreadsheet. They sort the batch of audits in descending order according to their required skill levels. The auditors then process the audits in parallel as a queue, starting from an audit within their skill level, as shown on Figure 5.14.

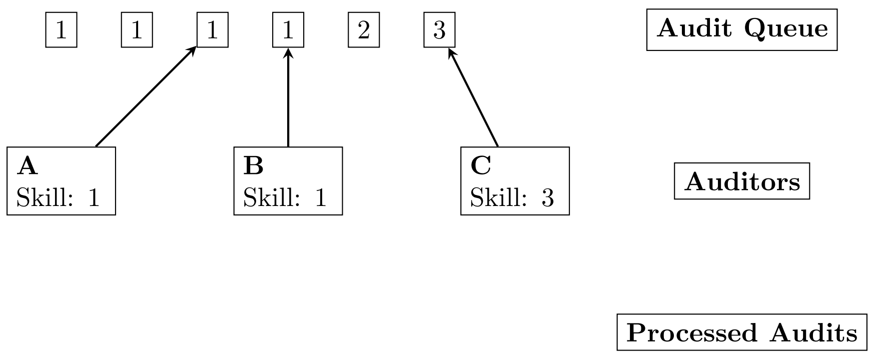


Figure 5.14: *Auditors processing a queue of desk audits.*

When an auditor completes an audit he performs the next available audit in the queue. Since the list is sorted in descending order of skills, the auditor is guaranteed to be capable of performing the audit. This is demonstrated on Figure 5.15.

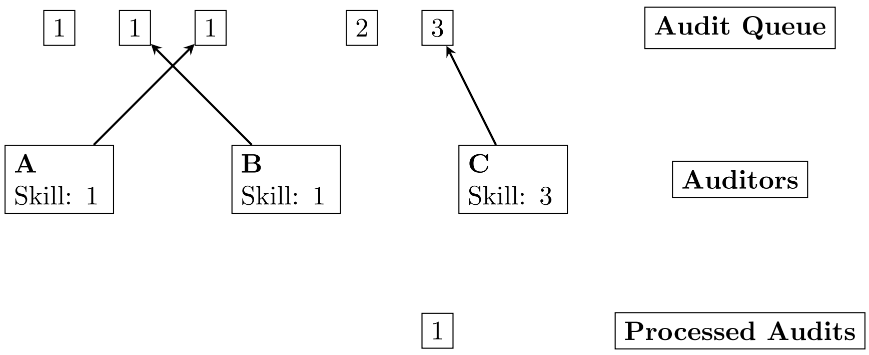


Figure 5.15: *Auditor B moves to next available audit.*

When all audits are completed a new batch for the next year is generated. This is shown on Figure 5.16.

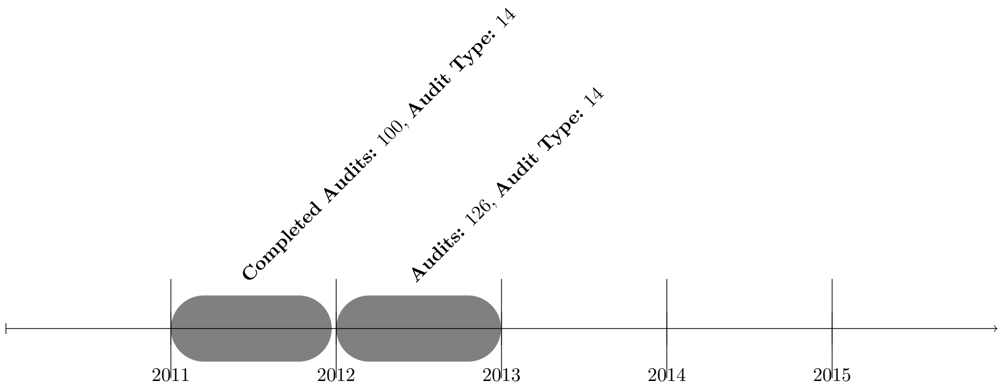

Completed Audits: 100, Audit Type: 14
Audits: 126, Audit Type: 14
2011
2012
2013
2014
2015

Figure 5.16: *A batch of audits for next year is generated.*

Creating the batches of desk audits will remain the responsibility of the auditors. This is due to the fact that, as noted previously, all new audits and their respective due dates, release dates, and priorities are handled externally from the model.

5.10 Conclusion

Risk assessments are categorical values that describe how important an audit is. To represent this, all risk assessments are converted to priorities. This opens up the possibility of using these priorities as weights in a model, and they can serve as a more accurate way for the DGA to prioritize audits, since they can be any R_0^+. However, for the time being, the priorities are primarily used to determine the audit windows. The audit window refers to the time range between an audit's release date and due date. An audit can only be scheduled within the audit window which is determined manually considering the type of facility, the audit priority, and the audit type.

Each day from 01-01-2000 until 12-31-2030 was divided into hours and each hour represents a time slot. All auditors and vehicles are available or unavailable in a given time slot. The availability of vehicles and auditors forms the basis for calculating the daily depot capacity k_{lt} and the daily auditor capacity q_{et}. The daily depot capacity refers to how many vehicles are available to drive on a given day t. It is calculated by counting how many vehicles at a depot are available on all time slots from 6 to 18. This definition ensures that auditors can be delayed without affecting other employees at the tax ministry. The daily auditor capacity refers to how many hours an auditor is available on a given day t. It is calculated by counting the number of available time slots on day t.

Auditors must leave and return to their designated depot. To ensure this, an auditor-depot matrix was introduced. The matrix is denoted by b_{el} and if auditor e is designated to depot l then $b_{el} = 1$ otherwise $b_{el} = 0$.

There is a cost associated with driving between facilities. To represent this, a symmetrical travel-time matrix, which applies a distance measure to all pairwise data points, is introduced. It is calculated by taking the earth's haversine distance between two facilities and dividing it by the travel speed in km/h. The resulting matrix, which is denoted by c_{ij}, contains the travel time in hours between facility i and j. Since the matrix is symmetrical, the value at c_{ij} is identical to the value at c_{ji}, and the values on the diagonal will always be zero. However, it must be noted that this matrix only provides a rough estimate of travel times. When the speed was set to 80 km/h, it was deemed an acceptable proxy; however, it has a tendency to overestimate the travel times on long routes and to underestimate the travel times in cities.

All audits have a duration p_i, which is calculated in pre-possessing by using Formula 5.3. This formula takes the preparation time; how many types of betting a facility provides; and the number of slot machines a facility owns and multiplies these values with predefined parameters for each audit type. The values of these parameters were determined by interviewing auditors and letting them demonstrate how they performed different audit types. Auditors were also asked to access the duration of the longest audits and the duration of the shortest audits. The parameter values were then tested until the longest and shortest audit durations aligned with the auditors' assessments.

All audits have a required skill level within an audit type, and all auditors have a skill level within each audit type. To perform an audit, an auditor's skill level must be higher than the required skill level. To represent this, an accomplice matrix, denoted by g_{ie}, was introduced. If auditor e can perform audit i the value at $g_{ie} = 1$ otherwise $g_{ie} = 0$. Since the accomplice matrix can be created in pre-processing, this matrix eliminates the need for dealing with skill levels and audit types directly in the model.

The concept of on-site audits, which must be performed at the gambling facility, was introduced. On-site audits are currently distributed to auditors according to districts which is comprised of a zip code and a facility type. In addition, the concept of yearly on-site projects was introduced. On-site projects are scheduled according to a list of 1200 - 1500 facilities that are released at the start of the year. Of these facilities, the auditors must select 25 - 50 locations, within their districts, to audit before the end of the year.

Some audits can be performed at the depot. These are referred to as desk audits and they are released in batches each year with a year-long audit window. They are distributed to small teams of 3 - 4 auditors, who sort them according to skill levels, and processes them as a queue starting from the skill level they are capable of performing.

Chapter 6

Data

A substantial part of the project involved data cleaning, data collection, and creating a standardized way to store the data. This process is described in the following chapter. Section 6.1 provides a description of how the data was collected, while Section 6.2 details the schema of the database and the following subsections provide a detailed description of each table in the database.

6.1 Data Collection and Pre-processing

The data for this project was scattered across multiple isolated systems; therefore, it was paramount that these data sources were pre-processed and stored at a centralized location for easy retrieval and analysis. This process is visualized in Figure 6.1.

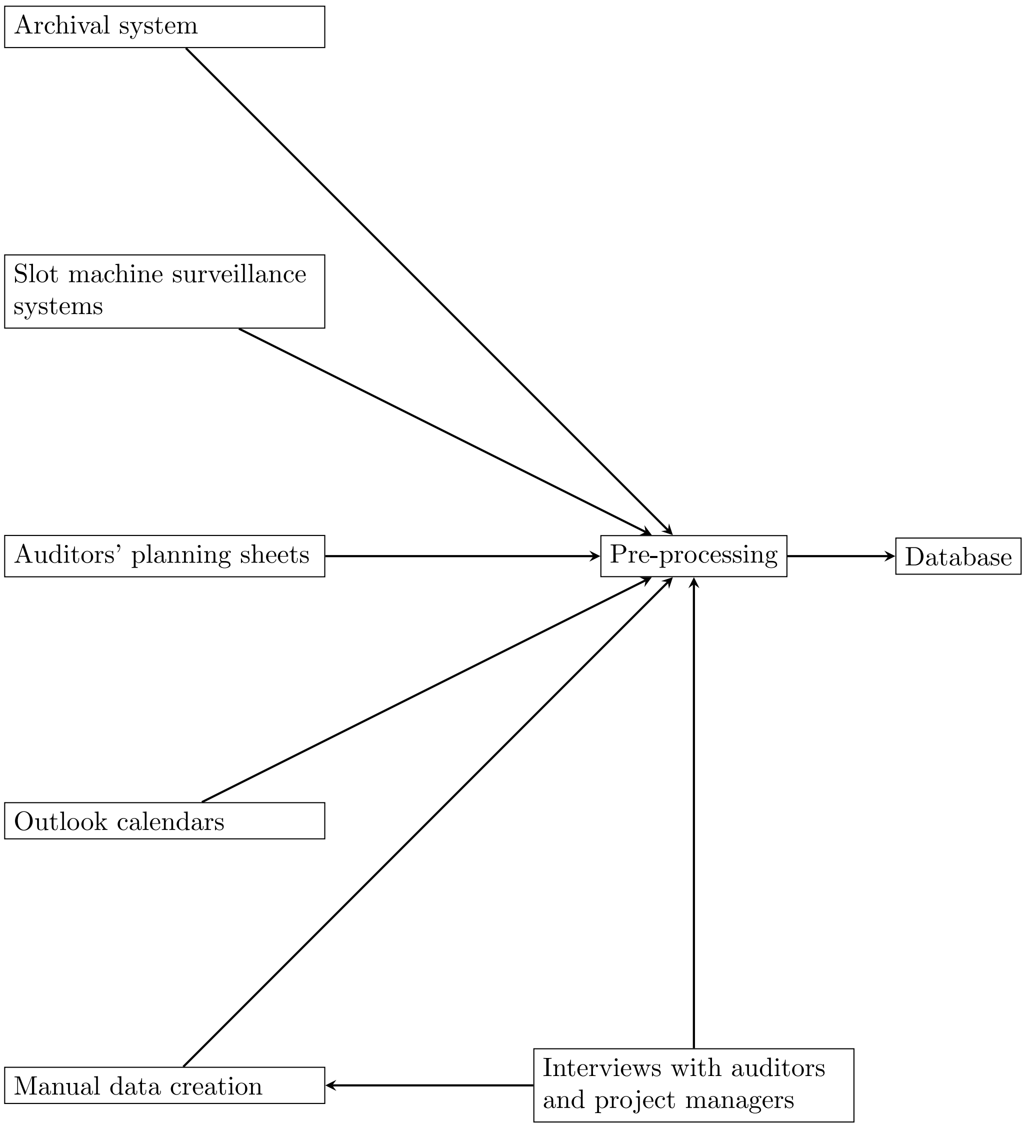

Archival system
Slot machine surveillance systems
Auditors' planning sheets
Pre-processing
Database
Outlook calendars
Manual data creation
Interviews with auditors and project managers

Figure 6.1: *The data sources for pre-processing and storage.*

The archival system was used to extract information about the audit history of each facility and to estimate the number of betting types (i_b) provided. The surveillance systems, which monitor the slot machines, was used to estimate the number of slot machines (i_m) at the facilities. The planning sheets provided data on future audits and auditor districts. The outlook calendars were used to access auditor and vehicle availability. The manual data creation involved creating auditor skill levels, which was gathered through semi-structured interviews with auditors and project managers.

For pre-processing the Python package *pandas* [34] was used. Semi-structured interviews with auditors and project managers were used during pre-processing such that the calculation of audit durations, release dates, due dates, and future planned audits were calculated accurately. Lastly, the data were stored in a relational database.

6.2 Database Schema

The schema for the relational database, used in this thesis, is reflected on the *entity-relationship* diagram, also known as an ER-diagram, in Figure 6.2.

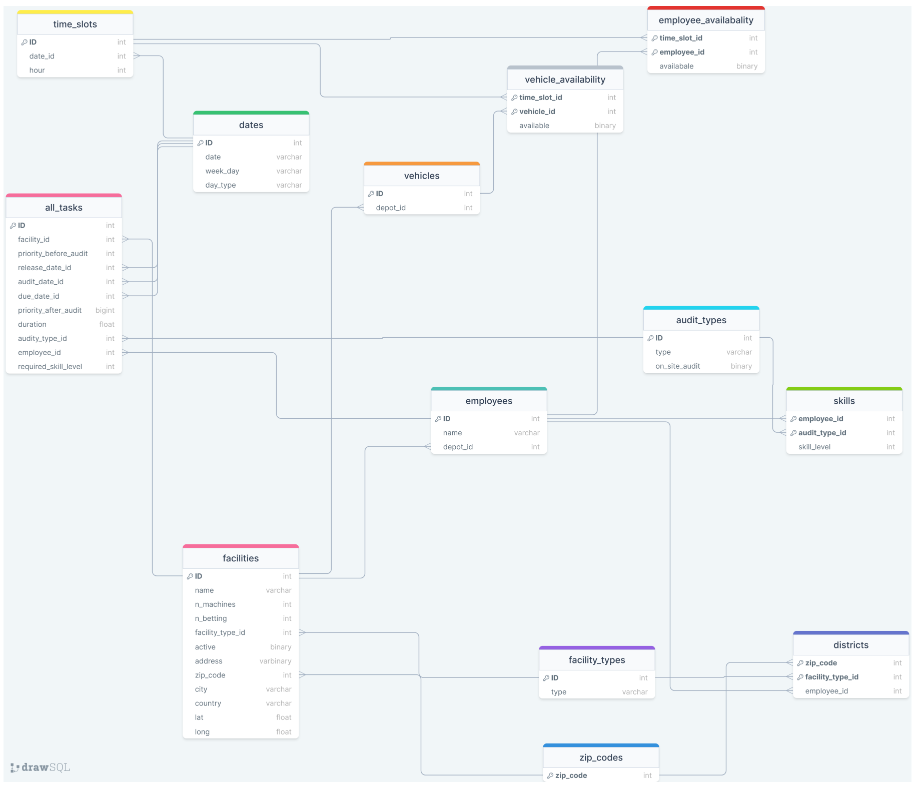


Figure 6.2: *ER-diagram displaying the database schema for this project.*

This normalized database is intended to accurately reflect how the organization currently handle audits. It compiles all relevant information, regarding the audits; audit types; skill levels; facilities; facility types; dates; vehicles; employees; and availability, at one central location. Each table is described in Sections 6.2.1 to 6.2.13.

6.2.1 facilities

All gambling facilities, permit holders, and depots are stored in a table named *facilities*. It creates a *many-to-one* relation [53] between itself and the tables *zip_codes* (see Section 6.2.4) and *facility_types* (see Section 6.2.2). This means that each facility has one zip code and one facility type, while all zip codes and facility types can have multiple facilities linked to them.

ID	name	n_machines	n_betting	facility_type_id	active	address	zip_code	city	country	lat	long
1	SPILLEMYNDIGHEDEN SYD	0	0	15	1	Englandsgade 25	5000	Odense C	DK	55.4	10.4
2	SPILLEMYNDIGHEDEN VESTSJÆLLAND	0	0	15	1	Universitetsvej 2	4000	Roskilde	DK	55.7	12.1
3	SPILLEMYNDIGHEDEN ØSTSJÆLLAND	0	0	15	1	Kratbjerg 236	3480	Fredensborg	DK	56.0	12.4
4	SPILLEMYNDIGHEDEN NORDJYLLAND	0	0	15	1	Vestre havnepromenade 5-9	9000	Aalborg	DK	57.1	9.9
5	SPILLEMYNDIGHEDEN MIDTJYLLAND	0	0	15	1	Lyseng allé 1	8270	Højbjerg	DK	56.1	10.2

Table 6.1: *Table with facilities.*

It contains the following columns:

- **ID**: An unique integer that is the primary key of the table.
- **name**: A string containing the name of the facility.
- **n_machines**: An integer containing the number of slot machines in the facility.
- **n_betting**: An integer denoting how many types of betting the facility provides.
- **facility_type_id**: An integer foreign key that refers to the table *facility_types*.
- **active**: A binary value that denotes whether the facility is active or not.
- **address**: A string representing the address of the facility.
- **zip_code**: An integer foreign key representing the zip code that references the table *zip_codes*.
- **city**: A string representing the city of the facility.
- **country**: A string representing the two-letter country code of the facility.
- **lat**: A floating point representing the latitude of the facility.
- **long**: A floating point representing the longitude of the facility.

The latitude and longitude were obtained by concatenating the address, zip code, city, and country of each facility and then using geopy to extract the coordinates.

6.2.2 facility_types

This table ensures that each facility is of a valid facility type. It contains 16 types of facilities.

ID	type
1	BUT, Kiosk/butik
2	DISBUT, Spilsted kun med lotteri
3	REST, Restauration
4	SHTIL, Spillehal, tilstødende
5	SPBH, Spillebutik og spillehal
6	SBBK, Spillebutik, butik/kiosk
7	SHBE, Spillehal, bemandet
8	SHBUTKIO, Spillehal og butik/kiosk
9	SBSHBK, Spillebutik, spillehal, butik/kiosk
10	SRS, Spillebutik, restauration og spillehal
11	SPBUT, Spillebutik
12	KA, Kasino
13	SR, Spillebutik og restauration
14	KOMBI, Kombi-spillehal
15	DEPO, Depot
16	TILL, Tilladelsesindehaver

Table 6.2: *Table with facility types.*

It contains the following columns:

- **ID**: An unique auto incremented integer, which is the primary key of the table.
- **type**: A string representing the type of facility.

Facility types with IDs 1 - 14 are extracted directly from the DGA's archival system, while IDs 15 and 16 have been created to represent depots and permit holders. Permit holders are facilities that do not provide gambling products, but manage slot machines on behalf of other facilities.

6.2.3 districts

The districts table is used to store which employees are assigned to which districts. It creates a many-to-one relation between between itself and the tables *facility_types* (see Section 6.2.2), *zip_codes* (see Section 6.2.4), and *employees* (see Section 6.2.5). Therefore, each district consists of one facility type and one zip-code, and has one designated employee (auditor). On the other hand, each facility type, zip code, and employee can be attached to multiple districts.

zip_code	facility_type_id	employee_id
1050	1	14.0
1050	2	14.0
1050	3	14.0
1050	4	14.0
1050	5	14.0

Table 6.3: *Table with districts.*

It contains the following columns:

- **zip_code**: An integer foreign key referencing the table *zip_codes* which represents the zip code of a district. It is part of a composite primary key of this table.
- **facility_type_id**: An integer foreign key referencing the table *facility_types*. This column is the second composite primary key.
- **employee_id**: An integer foreign key that references the table *employees* representing the auditor assigned to the district.

6.2.4 zip_codes

The table *zip_codes* stores all Danish zip-codes. It is used to ensure that all facilities and districts have a valid zip code.

zip_code
1000
1001
1002
1003
1004

Table 6.4: *Table with zip codes.*

It contains the following columns:

- **zip_code**: An integer primary key that contains the zip code.

6.2.5 employees

This table stores information about auditors and their depot. It creates a many-to-one relation between itself and the table *facilities* (see Section 6.2.1). This relationship ensures that each employee has one facility, which is their depot, while each depot can have multiple designated employees.

ID	name	depot_id
1	John	3
2	Marianne	3
3	Peter	3
4	Lasse	5
5	Line	5

Table 6.5: *Table with employees.*

It contains the following columns:

- **ID**: An unique auto incremented integer which is the primary key of the table.
- **name**: A string representing the name of the auditor.
- **depot_id**: An integer foreign key referencing the table *facilities* that represents the depot.

6.2.6 audit_types

This table lists all audit types.

ID	type	on_site_audit
1	wrong_h_numbers	0
2	tax_fees	0
3	missing_connection	0
4	night_opening	0
5	past_permission_date	0
6	no_permission_proj	0
7	service_calls	0
8	base_DK_machines	1
9	base_non_DK	1
10	fillings_proj	1
11	minors_using_machines_proj	1
12	illegal_lottery_proj	1
13	money_laundering_proj	1
14	betting_terminals_proj	1
15	lottery_proj	1
16	service_visits	1

Table 6.6: *Table with audit types.*

It contains the following columns:

- **ID**: An unique auto incremented integer which is the primary key of the table.

- **type**: A string representing the name of this type of audit.
- **on_site_audit**: A binary value where 0 represents a desk audit, while 1 represents an on-site audit.

6.2.7 skills

This table represents the skill level of each auditor within different audit types. It is a junction table [53] that links *employees* (see Section 6.2.5) and *audit_types* (see Section 6.2.6). This ensures that all employees have skill levels, within each audit type, and all audit types have multiple employees with differing skill levels. Therefore, it forms a *many-to-many* relation [53] between itself, *employees* and *audit_types*.

employee_id	audit_type_id	skill_level
1	1	0
1	2	0
1	3	0
1	4	0
1	5	0

Table 6.7: *Table with skill levels.*

It contains the following columns:

- **employee_id**: An integer foreign key referencing the table *employees*. This key is part of a composite primary key for the table.
- **audit_type_id**: An integer foreign key referencing the table *audit_types*. This key is also part of a composite primary key for the table.
- **skill_level**: An integer representing the auditor's skill level within an audit type

6.2.8 dates

This table stores all dates from the 1st of January 2000 until the 31st of December 2030.

ID	date	week_day	day_type
1	2000-01-01	Saturday	weekend
2	2000-01-02	Sunday	weekend
3	2000-01-03	Monday	workday
4	2000-01-04	Tuesday	workday
5	2000-01-05	Wednesday	workday

Table 6.8: *Table with dates.*

It contains the following columns:

- **ID**: An unique auto incremented integer which is the primary key of the table.
- **date**: A string representing the date in *YYYY-MM-DD* format.
- **week_day**: A string representing the day of the week from Monday to Sunday.
- **day_type**: A string representing the type of day. It can be weekend, workday, or holiday.

6.2.9 time_slots

This table stores all time slots associated with each date. In this case, each time_slot represents an hour of each day. It creates a many-to-one relation between itself and the table *dates* (see 6.2.8). This means that each date has multiple time slots and each time slot have one date.

ID	date_id	hour
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5

Table 6.9: *Table with time slots.*

It contains the following columns:

- **ID**: An unique auto incremented integer which is the primary key of the table.
- **date_id**: An integer foreign key that references the table *dates*.
- **hour**: An integer representing the time slot of the day.

6.2.10 vehicles

This table stores information about vehicles and their depots. It creates a many-to-one relation between itself and the table *facilities* (see Section 6.2.1). This ensures that each vehicle has a facility, which is their depot, while each depot can have multiple designated vehicles.

ID	depot_id
1	2
2	2
3	2
4	2
5	2

Table 6.10: *Table with vehicles.*

It contains the following columns:

- **ID**: An unique auto incremented integer which is the primary key of the table.
- **depot_id**: An integer foreign key referencing *facilities* and representing the designated depot of this vehicle.

6.2.11 vehicle_availability

This table stores information about vehicle availability on a given time slot. It is a junction table between *vehicles* (see Section 6.2.10) and *time_slots* (see Section 6.2.9). This ensures that all time slots have multiple vehicles attached and that all vehicles are attached to multiple time slots with differing availability values. Therefore, this table forms a many-to-many relationship between *vehicles* and *time_slots*.

time_slot_id	vehicle_id	available
1	1	1
1	2	1
1	3	1
1	4	1
1	5	1

Table 6.11: *Table storing the vehicle availability at each time slot.*

It contains the following columns:

- **vehicle_id**: An integer foreign key that references the table *vehicles*. This column represents the vehicle and is part of a composite primary key for this table.
- **time_slot_id**: An integer foreign key referencing the table *time_slots*. This column represents the time slot and is part of a composite primary key for this table.
- **available**: A binary value where 1 is available and 0 is unavailable.

6.2.12 employee_availability

This table stores information on the availability of employees on a given time slot. It is a junction table between *employees* (see Section 6.2.5) and *time_slots* (see Section 6.2.9). This ensures that all time slots have multiple employees attached, and all employees have multiple time slots with differing availability values. Therefore, this table forms a many-to-many relationship between *employees* and *time_slots*.

time_slot_id	employee_id	available
1	1	0
1	2	0
1	3	0
1	4	0
1	5	0

Table 6.12: *Table storing the employee availability at each time slot.*

It contains the following columns:

- **employee_id**: An integer foreign key referencing the table *employees*. This column represents the employee and is part of a composite primary key for this table.
- **time_slot_id**: An integer foreign key referencing the table *time_slots*. This column represents the time slot and is part of a composite primary key for this table.
- **available**: A binary value where 1 is available and 0 is unavailable.

6.2.13 all_tasks

This table stores all historical and future audits which are planned from 01-01-2000 until 12-31-2030. It creates a many-to-one relation between itself and the tables *dates* (see Section 6.2.8), *employees* (see Section 6.2.5), *facilities* (see Section 6.2.1), and *audit_types* (see Section 6.2.6). This means that all audits have a single release date, due date, audit date, facility, employee, and audit type. However, all dates, facilities, employees, and audit types can be linked to multiple audits.

ID	facility_id	priority_before_audit	release_date_id	audit_date_id	due_date_id	priority_after_audit	duration	audit_type_id	employee_id	required_skill_level
1	7888	1	9220	NaN	9342	NaN	0.4	8	NaN	1
2	7859	1	9556	NaN	9676	NaN	0.4	8	NaN	1
3	7843	1	9294	NaN	9416	NaN	0.4	8	NaN	1
4	7296	1	9272	NaN	9395	NaN	0.4	8	NaN	1
5	522	1	10235	NaN	10356	NaN	0.4	8	NaN	1

Table 6.13: *Table storing all planned audits from 01-01-2000 until 12-31-2030.*

It contains the following columns:

- **ID**: An unique auto incremented integer which is the primary key of the table.
- **facility_id**: An integer foreign key that references *facilities*. This column represents the facility which must be audited.
- **priority_before_audit**: An integer representing the priority of the audit prior to when it was performed.
- **release_date_id**: An integer foreign key referencing *dates*. This column represents the audit release date.
- **audit_date_id**: An integer foreign key referencing *dates*. This column represents the audit date. If an audit has not been performed yet a NULL value is inserted. Note that setting $\mathcal{C}_i = -1$ for unscheduled audits is handled in the pre-processing of the simulation model, not in the database itself.
- **due_date_id**: An integer foreign key referencing *dates*. This column represents the due date of the audit.
- **priority_after_audit**: An integer representing the priority after an audit has been performed. If an audit has not been performed yet, a NULL value is inserted.
- **duration**: A float representing the duration of the audit in hours.
- **audit_type_id**: An integer foreign key referencing *audit_types*. This column represents the type of audit that must be performed.
- **employee_id**: An integer foreign key referencing *employees*. This column represents the auditor who has been assigned to the audit. If the audit have not been performed yet, a NULL value is inserted.
- **required_skill_level**: An integer representing the required skill level, within the audit type, an auditor needs to perform the audit.

6.3 Conclusion

The data for this project came from multiple data sources. These data sources included slot machine surveillance systems, auditor spreadsheets, the DGA's archival system, manual data creation, and outlook calendars. Therefore, it was necessary to compile this data and store it in a centralized location. This process involved manually creating new data and performing pre-processing which included calculating audit durations and determining future release and due dates. The pre-processing and the manual data creation relied heavily on semi-structured interviews to ensure that the data accurately reflected the DGA. The data was stored in a relational database which provides a single access point to all relevant data and simplifies the process of analyzing and modeling the problem.

Chapter 7

Preliminary Analysis

This chapter provides a preliminary analysis of the problem. Section 7.1 investigates how the full problem can be decomposed into three subproblems, and Section 7.2 explores how auditors prefer to schedule their on-site and desk audits during a standard work week. Section 7.4 outlines how the routing can be decomposed to different levels. Section 7.5 analyzes how densely audit windows are spread each year from 2020 to 2024, while Section 7.6 outlines the workload and auditor capacity from 2020 until 2024. Section 7.7 investigates how audit durations are distributed. Lastly, Section 7.8 explores the scale of the problem, and how it can be solved within a reasonable run-time.

7.1 Master Problem and Subproblems

It seems that the full problem, which in this report will be referred to as the *master problem*, can be decomposed into three subproblems. These subproblems are visualized in Figure 7.1.

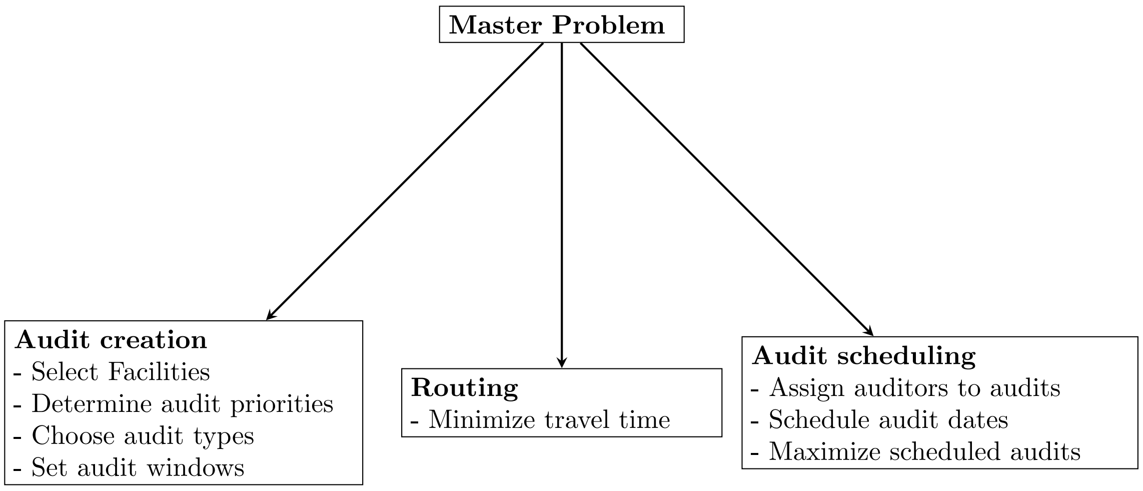


Figure 7.1: *The master problem decomposed to subproblems.*

Due to the substantial amount of manual work required to create new audits, the audit creation subproblem, which includes deciding which facilities to audit; determining audit priorities; and creating appropriate audit windows, is a prime candidate to solve at the DGA. Moreover, an optimizer or a machine learning algorithm could easily take the audit history of a facility; previous infractions at a facility; the priority of a facility; and even fairness into account when selecting which facilities to audit. However, to limit the scope of this project, it will be assumed that the audit creation subproblem is solved since auditors already create new audits manually. Instead, this report will focus on solving the routing and scheduling subproblems. Together these problems form the basis for ASRP and the rest of this report exclusively explores aspects of these subproblems.

7.2 Scheduling Preferences

Since auditors are responsible for scheduling audits and planning routes, within their districts, their scheduling and routing preferences were discussed in semi-structured interviews. Auditors prefer to drive a few long routes each week, rather than short daily routes, and they generally prefer not to have desk audits and on-site audits on the same day, unless they are back at the depot earlier than expected. An example of their preferred schedule can be seen on Figure 7.2.

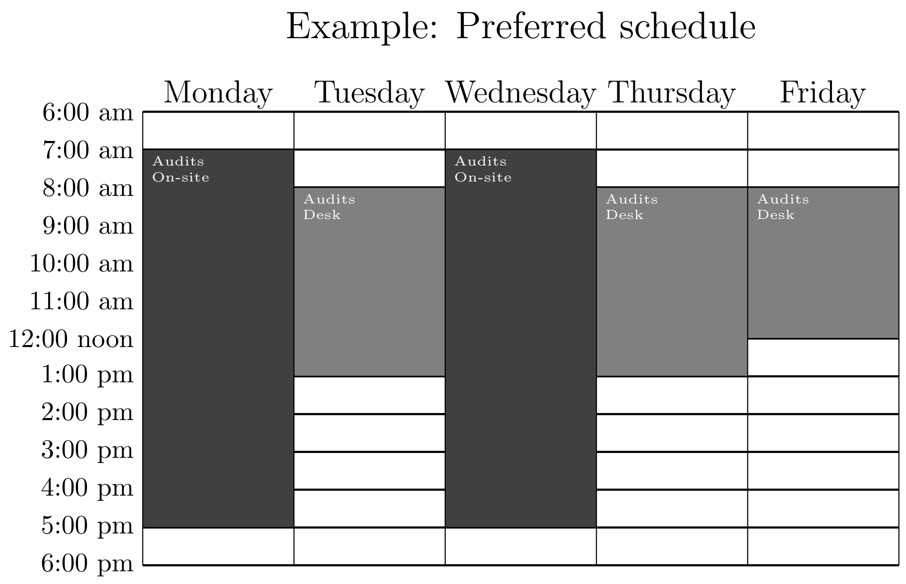


Figure 7.2: *An example of a schedule which auditors prefer.*

As Figure 7.2 shows, the auditors prefer to have a few days with long routes. The other days are used to perform desk audits and the auditors do not have both desk and on-site audits on the same day. A schedule, they try to avoid, is one which involves many short on-site routes spread across the whole week, where they have to perform desk audits and on-site audits on the same day. An example of an ill-favored scheduled can be seen on Figure 7.3.

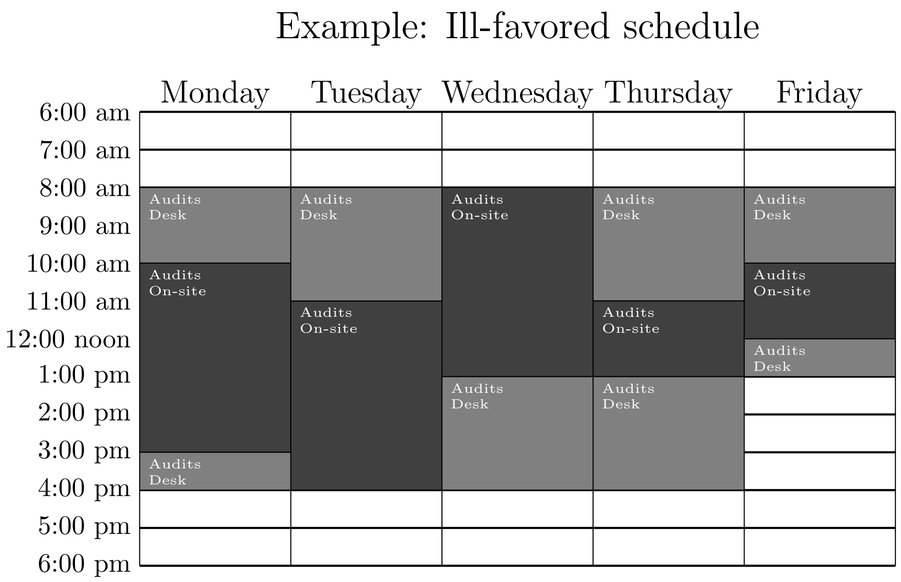


Figure 7.3: *An example of an ill-favored schedule.*

Figure 7.3 illustrates an ill-favored schedule, where the days contain a mixture of both desk audits and on-site audits and the trips are short trips. In order to keep this project manageable, the ASRP formulation in this thesis does not incorporate these as constraints in the ILP formulation. However, they will be used as a potential evaluation metric.

7.3 Routing Preferences

To end up with a schedule, as shown in Figure 7.4, the auditors will sometimes perform an audit earlier than its release date or after its due date as shown on Figure 7.4.

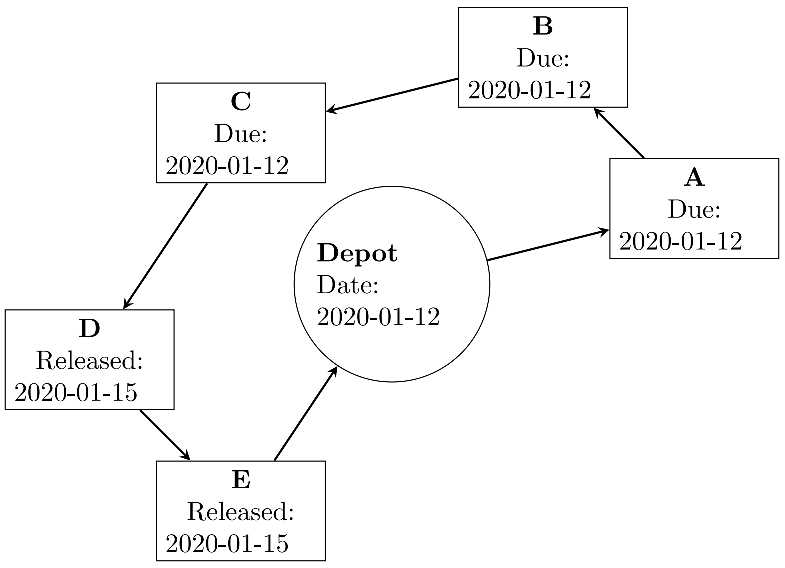


Figure 7.4: *An example of a route which violates the release dates of audit D and E.*

The figure above displays a route were the auditor performs audit D and E before their release dates to obtain a longer route. If the auditor were to respect the release dates, he would have to carry out the route over two days, as shown in Figure 7.5.

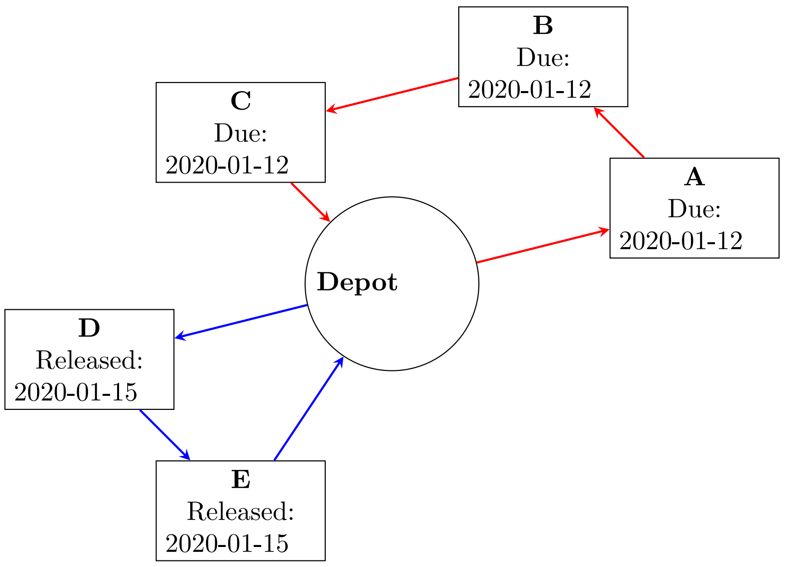
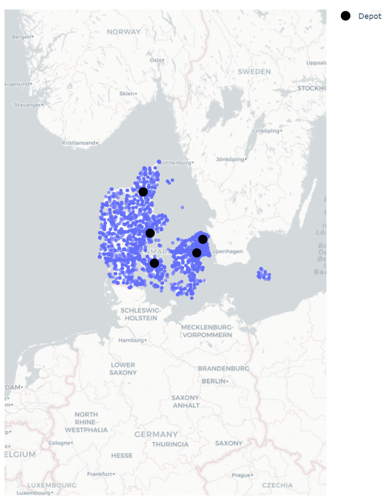


Figure 7.5: *An example of an auditor who splits up his route to respect the due dates and release dates.*

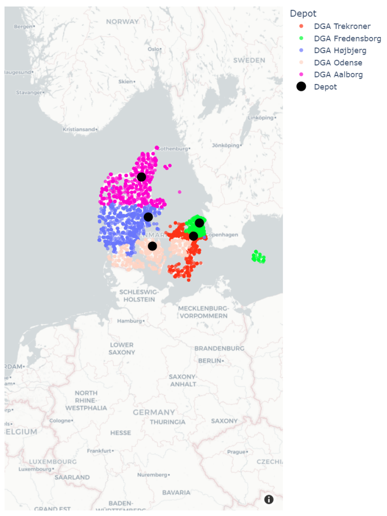
In Figure 7.5 each color represent a route on a different day, and the auditor performs the route over two days to respect the due dates and release dates. It is permitted to perform an audit outside of the audit window, but auditors must assess whether it is acceptable. This is usually done by looking at the audit priority and using their domain knowledge of the facility. This suggests that it is possible to represent the ASRP with soft audit windows. Therefore, for future work on the project, it might be worth exploring how soft audit windows affect the results of the simulation model.

7.4 Routing Decomposition

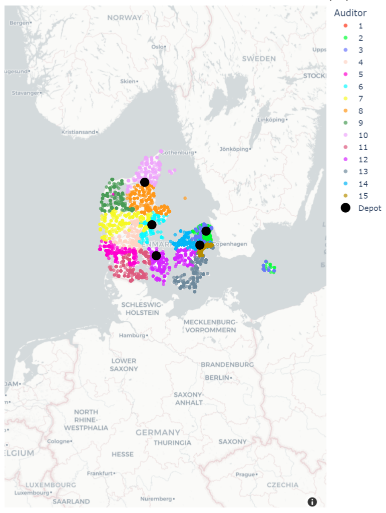
By implementing the heuristic in Figure 5.12 it is possible to visualize the current districts of each auditor. Moreover, since each auditor is located at a specific depot, it is also possible to visualize which facilities each depot is responsible for auditing. This is shown on Figure 7.6.



(a) All Facilities



(b) Depot Districts



(c) Auditor Districts

Figure 7.6: *How facilities can be distributed according to different district levels.*

As these visualizations demonstrate, the routing problem can be decomposed to three levels: district level, depot level, or global level. At the district level, the problem is solved for each auditor in their local district. In this report, this solution will be referred to as the *local* solution. At the depot level, the problem is solved for each depot and its auditors. This is referred to as the *semi-global* solution. Lastly, the problem can be solved globally such that the ILP formulation does not consider the current districts. The advantages and disadvantages of each approach are discussed in Sections 7.4.1, 7.4.2, and 7.4.3.

7.4.1 The Local Solution

The local solution would be the most computationally efficient approach since it would only have to compute the solution for a single auditor and a single depot. This approach would be easy to implement, since each auditor could run the solver locally on their workstation. However, it would most likely be the least effective solution. Moreover, there would need to be some way to assess vehicle availability and synchronize results across workstations. Otherwise, two auditors might risk using the same vehicle.

7.4.2 The Semi-Global Solution

The semi-global solution would still retain the problem as a single-depot problem; however, it would contain multiple auditors and would, therefore, be less computational efficient than the local solution. An advantage of the semi-global solution is that it eliminates the need for synchronizing vehicle availability across districts since each vehicle is attached exclusively to one depot. Moreover, the solution would probably be more effective than the local solution, but less effective than the global solution.

7.4.3 The Global Solution

The global solution represents a multi-depot problem and is, therefore, assumed to be the least computationally efficient solution. However, it is suspected that it will provide the most effective solution to the problem, since it does not need to adhere to any pre-defined districts. Therefore, the aim of this thesis will be to solve the problem at the global level, but, for future work on the problem, it will be relevant to compare the results of the global solution with both the local and the semi-global solution.

7.5 Audit Density

By using a Gantt chart it is possible to visualize the density of the audit windows, which is shown in Figure 7.7.

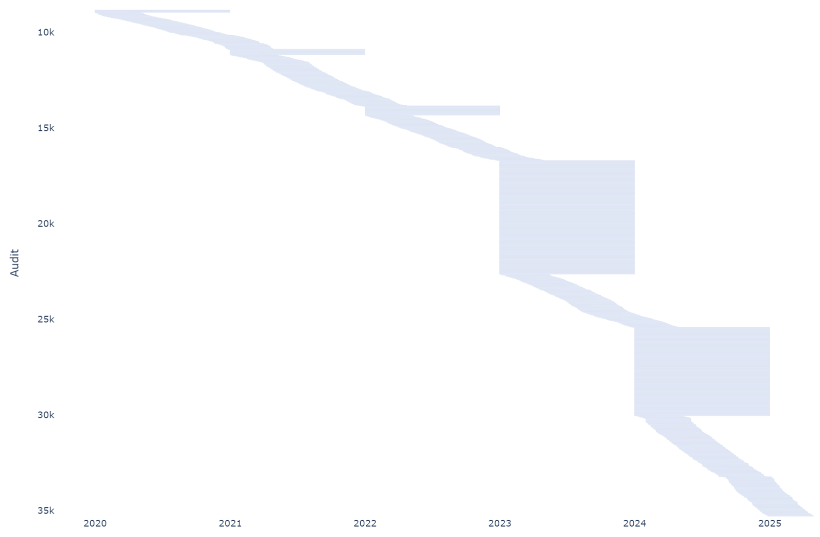


Figure 7.7: *A Gantt chart displaying the density of audit windows.*

In Figure 7.7 all audits are plotted along the y-axis and the x-axis displays the date. The time between the release of an audit and the due date determines the width of the bar. Only audits, which are released after 01-01-2020 and before 12-31-2024, are displayed. As the figure demonstrates, the audit windows follow a regular pattern where a large number of audits, with a yearly audit window, are released at the beginning of the year, while audits, with shorter audit windows, are spread out evenly during the rest of the year. This pattern arises from the distinction between desk audits, yearly on-site projects, and on-site audits. As noted in Sections 5.9 and 5.8.1 desk audits and on-site projects are released in batches the first of January, creating these "blocks" of audits with identical release and due dates each year. During the rest of the year, on-site audits are then released regularly based on audit priorities and facility types.

It should be noted that there seem to be substantially more desk audits and yearly on-site projects from 2023 onward. This pattern is the result of the following factors:

- As described in Section 5.8.1, each auditor only select a subset of the 1200 - 1500 available on-site projects. However, the data from 2023 and onward include all potential on-site projects, while data before 2022 only include projects which were actually performed.
- Since 2023, some types of desk audits have gone from being performed ad hoc to the method described in Section 5.9. This is part of the DGA's new regulatory strategy, and their aim is to perform substantially more desk audits than they did in previous years.

- Before 2023, some types of desk audits were performed as part of the on-site audit. They are, therefore, not registered as a standalone audit.
- When a facility's gambling license expires it is added as a desk audit; however, most facilities will renew their permission before an audit is necessary.

Since more desk audits are being released from 2023 onward, it must be assumed that the total workload from 2023 and onwards will be substantially higher, which is discussed in Section 7.6.

7.6 Workload and Capacity

The *workload*, which is defined as the total audit duration $\sum_{t \in T} \sum_{i \in V_t} p_i$, can be compared to the *total auditor capacity*, which is defined as $\sum_{t \in T} \sum_{e \in E} q_{et}$, for each year. This is done with a bar chart, as shown in Figure 7.8.

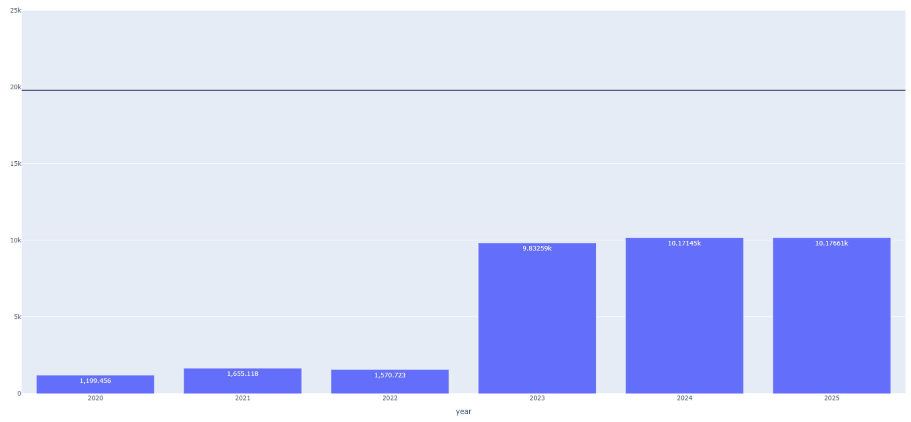


Figure 7.8: *A bar chart with the yearly workload and total auditor capacity.*

In Figure 7.8 the bars represent the yearly workload and the black horizontal line represents total auditor capacity. The total auditor capacity is assumed to be 6 hours a day 220 days a year for all 15 auditors. As the figure demonstrates, there is a substantial increase in the workload from 2023 onward. There are multiple reasons for this:

- Some types of desk audits used to be included in the on-site audit, therefore, the duration of on-site audits before 2023 is probably underestimated.
- Fewer audits were scheduled during the corona pandemic. Therefore, a backlog of audits had to be performed.
- Potential yearly on-site projects are included in 2023 and forward compared to 2022, where only scheduled on-site projects are included.

- Facilities that are expected to lose their gambling license from 2023 onward are included as potential audits.
- The DGA only started performing desk audits, in accordance with the process described in Section 5.9, from 2023. Therefore, more desk audits are included from 2023 onward.

It remains to be seen, which year best reflects the DGA's workload going forward. However, for this project, 2022 was selected as the year to solve, since it provides a small scale of the problem that can serve as preliminary results. Moreover, since 2022 only contains audits, which have been performed, it is possible to compare the result of the simulation model with how the DGA in practice solved the problem.

Auditors do have other tasks than auditing; therefore, there is a gap between the yearly auditor capacity and the yearly workload. However, it is unknown whether this gap can be attributed exclusively to the auditors' other tasks at the DGA. Other potential explanations could be that:

- The audit durations are underestimated.
- The data does not capture all the tasks associated with auditing.
- The current way of scheduling audits is ineffective.
- Auditors spend a substantial amount of time driving to and from audit facilities.

It is currently unknown whether any of these items contribute to the gap, but it would be relevant to explore this discrepancy for future work on the project.

7.7 Distribution of Durations

It is possible to get an overview of the distribution of audit durations with a histogram, as shown in Figure 7.9.

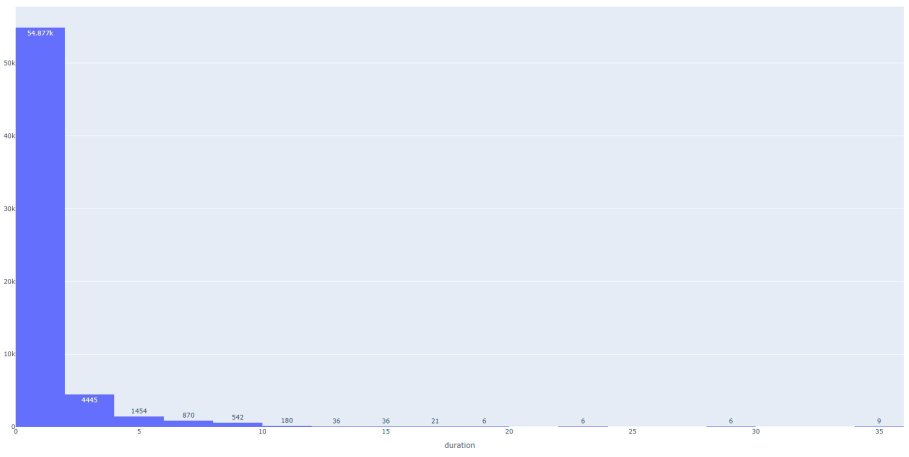


Figure 7.9: *A histogram displaying the distribution of audit durations.*

As can be seen on Figure 7.9, the audit distribution is heavily skewed towards short audits with the vast majority of audits taking less than two hours to process. However, there are some audits that take up to 35 hours to complete. These outliers are desk audits that can take up to a week to process. This can cause infeasible solutions if the time-horizon is discretized into days. This is avoided by splitting up audits, where $p_i > 8$, into 5-hour chunks, which is discussed in Section 9.1.

7.8 Scale of the Problem

With around 6500 facilities, 15 auditors, 5 depots, and between 2700-6000 audits released each year, the ASRP presents a large-scale optimization problem. This is further exacerbated by the on-site projects and desk audits, which have a year-long audit window. Therefore, the full scale of the problem is infeasible to solve with any known solver. Hence, to solve the problem, it is necessary to implement heuristics or decompose the problem. For this project, the second option was selected by using a *rolling time-horizon*.

A rolling time-horizon is a common approach to solving large-scale optimization problems that incorporate time-spans [17]. The reasoning behind this approach is to exploit the underlying time-structure when decomposing the problem [17]. This is done by repeatedly modeling and solving the problem for each step in the time-horizon until the problem is solved or infeasibility is detected [17]. This approach can lead to high-quality solutions for problems that would otherwise be unsolvable [17]. Therefore, this project uses a rolling time-horizon by decomposing the problem into a daily problem and solving it each day. This should provide a high-quality solution which is more simple to implement than any specialized heuristics.

7.9 Conclusion

The master problem can be decomposed to three subproblems: *audit creation*, *routing*, and *audit scheduling.* Audit creation is concerned with inserting new audits; deciding which facilities to audit; and determining appropriate audit windows. Since an optimizer or a machine learning algorithm could take facility history; fairness; and previous infractions into account, when selecting facilities to audit, this subproblem was identified as a prime candidate for future work at the DGA. However, since the DGA manually solves this problem and to keep the scope of this project limited, it was assumed that this subproblem is solved. Therefore, this thesis deals exclusively with the routing and the audit scheduling subproblems.

Through semi-structured interviews with the auditors, it was revealed that they prefer to drive long routes a few times each week rather than short daily routes. Moreover, unless they are back at the depot earlier than expected, they prefer to only perform on-site or desk audits on a given day. Even though these preferences are not directly considered in the simulation model, they can serve as useful evaluation metrics when comparing results or as constraints in a future ILP formulation.

In order to achieve this schedule, an auditor will sometimes perform an on-site audit earlier or later than its due date or release date. This suggests that the ASRP can be solved as a problem with soft release and due dates. However, to keep the project manageable, this thesis will only tackle the problem with hard release and due dates.

It is possible to visualize, what district each facility is attached to, with a scatter map. Furthermore, since all auditors are located at a specific depot, it is also possible to visualize which depot each facility is attached to. These scatter maps are shown in Figure 7.6 and demonstrate that the problem can be solved at three levels: the district level, which in this report is named the *local* solution; the depot level, which is named the *semi-global* solution; or the global level, which is named the *global solution.*

The local solution is presumably the most computationally efficient approach since it would only require to compute the solution for a single auditor and a single depot. Moreover, it is also relatively easy to implement, since a solver could run locally on each auditor's workstation. However, it might not provide the most effective solution, and there would be a need to synchronize vehicle availability across auditor workstations to ensure that auditors always use an available vehicle.

The semi-global solution retains the problem as a single-depot problem, but solves it for multiple auditors. Therefore, it is assumed to be less computationally efficient than the local solution but more efficient than the global solution. Compared to the local solution, the semi-global solution does not need to synchronize vehicle availability across different work-stations, since each vehicle is tied exclusively to one depot. In terms of quality, the semi-global solution is assumed to be more effective than the local solution, but less effective than the global solution.

The global solution solves the problem as a multi-depot problem and is, therefore, assumed to be less computationally efficient than the local and semi-global solution. However, since this solution does not have to adhere to any predefined districts, it might offer a more effective solution. Therefore, the aim of this thesis is to solve the problem globally. Still, for future work on the project, it will be noteworthy to compare the results of the global solution with both the local and semi-global solution.

Figure 7.7 illustrates that audit windows follow a regular pattern in which a large number of audits, with year-long audit windows, are released at the beginning of each year. During the remainder of the year, audits, with a shorter audit-window, are then released with a regular interval. This is due to the distinction between on-site audits, which are released regularly throughout the year; on-site projects, which are released at the start of each year; and desk audits, which are released in batches each year. This chart also shows that there are substantially more on-site projects and desk audits from 2023 onward. This is because 2023 and onward include all potential audits, while 2022 only include performed audits. In addition, some types of desk audits used to be performed as part of the on-site audit and are, therefore, not included in the chart. Lastly, there is the fact that the DGA has increased their focus on desk audits and are, therefore, planning to perform substantially more of these.

From Figure 7.8 it was observed that the workload increases from 2023 and onward. This is due to the fact that the durations of on-site audits are probably underestimated before 2023, since they also included desk audits. Furthermore, the corona pandemic left a backlog of audits, which needed to be performed. Lastly, only performed audits are included before 2023, while 2023 and onward also contain potential audits. It was decided to solve the problem for 2022, since it would provide a small scale of the problem and be easy to compare the results of the simulation model with how the DGA solved the problem in practice. There is a substantial gap between the workload, which is planned each year, and the total auditor availability. This suggests that the current data does not properly capture the auditors' tasks; that their current scheduling method is ineffective; or that the auditors spend a substantial amount of time driving between audit facilities.

Figure 7.9 illustrates that the vast majority of audits have a duration of two hours or less. However, there are some outliers that last up to 35 hours. Therefore, these audits are divided into 5-hour chunks to avoid infeasibility.

Since the DGA needs to consider around 6500 facilities, 5 depots, 15 auditors, and a year long time-horizon; the problem is simply too large to be solved with a traditional solver. Therefore, it is necessary to either implement heuristics or a rolling time-horizon. For this thesis, the second option was selected since it provides a simple yet effective way of providing high-quality solutions to large-scale optimization problems.

Chapter 8

Integer Linear Programming Formulation

This chapter presents and discusses the ILP formulation of the ASRP. Sections 8.1, 8.2, and 8.3 recaps the mathematical notation, while Sections 8.4, 8.5, 8.6, and 8.7 present the constraints; the objective of the model; the ILP formulation; and provide a description of the model. Section 8.8 introduces an alternative formulation of the due date constraint which, in retrospect, reduces the likelihood of infeasibility. Section 8.9 discusses the objective function, while Section 8.10 discusses how subtours are eliminated.

8.1 Sets

- Let T be a set of days and let $t \in T$ represent the current date.
- Let the problem be represented as a network $N = (V_t, A)$
- Let V_t be a set of vertices which represents available audits on day t.
- Let O_t be a subset of V_t that represents available on-site audits on day t.
- Let L be the set of depots.
- Let $A = \{(i, j) \quad \forall (i, j) \in O_t \cup L\}$ be a set of directed arches which represents feasible roadways between the vertices
- Let E be the set of auditors.

8.2 Parameters

- Let each arch $(i, j) \in A$ have an associated travel cost (time in hours) $c_{ij} \in \mathbb{R}_0^+$ of travelling from vertex i to j.

- Let each audit $i \in V_t$ have an associated due date $d_i \in \mathbb{Z}_0^+ \wedge d_i \geq t$.
- Let $u_i = d_i - t$ represent the difference between the due date and the current date for all audits $i \in V_t$.
- Let $q_{et} \in \mathbb{R}_0^+$ represent the capacity of auditor $e \in E$ on day t.
- Let $k_{lt} \in \mathbb{Z}_0^+$ represent the number of vehicles available at depot $l \in L$ on day t.
- Let $p_i \in \mathbb{R}_0^+$ represent the audit duration in hours for audit $i \in V_t$.
- Let $g_{ie} = \begin{cases} 1 \text{ if auditor } e \in E \text{ can perform audit } i \in V_t \\ 0 \text{ Otherwise} \end{cases}$
- Let $b_{el} = \begin{cases} 1 \text{ if auditor } e \in E \text{ belongs to depot } l \in L \\ 0 \text{ Otherwise} \end{cases}$

8.3 Decision Variables

- Let $a_e = \begin{cases} 1 \text{ if auditor } e \in E \text{ is assigned to any on-site audit } i \in O_t \\ 0 \text{ Otherwise} \end{cases}$
- Let $x_{ije} = \begin{cases} 1 \text{ if arch } (i,j) \in A \text{ is traversed by auditor } e \in E \\ 0 \text{ Otherwise} \end{cases}$
- Let $y_{ie} = \begin{cases} 1 \text{ if audit } i \in V_t \text{ is assigned to auditor } e \in E \\ 0 \text{ Otherwise} \end{cases}$

8.4 Constraints

The model must respect the following constraints:

- All auditors must leave and return to their designated depot b_{el}, and they can only leave their depot if they have been assigned to an on-site audit.
- There cannot be more auditors leaving for on-site audits than the vehicle capacity k_{lt} at depot $l \in L$.
- An auditor must be qualified to perform the audit; therefore, if $y_{ie} = 1$, then $g_{ie} = 1 \ \forall e \in E, \forall i \in V_t$.
- An audit cannot be performed more than once.
- Auditors can only visit vertices that they have been assigned to audit.
- The combined travel cost and audit duration cannot exceed an auditor's capacity q_{et}.
- An audit cannot be performed later than its due date d_i.

8.5 Objective

The objective of the ASRP is to carry out as many audits as possible. The basic assumption is that the workload on day t might be higher than the total auditor capacity. Therefore, the critical decision is not only the sequencing and assignment of audits to auditors, but also the selection of audits to perform and postpone.

8.6 Model

$$\text{Max} \sum_{i \in V_t} \sum_{e \in E} y_{ie} \frac{1}{u_i} \tag{8.1}$$

Subject to:

$$\sum_{i\in O_t} x_{lie} = b_{el}a_e \qquad \forall e \in E, \forall l \in L \tag{8.2}$$

$$\sum_{i\in O_t} x_{ile} = b_{el}a_e \qquad \forall e \in E, \forall l \in L \tag{8.3}$$

$$\sum_{e\in E}\sum_{i\in O_t} x_{ile} \le k_{lt} \qquad \forall l \in L \tag{8.4}$$

$$\sum_{e\in E}\sum_{i\in O_t} x_{lie} \le k_{lt} \qquad \forall l \in L \tag{8.5}$$

$$g_{ie} \ge y_{ie} \qquad \forall i \in V_t, \forall e \in E \tag{8.6}$$

$$\sum_{e\in E} y_{ie} \le 1 \qquad \forall i \in V_t \tag{8.7}$$

$$\sum_{j\in O_t\cup L} x_{ije} = y_{ie} \qquad \forall i \in O_t, \forall e \in E \tag{8.8}$$

$$\sum_{j\in O_t\cup L} x_{jie} = y_{ie} \qquad \forall i \in O_t, \forall e \in E \tag{8.9}$$

$$\sum_{i\in O_t\cup L}\sum_{j\in O_t\cup L} c_{ij}x_{ije} + \sum_{i\in V_t} p_i y_{ie} \le q_{et} \qquad \forall e \in E \tag{8.10}$$

$$y_{ie} \le a_e \qquad \forall e \in E, \forall i \in O_t \tag{8.11}$$

$$u_i \ge \sum_{e\in E} y_{ie} \qquad \forall i \in V_t \tag{8.12}$$

$$\sum_{i\in O_t\cup L} x_{ije} - \sum_{i\in O_t\cup L} x_{jie} = 0 \qquad \forall e \in E, \forall j \in O_t \cup L \tag{8.13}$$

$$\sum_{i\in S}\sum_{j\in S, j\neq i} x_{ije} \le |S| - 1 \qquad (S \subset O_t, 2 \le |S|), \forall e \in E \tag{8.14}$$

$$x_{ije} \in \{0,1\} \qquad \forall e \in E, \forall i \in O_t \cup L, \forall j \in O_t \cup L \tag{8.15}$$

$$y_{ie} \in \{0,1\} \qquad \forall e \in E, \forall i \in V_t \tag{8.16}$$

$$a_e \in \{0,1\} \qquad \forall e \in E \tag{8.17}$$

8.7 Model Description

- **Equation 8.1:** The objective function, which seeks to maximize the number of audits performed. The underlying assumption is that to perform as many audits as possible, the routing and scheduling must be as effective as possible. The coefficients of the function are the fractional values $\frac{1}{u_i}$, as it ensures that audits, which are closer to their due dates, are prioritized.
- **Equation 8.2 - 8.3:** Auditors can only leave once for on-site audits from their designated depot and must always return to their designated depot. Additionally,

auditors can only leave if assigned to an on-site audit.

- **Equation 8.4 - 8.5:** The number of auditors leaving for on-site audits must be less than or equal to the number of vehicles available at the depot.
- **Equation 8.6:** An auditor must be able to perform an assigned audit.
- **Equation 8.7:** Audits cannot be performed more than once.
- **Equation 8.9 - 8.8:** Links the routing variable with the assignment variable. This ensures that the auditors only visit on-site facilities that they have been assigned to audit.
- **Equation 8.10:** The total travel cost and the total duration of audits for both on-site and desk audits must be less than or equal to the capacity of each auditor. This constraint allows an auditor to be assigned to both desk audits and on-site audits on the same date. The underlying idea being that if an auditor have been assigned to both on-site and desk audits, he will perform the on-site route first and then perform the desk audits when returning to the depot. This goes somewhat against the auditors' scheduling preferences; however, as discussed in Section 7.2, it was decided that this would not be a priority in this report.
- **Equation 8.11:** If an auditor is assigned to an on-site audit, the variable for any on-site audit assignment a_e becomes 1.
- **Equation 8.12:** Makes the model infeasible if the due date is less than the current date. After testing the simulation model, an alternative formulation was discovered, which is discussed in Section 8.8.
- **Equation 8.13:** Flow conservation constraint.
- **Equation 8.14:** Subtour elimination constraint.
- **Equation 8.15 - 8.17:** Binary constraints.

8.8 Alternative Due Date Constraint

Equation 8.12 presents a constraint which ensures that all solutions become infeasible if $u_i < 0$. While running the simulation model for 2022 this never happened; however, as is noted in Section 7.6, the workload increases from 2023 and forward. Therefore, some audits can potentially be postponed until $d_i < t$, resulting in infeasibility. Moreover, even if an audit is postponed until $d_i = t$ it will still create an error, since it will result in $u_i = 0$, which is not an appropriate denominator in the objective function. To avoid this, a better method of handling due dates is needed, which is presented in this section. This due date constraint ensures that $y_{ie} = 1$ if $d_i = t + 1$, which means that all audits must be scheduled at least one day before their due date. It is presented in Equation 8.18.

$$d_i + \sum_{e \in E} y_{ie} \geq t + 2 \quad \forall i \in V_t \tag{8.18}$$

For this inequality to be valid there are three cases, which must be satisfied, when scheduling audits:

- $d_i > t + 1$ in which case $\sum_{e \in E} y_{ie} \geq 0$.
- $d_i = t + 1$ in which case $\sum_{e \in E} y_{ie} = 1$.
- $d_i < t + 1$ in which case the solution must be infeasible.

All three cases are guaranteed to be fulfilled by Equation 8.18, which can be demonstrated by a concrete example.
For the first case, let $d_i = 6$ and $t = 4$ which gives the following inequality:

$$6 + \sum_{e \in E} y_{ie} \geq 4 + 2$$

This can be rewritten to:

$$\sum_{e \in E} y_{ie} \geq 0$$

Therefore, the first case is satisfied.
For the second case let $d_i = 5$ and $t = 4$ which gives the following inequality:

$$5 + \sum_{e \in E} y_{ie} \geq 4 + 2$$

This can be rewritten to:

$$\sum_{e \in E} y_{ie} \geq 1 \tag{8.19}$$

Since Constraint 8.7 states that $\sum_{e \in E} y_{ie}$ cannot be greater than 1, and Constraint 8.16 states that y_{ie} must be either 0 or 1, an audit has to be scheduled to provide a feasible solution.
For the last case, let $d_i = 4$ and $t = 4$ which results in the following inequality:

$$4 + \sum_{e \in E} y_{ie} \geq 4 + 2$$

This can be rewritten to:

$$\sum_{e \in E} y_{ie} \geq 2$$

Therefore, the third case is satisfied since the solution will always be infeasible. Although Constraint 8.18 seems like a better formulation, it was, due to time limitations, not possible to test it. However, for further work on the project, it will be interesting to investigate what effect, if any, this new formulation will have on the solution.

8.9 Objective Function

The objective function, presented in Equation 8.1, is unusual for a scheduling and routing problem, since it aims to schedule as many audits as possible. Most routing problems seek to minimize some sort of cost such as the total distance traveled or the total CO2 emissions [19]. Even for a routing and scheduling problem, such as the TSRP, this is still an unusual objective, as the goal is not to optimize any particular quality of service, such as minimizing lateness or earliness [39].

The reason behind this objective function is that the model will be forced to generate efficient routes to perform as many audits as possible. However, it is assumed that the routing will only be effective if the model has to postpone some audits. Therefore, to design effective routes, the total workload should be greater than or slightly below the total auditor capacity on day t. If this is not the case, it should be more effective to directly minimize the travel cost with the following objective function:

$$\text{Min} \sum_{i \in O_t \cup L} \sum_{j \in O_t \cup L} \sum_{e \in E} c_{ij} x_{ije} \tag{8.20}$$

If this objective function is used Constraint 8.7 should be tightened to ensure that all audits are scheduled on day t:

$$\sum_{e \in E} y_{ie} = 1 \quad \forall i \in V_t \tag{8.21}$$

This alternative formulation was not tested. However, for future work on the project, it could be used in the simulation model on days when the total workload is substantially below the total auditor capacity.

8.10 Subtour Elimination

When solving routing problems, all visited vertices must form a coherent route, known as a *Hamiltonian circuit* [5]. A Hamiltonian circuit always starts and ends at the same vertex and visits all vertices once, as shown in Figure 8.1.

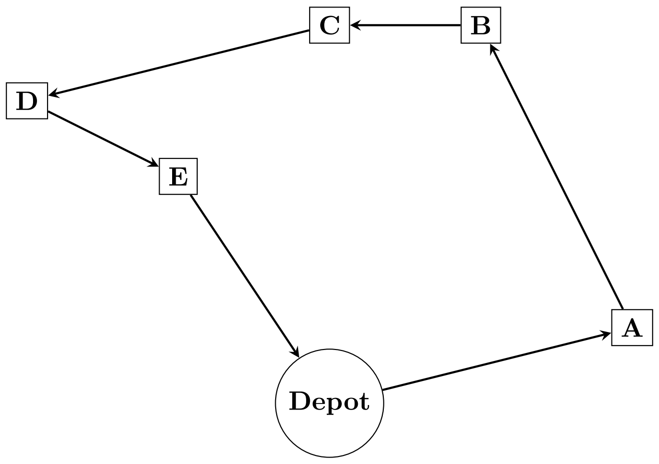


Figure 8.1: *A route which forms a Hamiltonian circuit.*

To ensure this, most routing formulations employ *subtour eliminations constraints* (SEC). If these constraints are not added, the model is likely to generate disconnected routes, as shown in Figure 8.2.

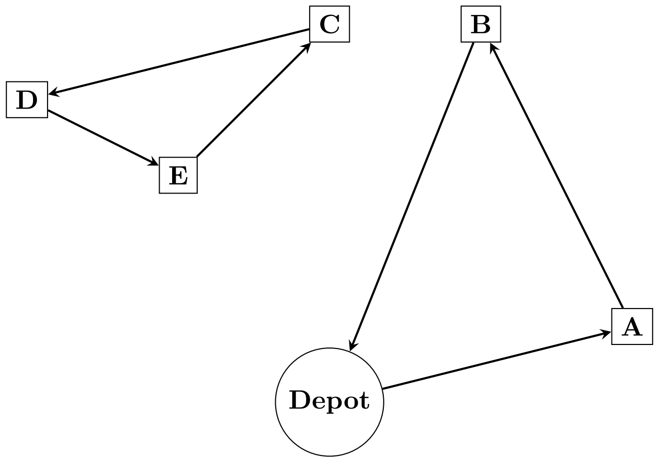
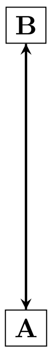


Figure 8.2: *A route which does not form a Hamiltonian circuit.*

Numerous SECs have been proposed and it is not clear which SEC provides the tightest formulation [5]. For this report the *Dantzig-Fulkerson-Johnson* (DFJ) formulation was selected since it is one of the most commonly used formulations [5]. The DFJ formulation was originally presented by G. Dantzig, D. Fulkerson, and S. Johnson when they solved an instance of the TSP with 49 cities [11]. The DFJ subtour elimination procedure eliminates subtours by generating all proper subsets $S \subset O_t$ that include at least two vertices. This is demonstrated in Figure 8.3.



$$S = \{(A, B), (B, A)\}$$

Figure 8.3: *Subtour between A and B.*

The number of traversed edges in S is then constrained to be less than or equal to $|S| - 1$ as demonstrated on Figure 8.4.

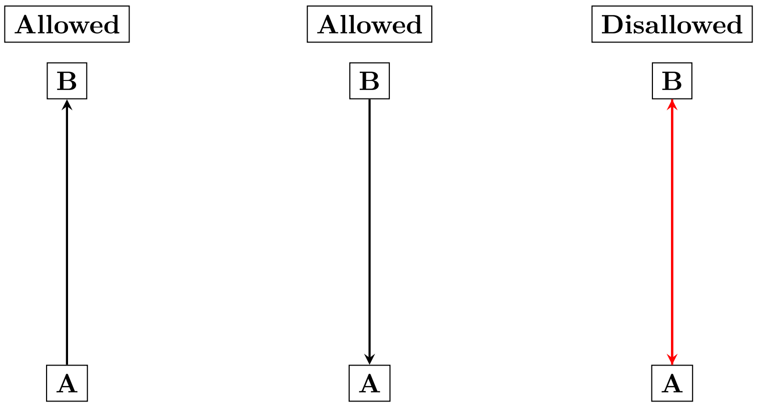


Figure 8.4: *Allowed routes after subtour elimination.*

Since all subsets of O_t are generated and added as constraints, the number of constraints generated is $2^{|O_t|}$ which is computationally inefficient [11]. Therefore, a cutting-plane approach, which was also proposed by Dantzig, Fulkerson, and Johnson [11], was used. In this approach, a relaxed version of the problem is solved. The solution is then checked for subtours. All detected subtours are then added as constraints to the model, and the problem is resolved. This process is performed iteratively until the solution contains no subtours.

To reduce the number of cuts added in the subtour elimination procedure, a disconnected network graph is used so that there are only edges between on-site audits.

This ensures that the subtour elimination procedure is not performed on desk audits since these audits are always performed at the depot, and, therefore, do not require any routing.

8.10.1 Alternative SECs

The SEC on Equation 8.14 applies the cut to each auditor. However, an alternative version, which in this report is named the *aggregated DFJ* (ADFJ), was tested. This SEC is similar to the one presented in Section 8.10, however, rather than applying the cut for each auditor, the cut is applied across the sum of all auditors:

$$\sum_{e \in E} \sum_{i \in S} \sum_{j \in S, j \neq i} x_{ije} \leq |S| - 1 \quad (S \subset O_t, 2 \leq |S|) \tag{8.22}$$

Even though this SEC seems similar to the one presented in the previous section it turned out during testing that it was substantially less efficient. Where the SEC from the previous section was able to solve most days within minutes, the ADFJ would spend up to 1.5 hours solving a single day. Therefore, this SEC was deemed unsuitable for the problem at hand.

In addition to the DFJ, another common approach to subtour elimination is the *Miller-Tucker Zemlin* (MTZ) formulation. This formulation was originally formulated in 1960 by Miller, Tucker, and Zemlin as an SEC for the traveling salesman problem [26]. The basic idea behind the MTZ formulation is to introduce new decision variables, which in this thesis will be denoted as $s_i \quad \forall i \in O_t$, to enumerate all vertices, excluding depots, such that $s_j > s_i$ when $x_{ije} = 1$ [6]. To demonstrate how a MTZ SEC could be used in this ILP formulation, let Q denote the maximum value of q_{et} on day t, that is, $Q = max(\{q_{et} \mid e \in E\})$, and let s_i denote the enumeration variable for all $i \in O_t$. The MTZ can then be formulated the following way:

$$s_j - s_i \geq p_j - Q(1 - x_{ije}) \qquad \forall e \in E, \forall j \in O_t, \forall i \in O_t \tag{8.23}$$

$$s_i \geq \sum_{e \in E} p_i y_{ie} \qquad \forall i \in O_t \tag{8.24}$$

$$s_i \leq Q \qquad \forall i \in O_t \tag{8.25}$$

This SEC eliminates subtours by enumerating all nodes such that $s_j \geq s_i + p_j$ when $x_{ije} = 1$. If auditor e performs audit i and j, then $x_{ije} = 1$ and Constraint 8.23 can be rewritten to $s_j \geq p_j + s_i$, which ensures that s_j is greater than s_i [1]. If auditor e does not perform audit i and j then $x_{ije} = 0$ and 8.23 can be rewritten as $s_j - p_j \geq s_i - Q$. Since $s_i \leq Q$ and $s_j \geq p_j$ Constraint 8.23 will still be valid [1]. The decision variable y_{ie} is included in Constraint 8.24 such that unvisited audits can be enumerated as 0. This SEC has not been implemented and tested. However, for future work on the project, it would be relevant to research if an MTZ formulation can outperform a DFJ formulation in terms of computational efficiency.

8.11 Conclusion

This ILP formulation proposes an unusual objective function, which seeks to schedule as many audits as possible. The basic assumption being that to schedule as many audits as possible, the model needs to find the most effective routes. Therefore, the critical decision is not only to find the most effective routes, but also to determine which audits to perform and which audits to postpone. Since the coefficients in the objective function are fractional values, where the denominator is u_i, the model should tend to prioritize audits that are closer to their due date.

For the routes to be effective, there must be more audits available than the total capacity of the auditors; otherwise, it is assumed that the routes will be more effective if the travel cost is directly minimized in the objective function.

The ILP formulation allows an auditor to perform both on-site and desk audits on the same date. The assumption is that an auditor will first perform on-site audits and then perform desk audits when returning to the depot. This goes against the auditors' scheduling preferences; however, as discussed in Section 7.2, it was decided not to be a priority in the report.

The ILP formulation, which was tested in this thesis, only ensures that the solution is infeasible if $d_i > t$. However, an alternative formulation was presented that forces the model to schedule an audit if $d_i = t + 1$. This formulation was not tested, but it is assumed to reduce the likelihood of infeasibility.

To eliminate subtours, a cutting-plane version of the Dantzig-Fulkerson-Johnson (DFJ) formulation was used. This subtour elimination constraint (SEC) was proposed back in 1954 and is still one of the most common approaches to subtour elimination. With the DFJ cutting-plane approach, a relaxed version of the problem is solved. The solution is then checked for any subtours. All subtours in the solution are then added to the model as constraints, and the problem is resolved. This is done iteratively until the solution contains no subtours. To reduce the number of added cuts, a disconnected graph was used, as this ensures that only subtours between on-site routes are considered in the procedure.

Since there are multiple ways to eliminate subtours, another SEC, named the aggregated DFJ (ADFJ), was tested. This SEC is almost identical to the DFJ described above, but, rather than applying the cut for each auditor, the cut is added across the sum of all auditors. However, during testing, it was considerably slower and was therefore discarded as a potential SEC for the problem.

Another SEC, named the Miller-Tucker-Zemlin (MTZ) formulation, was also outlined. This SEC eliminates subtours by enumerating all nodes such that $s_j \geq s_i + p_j$ when $x_{ije} = 1$. This SEC was not tested, and it is unknown whether it can outperform the DFJ formulation.

Chapter 9

Simulation Modeling

This chapter describes how 2022 was simulated using a simulation model. Section 9.1 presents how audits, which last more than 8 hours, are represented in the model. Sections 9.2 and 9.3 detail how audits and Bornholm and Greenland are scheduled, while Section 9.4 presents how vacation days are incorporated into the simulation. Section 9.5 introduces the hyper-parameters ϕ and ϵ, which determine the distribution of release dates. Section 9.6 describes how the simulation model is implemented. Lastly, Sections 9.7 and 9.8 describe the input and output data, respectively.

9.1 Multi-Day Audits

Multi-day audits are defined as audits where $p_i > 8$. All multi-day audits are repeatedly inserted as 5-hour audits, until the remaining audit duration is less than 5 hours, as shown in Figure 9.1.

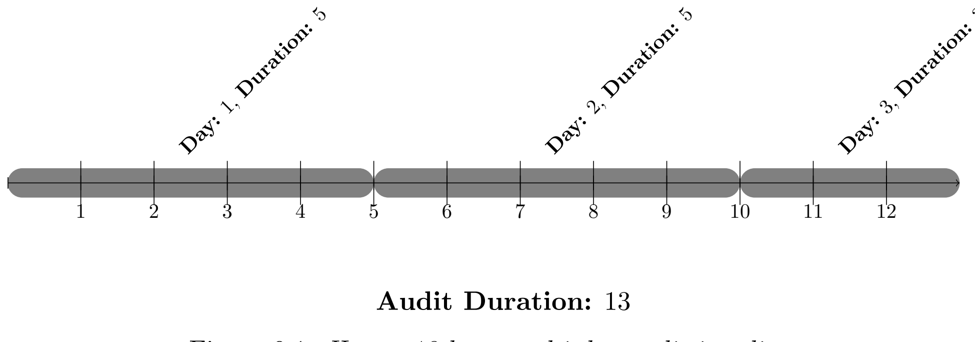


Figure 9.1: *How a 13-hour multi-day audit is split up.*

As Figure 9.1 shows, the 13-hour audit is inserted as a 5-hour audit on day 1 and day 2, while it is inserted as a 3-hour audit on day 3. To avoid having multiple auditors working on the same audit, the auditor, who was first assigned to the audit, is designated as the only auditor capable of handling the audit in the accomplice matrix.

9.2 Greenland

Greenland is not part of any district, as the DGA prefers to schedule these audits manually. This is usually done by setting aside a week once a year for two auditors to visit as many facilities as they can in this region. This was modeled by first removing all audits that were located in Greenland. Auditor 10 and auditor 11, who have previously audited Greenland facilities, were then set as unavailable on all times-slots from 01-31-2022 until 02-07-2022. The assumption is that these auditors will perform as many audits in Greenland as possible in this period.

9.3 Bornholm

Unlike Greenland, Bornholm is divided according to districts; however, auditors, with districts in Bornholm, usually set aside one or two days once a year for handling those audits, regardless of their due and release date. Therefore, this was modeled in a similar way to audits in Greenland. All auditors, who have facilities in Bornholm, in this case auditors 2 and 3, were set as unavailable on all time slots from 01-31-2022 until 02-02-2022. The assumption being that they will spend this time performing audits on Bornholm.

9.4 Vacations

Since it was not possible to obtain the exact vacation dates of the auditors, it was decided to give each auditor 42 uninterrupted days of vacation. During their vacation, the auditor capacity is 0. The vacation dates was the following:

- **Auditor 1 - 5:** From 06-24-2022 until 08-05-2022.
- **Auditor 6 - 10:** From 06-30-2022 until 08-11-2022.
- **Auditor 11 - 15:** From 09-08-2022 until 10-20-2022.

It will, of course, be necessary to implement a more sophisticated method for generating holidays. This can be done by either gathering the data directly from the auditors' outlook calendars or by using historical data to get a more accurate representation of how auditors prefer to use their vacations. However, for preliminary testing, this assignment of vacations was deemed sufficient.

9.5 Distribution of Release Dates

A significant challenge is the fact that all yearly desk audits and on-site projects are released at the same time, which was discussed in Section 7.5. This gives the following distribution of how audits are released.

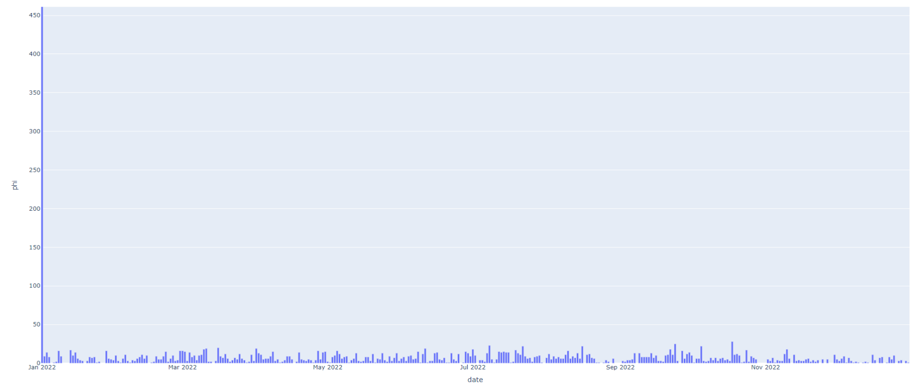
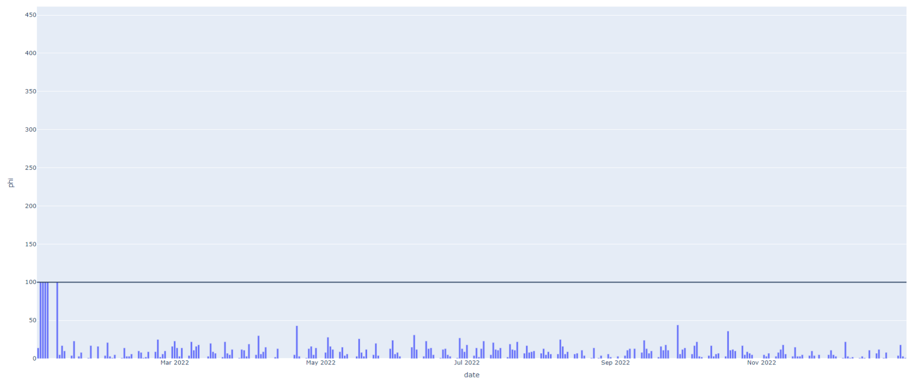


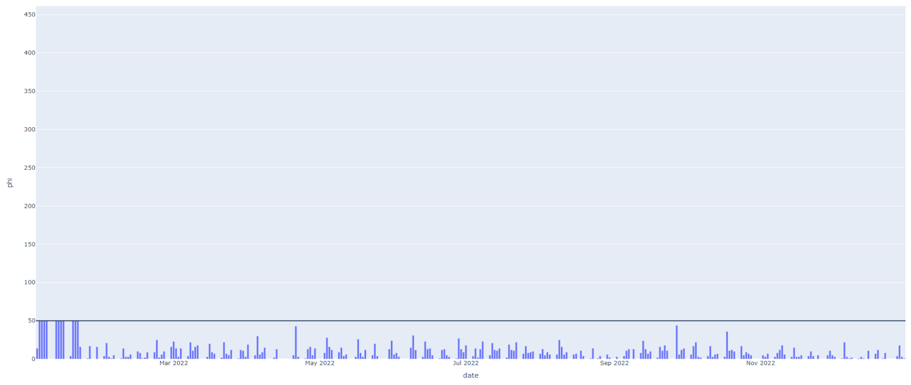
Figure 9.2: *How audits are released during 2022.*

As Figure 9.2 shows, there is a 'bottleneck at the start of the year where more than 400 audits are being released. Due to limited computational resources, it was decided, therefore, to create a hyper-parameter ϕ, which determines how many audits can be released each day when initializing the model. During initialization, only ϕ audits are released each day. To demonstrate this, let R_t denote the set of audits released on day t. If $|R_t|$ is greater than ϕ on day t, then $|R_t| - \phi$ audits will have their release date moved to the nearest work day. Furthermore, a parameter named ϵ was introduced, which determines how close to its due date an audit can be released. The function for redistributing the release dates is named $relax\_realease\_dates(\epsilon, t_{start}, t_{stop}, \phi, \mathcal{D})$. $\mathcal{D}$ is a data frame of the same form as the one discussed in Section 6.2.13, while t_{start} and t_{stop} refer to the ID of the first date and the last date to which the function should be applied.

With ϕ and ϵ it is possible to test different distributions of release dates. A higher value of ϕ will result in a distribution that more closely resembles the original distribution presented in Figure 9.2.



(a) $\phi = 100$



(b) $\phi = 50$

Figure 9.3: *How audits are released in 2022 when* $\phi = 100$ *(a) and when* $\phi = 50$ *(b).*

In Figure 9.3 the black horizontal line indicates the value of ϕ and on both figures $\epsilon = 5$. As can be seen in the figure, a higher value of ϕ changes the distribution so that more audits are released earlier in the year. However, if the value of ϕ is low, the distribution will be the opposite of that shown in Figure 9.2.

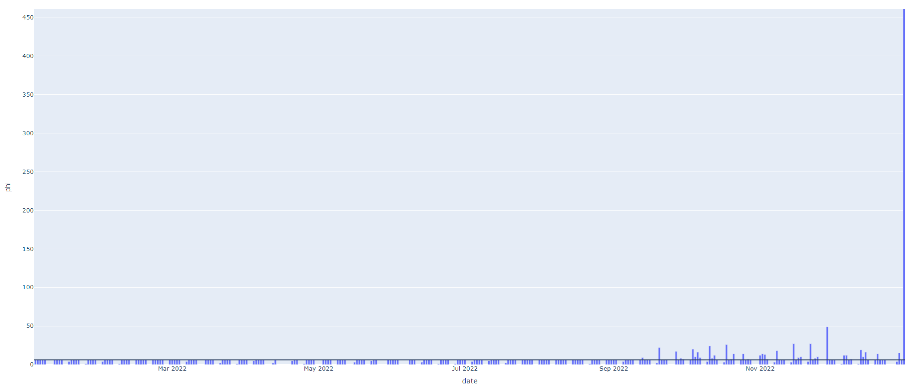


Figure 9.4: *How audits are released during 2022 when $\phi = 6$ and $\epsilon = 5$.*

What happens in Figure 9.4 is that only 6 audits are released each day until the last days of the year when most desk audits are released 5 days before their due date. For preliminary testing, the simulation model was initialized with $\phi = 15$ and $\epsilon = 5$. This resulted in the following distribution of released audits.

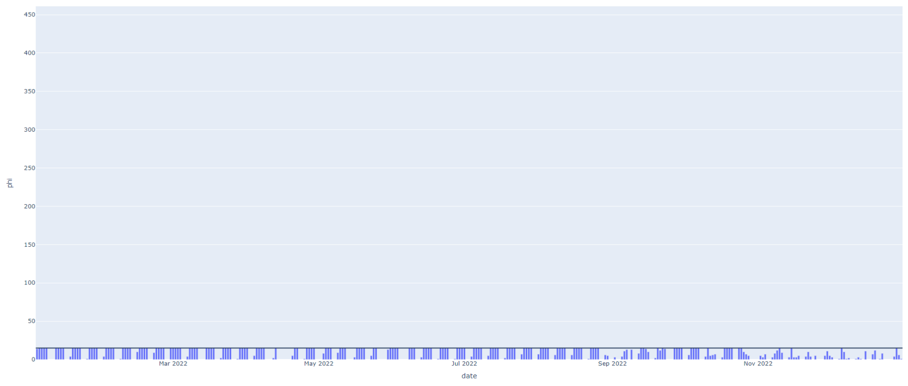


Figure 9.5: *How audits are released during 2022 when $\phi = 15$ and $\epsilon = 5$.*

The distribution on Figure 9.5 was selected for preliminary testing, since it provides a uniform distribution, which should be easy to model and solve. However, for future work on this project, ϕ is an important hyper-parameter, as a higher value of ϕ should force the simulation model to design more effective schedules and routes.

9.6 The Simulation Model

The simulation model in this thesis belongs to a class of models named *Discrete-event simulations* [24]. In a discrete-event simulation, each event occurs in time t and the intervals between each event are fixed [24]. In each event, the state variable changes instantly [24]. These types of simulations use a *simulation clock* to keep track of events [24]. Usually, the simulation clock is a *next-event time advance clock*, where a start time t_{start} and a stop time t_{stop} are defined [24]. The current day t is then initialized as $t = t_{start}$, and, after each event, t is incremented to the nearest time step $t = t + 1$. This is repeated n times until $t + n > t_{stop}$ [24].

In the simulation model used in this thesis, each time t represents a day in 2022 and the event is that V_t changes so V_t includes all released audits, which were not scheduled, where $r_i \leq t$. Therefore, $V_t = \{i \mid i \in R_{t'} \; \forall \; t' \in T \;\wedge\; r_i \leq t \;\wedge\; \mathcal{C}_i = -1\}$. An example of this process, where $t_{start} = 0$, is illustrated in Figure 9.6.

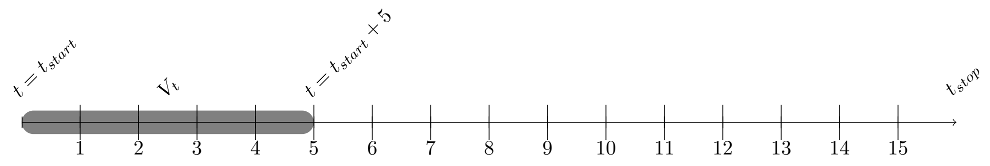


Figure 9.6: *How V_t contains unscheduled audits, and how t is incremented*

Discrete-event simulation models have been applied to a wide variety of real-world systems, and can, therefore, take many forms [24]. However, they generally employ the following routines:

- *Initialization routine:* A subprogram that initializes the model [24].
- *Timing routine:* A subprogram that advances the simulation clock [24].
- *Event routine:* A subprogram that updates the state variables when an event occurs [24].
- *Library routines:* A subprogram that is used to determine the data distribution in the simulation model [24].
- *Report generator:* A subprogram that collects data from the model and computes performance measures [24].

The interconnectedness of these components can be seen in Figure 9.7, which provides an overview of the simulation process with a flowchart.

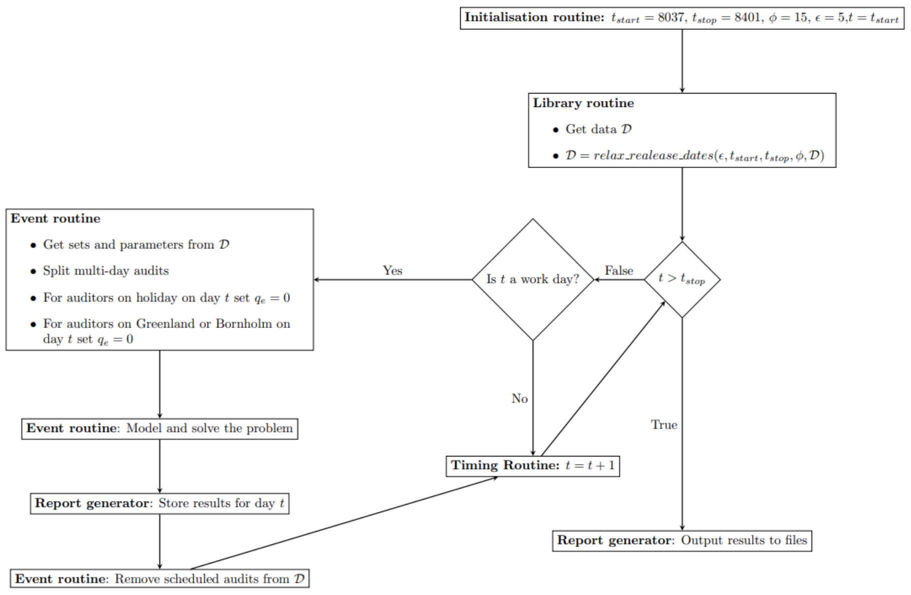

Initialisation routine: $t_{start} = 8037$, $t_{stop} = 8401$, $\phi = 15$, $\epsilon = 5$,$t = t_{start}$
Library routine
• Get data $\mathcal{D}$
• $\mathcal{D} = relax\_realease\_dates(\epsilon, t_{start}, t_{stop}, \phi, \mathcal{D})$
$t > t_{stop}$
False
Is t a work day?
Yes
Event routine
• Get sets and parameters from $\mathcal{D}$
• Split multi-day audits
• For auditors on holiday on day t set $q_e = 0$
• For auditors on Greenland or Bornholm on day t set $q_e = 0$
No
True
Event routine: Model and solve the problem
Timing Routine: $t = t + 1$
Report generator: Store results for day t
Report generator: Output results to files
Event routine: Remove scheduled audits from $\mathcal{D}$

Figure 9.7: *A flowchart of the simulation model used in this thesis.*

As this flowchart displays, the initialization routine initializes the start parameters. The library routine is then used to redistribute the release dates of the audits. If $t \leq t_{stop}$, and it is a work day, the event routine retrieves the relevant sets and parameters and determines the auditor capacity. Multi-day audits are also split up during this event routine.

The problem is modeled and solved at time step t using the commercial solver *Gurobi* [21]. The solver incorporated a time limit of two hours for each day to ensure that there would be at least preliminary results; however, the model was not interrupted by the time limit.

Subsequently, the report generator stores the results for day t, and scheduled audits are removed from the data. Lastly, the simulation clock is incremented by 1 and if $t > t_{stop}$ the simulation is complete and the report generator saves the results as a Python dictionary and a csv file. Otherwise, the process is repeated. For the specific implementation, see Appendix I

9.7 Input Data

The input data was queried directly from the database and converted to the data types listed below:

- V_t and O_t are represented as lists of integers that refers to their audit ID in Table 6.13.

- L is inserted as a list of integers that represents the facility ID of a depot from Table 6.1.
- E is inserted as a list of integers that represents the employee ID from Table 6.5.
- The d_i of each audit is inserted as a Python dictionary where the key references the audit ID from Table 6.13, and the return value is an integer representing the due date, which refers to an ID from Table 6.9.
- The u_i of each audit is represented as a Python dictionary, where the key refers to the audit ID and the return value is the difference between d_i and t.
- The p_i of each audit is stored in a Python dictionary where the audit ID is used as the key and the return value is the audit duration.
- The k_{lt} of each depot is represented as a Python dictionary where the depot's ID is used as the key and the return value is the number of available vehicles.
- The q_{et} of each auditor is represented as a Python dictionary where the auditor's ID is used as the key and the return value is the number of available work hours on day t.
- b_{le}, g_{ei}, and c_{ij} are represented as nested Python dictionaries.

9.8 Output Data

The output data is a csv file, which is shown in Table 9.1:

ID	facility_id	release_date_id	audit_date_id	due_date_id	duration	audit_type_id	required_skill_level	priority_before_audit	employee_id	zip_code	lat	long	on_site_audit
62306	1344	8040	8040.0	8127	0.2	5	1	1	15.0	0000	00.0	00.0	0
62308	1254	8040	8040.0	8127	0.2	5	1	1	7.0	0000	00.0	00.0	0
62309	639	8040	8040.0	8127	0.2	5	1	1	8.0	0000	00.0	00.0	0
62311	1013	8040	8040.0	8127	0.2	5	1	1	15.0	0000	00.0	00.0	0
62312	335	8040	8040.0	8127	0.2	5	1	1	7.0	0000	00.0	00.0	0

Table 9.1: *The output from the simulation model.*

The results are also stored in a dictionary, named *results_dict*, which contains the routes and audits for each auditor on each day. The scheduled audits on day t are accessed with the following keys:

```
results_dict[t][auditor_id]['audits']
```

Figure 9.8: *How to access the scheduled audits for a given auditor on day t.*

The routes on day t are accessed with the following keys:

```
    results_dict[t][auditor_id]['route']
```

Figure 9.9: *How to access the on-site routes for a given auditor on day t.*

An short excerpt from the output dictionary on day 8039 can be seen on Figure 9.10.

```
    8039:
       {
           1: {
               'audits': [],
               'route': []},
           2: {
               'audits': [],
               'route': []},
           3: {
               'audits': [],
               'route': []},
           4: {'audits': [],
               'route': []},
           5: {
               'audits': [47758, 47173, 46983],
               'route': [(47758, 46983),
                         (47173, 5),
                         (46983, 47173),
                         (5, 47758)]},
           6: {
               'audits': [48011, 48013, 48014, 48016],
               'route': [(48011, 48013),
                         (48013, 48016),
                         (48014, 48011),
                         (48016, 5),
                         (5, 48014)]},
           ...,
           15: {'audits': [], 'route': []}}
```

Figure 9.10: *Excerpt from the output dictionary.*

In order to ensure, that it was easy to perform a manual inspection of the results, the simulation model also printed the route; assigned audits; travel time; and audit time for each auditor, as can be seen on Figure 9.11.

```
##### AUDITOR: 15 ######
Total Audit Time: 0.75
Total Travel Time: 0
Total Time: 0.75

AUDITS
- Audit: 62306
- Audit: 62311
- Audit: 62313

ROUTE


##### AUDITOR: 4 ######
Total Audit Time: 1.9999999999999998
Total Travel Time: 3.9803484796678354
Total Time: 5.980348478667835

AUDITS
- Audit: 48056
- Audit: 47736
- Audit: 51192

ROUTE

- FROM 48056 → 5
- FROM 47736 → 48056
- FROM 51192 → 47736
- FROM 5 → 51192
```

Figure 9.11: *Model output on 01-04-2022 showing the schedule of auditors 15 and 4.*

All output should be intuitive to understand, and can easily be handled in post-processing for further analysis and experimentation. Moreover, the printed output makes it easy to manually inspect the results.

9.9 Conclusion

The input data in the model was extracted from the relational database and converted to lists and dictionaries. All audits, which had a duration greater than 8, were divided into 5-hour chunks, and the first auditor, who was assigned to the audit, was designated as the only auditor capable of performing the audit. All auditors received 42 uninterrupted vacation days. Although this is a rather crude way of simulating vacations, it was considered an acceptable approach for preliminary testing and results.

Since the DGA prefers to schedule audits in Bornholm and Greenland manually, these audits were removed from the data set. To mimic how the DGA visits these facilities, two auditors were given a week with 0 capacity, and it is assumed that they will perform as many audits as possible in Greenland. Moreover, two auditors were given one day with 0 capacity, where it is assumed that they will perform all their audits on Bornholm.

Since a large number of audits are released at the beginning of the year, a hyper-parameter, named ϕ, was introduced. This parameter refers to the number of audits which can be released on a given day when initializing the model. If more than ϕ audits are released on a given day, the remaining audits will be released on the next workday. To ensure that the audits were not released right before their due date, another parameter, named ϵ, was also introduced. This parameter determines how close to the due date an audit can be released. These parameters are important because they form the basis for the library routine, which determines the data distribution, in the simulation model.

For this thesis, the simulation model was tested for the year 2022 with $\phi = 15$ and $\epsilon = 5$. However, it should be noted that a higher value of ϕ will probably result in more effective schedules and routes, since more audits will be released earlier in the year.

The simulation model belongs to a class of models named discrete-event simulations, where an event occurs at each time t. This event changes the state variables immediately and, to keep track of events, a simulation clock is employed. With this simulation model, the problem is repeatedly modeled and solved on each day t in the time-horizon using the commercial solver Gurobi, and the results are stored. When t is greater than the last day of the time-horizon, all results are saved as a Python dictionary and a csv file.

Chapter 10

Preliminary Results

This chapter analyzes and discusses the preliminary results of the simulation model. These results are intended to test whether the simulation model works as intended. Section 10.1 performs a manual inspection of the output data. Section 10.2 analyzes the number of audits scheduled by the model compared to how many audits were scheduled by the DGA. Section 10.4 presents the total run-time of the simulation and Section 10.3 presents how the simulation model routed the auditors in 2022 compared to the DGA's districts. These sections also discuss what effect ϕ might have on the computational efficiency of the simulation model and the effectiveness of the routes.

10.1 Manual Inspection

Upon manual inspection, the model appears to design valid routes and is capable of differentiating between on-site and desk audits. As Figure 9.11 from Section 9.8 shows, the model successfully eliminates subtours, calculates audit durations, travel duration, and the combined duration. Additionally, only on-site audits are being routed. However, as expected, the model will sometimes schedule both on-site and desk audits on the same date, as can be seen in Figure 10.1.

```
##### AUDITOR: 2 ######
Total Audit Time: 4.574666666666666
Total Travel Time: 0.7415598361360106
Total Time: 5.316226502802677

AUDITS
- Audit: 34994
- Audit: 62050
- Audit: 62052
- Audit: 34965

ROUTE

- FROM 62050 → 3
- FROM 62052 → 62050
- FROM 3 → 62052
```

Figure 10.1: *Model output on 01-12-2022 showing the schedule of auditor 2.*

As Figure 10.1 shows, only audit 62050 and 62052 are actually routed, since these are on-site audits. Again, as previously mentioned, the assumption is, that the auditor will first perform the on-site audits and then perform desk audits when returning to the depot.

10.2 Audits Scheduled

Excluding Greenland and Bornholm, the number of scheduled audits was the following:

- **Audits released:** 2739
- **Audits scheduled by model:** 2735
- **Audits scheduled by auditors:** 2507

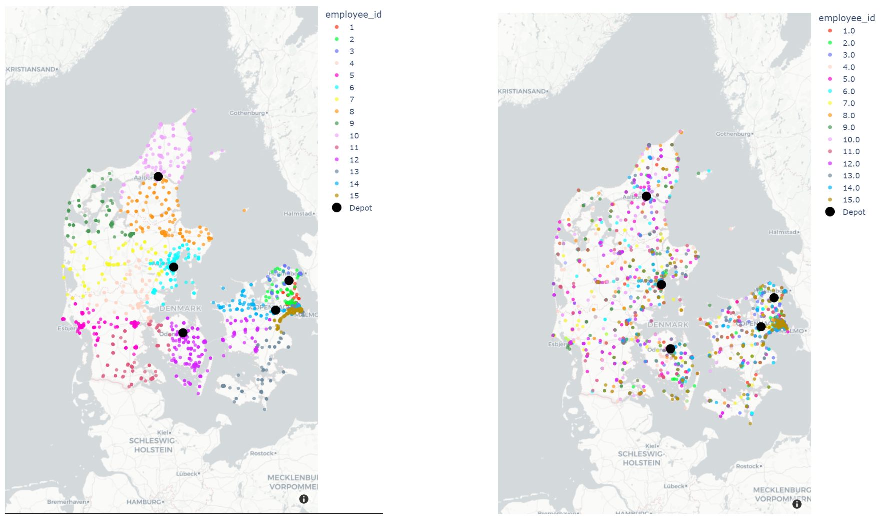
As the results show, there were 228 more audits scheduled by the model, compared to how the DGA scheduled their audits. However, it must be noted that this discrepancy might be the result of poor data quality since it was not always possible to obtain an audit date from the desk audits. To account for this, we can try assuming that all released desk audits in 2022 were performed. This gives the following results:

- **Audits Released:** 2739
- **Audits scheduled by model:** 2735
- **Audits scheduled by auditors:** 2629

Even when correcting for poor data quality, the model still outperforms the DGA's approach to scheduling when looking at the total throughput, however, this is probably not the best evaluation metric, since the routes might be of poor quality.

10.3 On-site Routing

The DGA's algorithm for assigning on-site audits to auditors, shown in Figure 5.10, can be compared to how the model assigns on-site audits, shown in Figure 10.2.



(a) DGA's On-Site Scheduling (b) Model's On-Site Scheduling

Figure 10.2: *How the DGA assigned on-site audits in 2022 and how the model assigned on-site audits in 2022.*

As Figure 10.2 displays, the DGA's current districts gives each auditor a much denser area to audit compared to the simulation model when ϕ is initialized to 15. Although these routes might be feasible, they are probably not effective. Therefore, it remains to be seen whether a higher value of ϕ will create more compact districts; If the model needs to be revised; or if the problem needs to be solved at a local or semi-global level as discussed in Section 7.4.

10.4 Efficiency

The total run-time was 33,346.34 seconds or 9.26 hours. However, it is expected that the run-time is heavily dependent on the value of ϕ upon initialization. The assumption being that, if more audits are released in the beginning of the year, the model will be forced to postpone more audits and find more effective schedules and routes.

10.5 Conclusion

A manual inspection of the results showed that the simulation model was able to design valid routes and schedules that respected auditor capacity and eliminated subtours. It was also noted that the simulation model would schedule both on-site audits and desk audits on the same day, which was expected.

Out of the 2739 audits released, the simulation model scheduled 2735 audits, while the DGA scheduled approximately 2629 audits (accounting for potential gaps in the data). Although it seems like the simulation model outperformed the DGA, in terms of total throughput, the DGA's districts provide more dense auditing areas. It remains to be seen whether increasing the value of ϕ will provide more compact auditing areas or if a different ILP formulation is needed.

The run-time was around 9 hours; however, a higher value of ϕ is expected to result in a longer run-time, since the simulation model is forced to find more effective routes and schedules.

Chapter 11

Evaluation Metrics

This chapter discusses different evaluation metrics for the ASRP. Section 11.1 discusses potential evaluation metrics for the schedule. Section 11.1.1 proposes measures based on either completion time. Section 11.1.3 proposes a measurement based on schedule preferences. Section 11.1.4 discusses an evaluating metric based on fairness, while Section 11.1.2 proposes a measure based on throughput. Section 10.3 discusses potential metrics to evaluate routes and proposes a metric that measures the relationship between hours spend auditing and the travel cost.

11.1 Evaluating the Schedule

There exist a myriad of different performance measures for scheduling problems, but they can generally be categorized into *completion time measures* and *due date measures* [28]. Due date measures are concerned with how late or early tasks are completed with respect to their release and due dates [28]. Since the ASRP exclusively deals with hard due and release dates, this category of performance measures can be excluded as potential evaluation metrics.

11.1.1 Completion Time Measures

Completion time measures are concerned with the time it takes to complete the tasks [28]. A common completion time metric is the *mean flow time* which represents the average time it takes for a released task to be completed. To define the flow time, let $\mathcal{F}$ denote the set of scheduled audits such that $\mathcal{F} = \{i \mid i \in V_t \ \forall t \in T \wedge \mathcal{C}_i \geq 0\}$. For this simulation model, the mean flow time can then be calculated with the following formula [28]:

$$\textit{Mean flow time} = \sum_{i \in \mathcal{F}} \frac{\mathcal{C}_i - r_i}{|\mathcal{F}|} \tag{11.1}$$

This measurement captures how long it on average takes for an audit to be scheduled after its release date. The aim of this measure is to, on average, have audits scheduled as soon as possible after their release date.

Another common completion time measure is the total *make-span*, denoted as $\mathcal{C}_{max}$, of the schedule. In this simulation model, the makes-span refers to the last audit date of the scheduled audits and can be defined as [28]:

$$C_{max} = max(\{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_{|\mathcal{F}|}\}) \tag{11.2}$$

The aim of this performance measure is to have the last audit date scheduled as early as possible [28]. The TISP, which was discussed in Section 4.4, uses a weighted make-span performance metric. To define the weighted make-span from the TISP, let $\mathcal{C}_{max^0}$ refer to the last audit date for an audit with priority 0. Let $\mathcal{C}_{max^1}$ refer to the last audit date for an audit with priority 1. Let $\mathcal{C}_{max^2}$ refer to the last audit date for an audit with priority 2. Finally, let $\mathcal{C}_{max^3}$ refer to the last audit date for an audit with priority 3. The weighted make-span for this simulation model can then be calculated with the following formula [14]:

$$\textit{Weighted make-span } = 28 \cdot \mathcal{C}_{max^3} + 14 \cdot \mathcal{C}_{max^2} + 4 \cdot \mathcal{C}_{max^1} + \mathcal{C}_{max^0} \tag{11.3}$$

Even though these completion time measures could provide some noteworthy insights into how long it takes to process audits, they do not account for the fact that a critical decision is whether to schedule an audit or to postpone it. Therefore, the simulation model might not schedule all audits within the time-horizon. The quality of the DGA's service is, therefore, not only dependent on how fast audits are scheduled, but also on how many audits the simulation model can schedule within the time-horizon. This suggests that the result shown in Section 10.2, which shows the total number of audits performed, could be used as a simple measure to evaluate the quality of the schedule. Within scheduling theory, this is known as the *throughput* [40].

11.1.2 Throughput Measures

Since it is generally assumed that all tasks are performed, the throughput is usually not categorized as a class of performance measures in the scheduling literature. However, in this case, the throughput could provide a good measure of schedule quality, since it can capture how many audits are being scheduled within the time-horizon. In this simulation model, the throughput can be calculated with the following formula:

$$\textit{Throughput } = \sum_{t \in T} \sum_{i \in V_t} \sum_{e \in E} y_{ie} \tag{11.4}$$

Formula 11.4 simply just counts how many audits are performed and it is equivalent to $|\mathcal{F}|$. This may seem like an intuitive approach; however, as shown in Figure 7.9, the vast majority of audits take less than 2 hours to complete. Therefore, a more appropriate evaluation metric is the *total audit time*, which in this thesis is defined as:

$$\textit{Total audit time } = \sum_{t \in T} \sum_{i \in V_t} \sum_{e \in E} p_i y_{ie} \tag{11.5}$$

The reason for using this metric is that a high throughput may not necessarily indicate effective scheduling if it mainly consists of short audits. Therefore, the total audit time on Formula 11.5 seems to be a more useful evaluation metric.

11.1.3 Schedule Preferences

It is possible to evaluate how closely an auditor's schedule resembles the scheduling preferences outlined in Section 7.2. This is done by taking the absolute difference between the proportion of time spent on desk audits and the proportion of the time spent on on-site audits. This is referred to as the auditor's *preference score* and is denoted by $\mathcal{P}_{et}$ $\forall e \in E, \forall t \in T$. It is calculated the following way for each auditor on each day:

$$\mathcal{P}_{et} = \sqrt{\left(\frac{\sum\limits_{i \in O_t \cup L} \sum\limits_{j \in O_t \cup L} c_{ij} x_{ije} + \sum\limits_{i \in O_t} p_i y_{ie}}{\sum\limits_{i \in O_t \cup L} \sum\limits_{j \in O_t \cup L} c_{ij} x_{ije} + \sum\limits_{i \in V_t} p_i y_{ie}} - \frac{\sum\limits_{i \in V_t / O_t} p_i y_{ie}}{\sum\limits_{i \in O_t \cup L} \sum\limits_{j \in O_t \cup L} c_{ij} x_{ije} + \sum\limits_{i \in V_t} p_i y_{ie}}\right)^2} \tag{11.6}$$

To demonstrate, let us assume that auditor e spends 2.5 hours driving, 2.5 hours performing on-site audits, and 5 hours performing desk audits on day t. This results in the following $\mathcal{P}_{et}$ value:

$$\sqrt{\left(\frac{2.5 + 2.5}{2.5 + 7.5} - \frac{5}{2.5 + 7.5}\right)^2} = 0$$

Therefore, since the auditors spends half of his time performing desk audits and half of his time performing on-site audits, the schedule is the worst possible schedule with respect to the auditors' scheduling preferences. If we instead assume that the auditor spends 0 hours driving, 0 hours performing on-site audits, and 10 hours performing desk audits, we get the following preference score:

$$\sqrt{\left(\frac{0 + 0}{0 + 10} - \frac{10}{0 + 10}\right)^2} = 1$$

This schedule, therefore, represents the best possible schedule, since the auditor exclusively performs one type of audits. To represent the schedule more broadly, the daily mean preference score, denoted by $\bar{\mathcal{P}}_t$ can be calculated with the following formula:

$$\bar{\mathcal{P}}_t = \frac{\sum\limits_{e \in E} \mathcal{P}_{et}}{|E|} \quad \forall t \in T \tag{11.7}$$

This can be extended to evaluate the schedule across the whole time-horizon by taking the mean of $\bar{\mathcal{P}}_t$ to get the *overall preference score* as shown in Formula 11.8:

$$\textit{Overall preference score} = \frac{\sum\limits_{t \in T} \bar{\mathcal{P}}_t}{|T|} \tag{11.8}$$

Since the overall preference score is an average, it is possible to compare models that was tested over different time horizons. However, one of its main drawbacks is that the resulting score is a statistical measure, which represents an abstract value. Therefore, it is mainly useful as a tool for comparing different models, rather than providing insigts into specific questions.

11.1.4 Fairness

Within scheduling theory, fairness is difficult to define and usually depends on the specific problem being solved [32]. However, a common metric of fairness is the dispersion of the workload between machines [32]. Dispersion refers to the spread of the data and, in this thesis, dispersion is given in terms of the *absolute mean deviation*, which measures the mean distance between each data point and the expected value (mean) [13]. This measurement was selected because it is more robust than the standard deviation [13] and more interpretable than the total variance.

In this thesis, the absolute mean deviation is calculated from the auditors' proportional time spent auditing. To define it, let f_e represent the proportion of an auditor's availability that is spent on auditing, which is calculated by the following formula:

$$f_e = \sum_{t \in T} \frac{\sum\limits_{i \in O_t \cup L} \sum\limits_{j \in O_t \cup L} c_{ij} x_{ije} + \sum\limits_{i \in V_t} p_i y_{ie}}{q_{et}} \tag{11.9}$$

The mean of all f_e, which is denoted as $\bar{f}$, can then be calculated with the formula below:

$$\bar{f} = \frac{\sum\limits_{e \in E} f_e}{|E|} \tag{11.10}$$

The dispersion is then given as:

$$Dispersion = \frac{\sum\limits_{e \in E} \sqrt{(f_e - \bar{f})^2}}{|E|} \tag{11.11}$$

What Formula 11.11 calculates is how dispersed each f_e $\forall e \in E$ is from the mean [13]. That is, how far is the average distance from f_e to $\bar{f}$ for all auditors. Therefore, unfair schedules, where some auditors spend substantial more proportional time auditing, will have a higher dispersion, as shown on Figure 11.1.

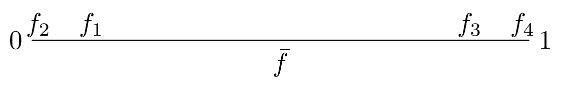


Figure 11.1: *Dispersion of an unfair schedule*

As can be seen, Figure 11.1 presents a highly unfair schedule, where auditor 3 and 4 spend almost all their available time auditing, while auditor 1 and 2 spend almost none of their available time auditing. Therefore, this schedule would result in a high dispersion, which means that the schedule is less fair. On the other hand, a fair schedule will have a low dispersion as seen on Figure 11.2.

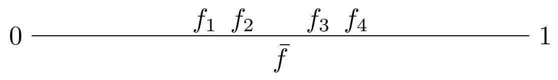


Figure 11.2: *Dispersion of a fair schedule*

Figure 11.2 presents a fair schedule where auditors approximately spend the same proportion of their available time auditing. This will result a lower dispersion, which means that the schedule is more fair. However, like the preference score from the previous section, the dispersion is an abstract statistical measure and is, therefore, mainly useful for comparison purposes.

11.2 Evaluating the Routing

Since most VRP formulations require that all vertices are visited, an intuitive evaluation metric is the *total travel cost*. This can be calculated with the following formula:

$$\textit{Total travel cost in hours} = \sum_{t\in T}\sum_{i\in O_t\cup L}\sum_{j\in O_t\cup L}\sum_{e\in E} c_{ij}x_{ije} \tag{11.12}$$

Note that even though the ASRP represents the travel cost in hours, it is also possible to represent the cost as kilometers driven, this is useful for calculating the fuel consumption. If we let c'_{ij} denote the travel cost in kilometers between facility $i \in V_t$ and $j \in V_t$, and assume that a vehicle spends 15 liters of gasoline per kilometer, then the total fuel consumption is calculated as follows:

$$\textit{Total travel cost in fuel} = \frac{\sum_{t\in T}\sum_{i\in O_t\cup L}\sum_{j\in O_t\cup L}\sum_{e\in E} c'_{ij}x_{ije}}{15} \tag{11.13}$$

This can be extended to calculate the amount of CO2 emitted. If we assume that each liter of gasoline emits 2.31 kg. of CO2 [31], then the total amount of CO2 emitted is calculated with the following formula:

$$\textit{Total travel cost in CO2} = \frac{\sum_{t\in T}\sum_{i\in O_t\cup L}\sum_{j\in O_t\cup L}\sum_{e\in E} c'_{ij}x_{ije}}{15} \cdot 2.31 \tag{11.14}$$

However these evaluation metric do not account for the fact that some on-site audits might not be scheduled. Therefore, the metric to evaluate routing must capture the relationship between auditing time and the travel cost. An approach would be to measure the cost of each hour spent auditing, which will be referred to as the *normalized* *travel cost.* This is defined as:

$$\textit{Normalized travel cost} = \frac{\textit{Total travel cost}}{\sum_{t\in T}\sum_{i\in O_t}\sum_{e\in E} p_i y_{ie}} \tag{11.15}$$

What Formula 11.15 calculates is the total cost divided by the total time spent auditing. A lower normalized travel cost means that the cost of each auditing hour is lower. Therefore, a low value suggests that auditors are sent on more effective routes. This should ensure that it is possible to compare the effectiveness of the routes even if the results have different throughput measures.

11.3 Conclusion

Performance measures for scheduling problems can generally be categorized into completion time measures and due date measures. Due date measures are concerned with how early or late a task is performed. Since the ASRP only deals with hard due dates and release dates, this category of measurements was discarded as potential evaluation metrics.

Completion time measures are concerned with how long it takes to complete the tasks. One such measurement is the mean flow time, which measures how long it takes on average for an audit to be scheduled after its release date. Another completion time measure is the total make-span, which measures the last date an audit is scheduled. In the TISP, a weighted make-span measurement was used, which assigns a weight to the make-span of audits with a specific priority. All of these metrics could provide valuable insight into the quality of the schedule; however, they do not account for the fact that some audits might be unscheduled.

Therefore, a throughput-based metric was also discussed. Since most scheduling problems assume that all tasks are performed, throughput is usually not used as an evaluation metric within the scheduling literature. However, for the ASRP, where some audits may be unscheduled, this metric can measure how many audits are actually scheduled in the time-horizon. However, a high throughput may not indicate optimal scheduling if it primarily consists of short audits. Therefore, a measure, which calculates the total time spent auditing within the time-horizon, was proposed instead.

To evaluate how closely a schedule aligns with the auditors' scheduling preferences, a preference score for each auditor on each day was presented. This preference score takes the absolute difference between the proportional time spent performing on-site audits and the proportional time spent on desk audits. This score is then averaged across all auditors and all days to get the overall preference score for the whole schedule. For fairness, a measure of dispersion between auditors' proportional auditing time and the mean was presented. The main drawbacks of both the preference score and dispersion is that they represent statistical measurements, which are abstract values. Therefore, they are mainly useful for comparison purposes. However, since they both represent averages, they can be used to compare simulation models with different time-horizons and different number of auditors.

Since most VRP models require that all vertices are visited, an intuitive metric is the total travel cost. However, since the ASRP allows audits to be postponed, models with a lower throughput could obtain a lower travel cost. Therefore, a metric was proposed that measures the cost of each hour spent auditing. This was referred to as the normalized travel cost.

Chapter 12

Concluding Remarks

This chapter provides a discussion of future work on the project and a conclusion. Section 12.1 discusses future work on the project, while Section 12.2 provides a summary and conclusion of the report.

12.1 Future Work

This project is only the first step towards the DGA's goal of incorporating data in their auditing approach. First, the evaluation metrics must be incorporated into the report generator of the simulation model, so that the results of different mathematical models and ϕ values can be compared. Furthermore, data from the DGA should be more closely integrated with the relational database. Auditor and vehicle availability could, for example, be retrieved directly from auditors' calendars.

Different ILP formulations should also be tested. The MTZ subtour elimination constraint, which was presented in Section 8.10, should be compared to the current DFJ formulation. Furthermore, the alternative due date constraint, which was presented in Section 8.8, should be incorporated into the current ILP formulation. Moreover, ILP models that use the current districts of the DGA to address the problem as a local or semi-global problem, which was discussed in Section 7.4, should be formulated and compared with the current simulation model. The ASRP, which is presented in this thesis, tackles the problem globally by solving it as a daily problem with a rolling time-horizon. However, it would also be relevant to solve the problem as a global problem over the full time-horizon by using heuristics.

As demonstrated in Section 7.6, there is a substantial increase in workload from 2023 and beyond. Therefore, it is necessary to test the simulation model on different years to see how the increasing workload affects the run-time of the simulation model and quality of the results.

Lastly, as discussed in Section 7.1, the overall problem can be decomposed into three subproblems: audit creation, routing, and audit scheduling. The ASRP, presented in this thesis, deals exclusively with routing and scheduling. However, for future work on the project, it will be interesting to explore the problem of audit creation. To do this, it would be necessary to develop an optimizer or machine learning algorithm that determines audit windows and audit priorities based on the audit history of a facility.

12.2 Conclusion

To aid in the DGA's goal of incorporating data when auditing facilities, this thesis has presented the Auditor Scheduling and Routing Problem (ASRP). This problem was formulated as an Integer Linear Programming (ILP) problem that is solved using a solver. A solver incorporates a mixture of the simplex method; duality; the Branch and Bond (BnB) method; and cutting-plane algorithms to find the optimal solution to a mathematical problem.

The ILP formulation presented in this thesis belongs to a class of rich VRPs named resource-constrained routing and scheduling problems. These problems incorporate various resources, such as skills or spare-parts, to route a fleet of vehicles and meet the specific demands of customers. The ASRP is similar to the Technician Scheduling and Routing Problem (TSRP), which seeks to schedule and route technicians to requests. Technicians must have the required skills and spare-parts to meet these requests. In the ASRP, a distinction is made between on-site and desk audits. On-site audits must be performed at the facility, while desk audits must be performed at one of the depots. All auditors are attached exclusively to a designated depot. They can only leave this depot once a day and must return to the same depot at the end of the day. Unlike the TSRP, where it is assumed that there is enough vehicles for all auditors, each depot in the ASRP only has a limited number of available vehicles, and an auditor can only leave for on-site audits if there is an available vehicle.

All audits have a risk assessments, which can be white, green, yellow, or red. These assessments are converted to numerical priorities since it offers more flexibility when assessing how important an audit is, and it opens up the possibility of using the priorities as weights in a model. Each audit has a certain duration it takes to perform the audit. To develop a formula for calculating audit durations, auditors were observed while demonstrating how they performed different audit types.

All audits must be performed after a hard release date and before a hard due date. The time between the due date and the release date represents the audit window within which an audit must be performed. After an audit has been performed, the auditor, who performed the audit, manually inserts a new audit with an audit window and a priority. All dates from 01-01-2000 to 12-31-2030 have been discretized into days and each day has been discretized into 24 hours. Each hour represents a time slot, and all auditors and vehicles can be available or unavailable at a given time slot. Currently, the availability has been determined manually according to heuristics; however, for future work on the project, it will be necessary to collect these data directly from the DGA. This can be done by either gathering the data from the auditors' outlook calendars or by using historical data to estimate auditor availability. The availability is used to calculate the daily depot and auditor capacity. Auditor capacity denotes how many hours an auditor is available to work on day t. It is calculated by counting the number of available time slots on day t. The daily depot capacity refers to how many vehicles are available from 6 to 18 in a depot on day t.

Each audit is of a certain audit type, and each audit requires a certain level of skill within that type of audit. If an auditor is not skilled enough, he cannot perform the audit. Instead of dealing with skill levels and audit types directly in the model, a binary matrix, named the accomplice matrix, is used to represent this. The cost of traveling between facilities is represented by a travel-time matrix. This matrix provides a rough estimate of the travel times by calculating the earth's haversine distance between all facilities and dividing it by the speed per hour. In this thesis, the speed per hour was set to 80 km per hour. The resulting matrix is a symmetrical matrix in which the diagonal only contains zeros. When auditors leave for an on-site audit, they can only leave their designated depot once, and they must return to the same audit on the same day. To represent this, a binary auditor-depot matrix is used.

The DGA currently schedules on-site audits according to districts. Each district consists of all facilities, of a given type, within a given zip code. Each district has an auditor assigned who is responsible for performing all on-site audits within this district. Yearly on-site projects represent a type of on-site audits, where 1200-1500 audits are released at the beginning of the year, and all auditors must choose between 25-50 audits, within their districts, to perform before the end of the year. Before 2023, desk audits were also assigned to auditors based on districts and tended to be performed as part of the on-site audit or on an ad hoc basis. However, from 2023, the DGA started assigning desk audits according to small teams, where each team performs a specific type of desk audits. With this new approach, a batch of desk audits, with a year-long audit window, is released at the beginning of each year. The auditors assigned to the given audit type will then sort the batch in descending order according to the required skill levels. The auditors will then process the audits in parallel as a queue starting from an audit within their skill level.

Semi-structured interviews with auditors were used to determine their scheduling preferences. It was noted that auditors generally prefer to have long days where they only perform on-site audits and short days where they only perform desk audits. To achieve this, the auditors may relax the release dates or the due dates. However, an auditor can only do this using his domain expertise of the facility. This suggests that the ASRP can be modeled with soft audit windows. However, to keep the scope of this thesis limited, it was decided to use hard due and release dates and to not incorporate the scheduling preferences in the model.

In the preliminary analysis, it was noted that the master problem can be divided into three subproblems: audit creation, which is concerned with determining audit priorities and audit windows; routing, which seeks to minimize the travel time; and audit scheduling, which seeks to maximize the number of scheduled audits. The ASRP only deals with the routing and scheduling problem, since the audit creation is done manually. However, for future work on the problem, it would be worth tackling the audit creation problem. This could be done by an optimizer or a machine learning algorithm, which can consider the audit history of a facility when determining audit priorities and audit windows.

Using exploratory visualizations, it was recognized that the routing subproblem can be solved at three levels: The local level, where auditors are assigned exclusively to audits within their district; the semi-global level, where auditors are assigned exclusively to audits within the combined districts of all auditors belonging to the same depot; and the global level, where the districts are not respected. The local level is a single depot problem with a single vehicle. It is expected to be the least effective in terms of quality, but the most efficient in terms of computation. The global level solves the problem as a multi-depot problem with multiple vehicles. It is expected to be the most effective in terms of quality, but the least efficient in terms of computation. Lastly, the semi-global level falls somewhere in-between the local and global level in terms of quality and computational efficiency. For this thesis, the problem was solved at the global level; however, it would be noteworthy to compare the results of the global level with a simulation model that solves the problem at a local or semi-global level.

With a Gantt chart, it was discovered that the audit windows follow a regular pattern. This pattern shows that a large number of audits, with a yearly audit window, are released at the beginning of each year, whereas audits, with shorter audit windows, are released evenly during the rest of the year. This is due to the fact that all desk audits and yearly on-site projects are released each year the first of January. During the rest of the year, on-site audits are released. Furthermore, it was also noted that there are substantially more audits released in 2023 and beyond. A bar chart was used to investigate the workload from 2020 to 2023, and it was noted that there is a substantial increase in the workload from 2023. This is due to the fact that substantially more desk audits are released from 2023, and that this year and onward also contain potential audits, while 2022 only contains performed audits. This bar chart also demonstrates that the auditor capacity is substantially greater than the workload. This suggests that the data does not fully capture all the auditors' tasks; that the current way of scheduling is inefficient; or that the auditors spend a long time driving to on-site audits.

From the exploratory visualizations, it was observed that the scale of the global problem is too large to be solved with any known solvers. Therefore, the problem would either have to be solved with heuristics or with a rolling time-horizon. In this thesis, the second option was selected, and the ASRP is, therefore, organized as a daily problem. Since the problem is solved on a daily basis, it is assumed that there might be a greater workload than capacity. Therefore, unlike most VRPs, the critical decision is not only the sequencing and assignment of audits, but also which audits to perform and which audits to postpone. Therefore, the objective of the ASRP is to schedule as many audits as possible by designing tours for the auditors. In these tours, the capacity; the required skills levels; the vehicle availability; and the hard due dates must be respected.

The ASRP is not the only governmental organization which incorporates a model with a rolling time-horizon when scheduling their regulatory duties. In the Toll Enforcement Problem (TEP) inspectors from the German Federal Office of Goods Transport (BAG) must ensure that tolls on German highways are paid. However, unlike the ASRP, the TEP does not consider skill levels, and the problem is organized as a monthly rather than a daily problem. The ASRP also shares similarities with the Technicians and Interventions Scheduling Problem (TISP), which was proposed in the

2007 ROADEF challenge. In the TISP, a group of technicians, with a combination of certain skill levels, must be assigned to perform interventions before their due date. This is similar to the ASRP, where auditors have skill levels within a set of audit types. Moreover, both the ASRP and the TISP have priorities attached to the vertices, but the ASRP does not use these priorities directly in the model. In terms of routing, the ASRP shares a lot of similarities with the Multi-Period Vehicle Routing Problem with Due dates (MPVRPD). In the MPVRPD a set of customers must be served after their release date and before their due date within a given time-horizon.

It was observed that the ILP formulation, presented in this thesis, ensured that the solution would be infeasible if $u_i < 0$. However, an alternative due date constraint was presented, forcing the model to schedule an audit if $u_i = t$.

The objective function in this formulation is rather unusual for most VRPs and TSRPs since the objective is neither to minimize the travel cost nor to optimize a quality of service. Instead, the objective is to perform as many audits as possible. The assumption being that in order to perform as many audits as possible, the model must design as effective routes and schedules as possible. However, since the travel cost is not directly minimized, it is assumed that the routes and schedules will only be effective if there is a higher workload than auditor capacity.

For the subtour elimination constraint (SEC), a cutting-plane version of the Dantzig-Fulkerson-Johnson (DFJ) formulation was used. In this approach, a relaxed version of the problem, without any subtour constraints, is solved. The solution is then checked for subtours and all detected subtours are added as cuts to the model, whereupon the problem is resolved. This is done iteratively until the solution contains no subtours. To reduce the number of cuts added in the subtour elimination procedure, the model uses a disconnected graph, where there are no edges going to or from the desk audits.

Another SEC, named the Aggregated Dantzig-Fulkerson-Johnson (ADFJ) formulation, was also tested. In this SEC the cut is applied across the sum of all auditors rather than for each auditor. However, this SEC was substantially slower than the standard DFJ formulation and was discarded. An SEC, named the Miller-Tucker-Zemlin (MTZ) formulation, was also presented. This SEC enumerates all vertices according to their duration, so that $s_j \geq s_i + p_j$ if $x_{ije} = 1$. This method was not tested, but for future work on the project, it will be necessary to test what effect this SEC will have on the run-time.

To test the ILP model, a discrete-event simulation model, which simulates each day of a year, was used. It was decided to test the model in 2022, since this year provides a small-scale version of the problem, which can be used to get preliminary results. In the simulation model, all audits in Greenland and Bornholm are removed from the set of audits. Then, two auditors are set to be unavailable for a week in February with the assumption that they will perform as many audits as possible in Greenland during this period. For Bornholm, two auditors were set as unavailable for two days in late January, with the assumption that they will perform all audits on Bornholm. For vacation, all auditors received 42 uninterrupted vacation days. This is a crude way of mimicking vacations, but, for preliminary results, it was considered an acceptable proxy. However, for future work on the project, it will be necessary to either gather the vacation dates directly from the auditors' outlook calendars or by using historical data to estimate vacation dates.

In the preliminary analysis, a histogram showed that the vast majority of audits take two hours or less to perform. However, some audits can take up to 35 hours to complete. Therefore, all audits, with a duration longer than 8 hours, were split into 5-hour chunks, and the first auditor assigned to a multi-day audit was set as the only one capable of performing the audit in the accomplice matrix. Since a large number of audits are released at the beginning of the year, the hyper-parameters ϕ and ϵ were introduced in the simulation model. Upon initialization, the value of ϕ determines how many audits can be released each day, while ϵ determines the minimum value u_i can take before audit i is released. With these parameters, it is possible to test different distributions of release dates and it is assumed that a higher value of ϕ will result in more effective routes and schedules. However, for preliminary testing, $\phi = 15$ and $\epsilon = 5$ were set as the initial parameters since they provide a uniform distribution of release dates, which should be easy to model and solve. However, for future work on the project, it is necessary to run the simulation model with different values of ϕ and ϵ to test how these affect the results.

The data was stored in a relational database that was designed with input from auditors and project managers. The simulation model retrieves the data directly from this database and converts it into lists and dictionaries in Python. The report generator outputs a csv file, a Python dictionary, and prints the results to the console. For further work on the project, the report generator must output some performance measures in the form of evaluation metrics. This will enable us to test and compare different simulation models. These evaluation metrics can take a variety of forms. The mean flow time could provide interesting information on how long it takes to complete the audits after their release dates. The make-span can measure when the last audit is performed. A weighted make-span metric, which was proposed in the TISP, would provide information on when audits, with different priorities, are scheduled. However, all these evaluation metrics do not account for the fact that the simulation model might not schedule all audits. Therefore, a performance measure based on throughput was proposed. An intuitive metric could be to count the number of audits performed. However, this performance measure does not account for the differing audit durations. Instead, an evaluation metric, which measures the total audit time, was proposed.

To measure how closely the schedule aligns with auditors' scheduling preferences, an evaluation metric, which takes the absolute difference between the proportional time spent on desk audits and on-site audits and averages it, was proposed. For fairness, a measure of dispersion, which calculates the absolute mean deviation between the auditors' proportional auditing time and the mean, was presented. Since these metrics represent abstract statistical measurements, they are mainly useful for comparing different simulation models.

For evaluating the routing, it might seem intuitive to sum up the total travel cost; however, the risk is that simulation models, with a lower throughput, might gain a better routing score. Therefore, a performance metric, which measures the travel cost of each auditing hour, was proposed . This measurement was named the normalized total travel cost and a lower value means more effective routes.

Upon manual inspection of the preliminary results, it was concluded that the simulation model successfully eliminated subtours, and it was also noted that the simulation model would sometimes schedule both on-site audits and desk audits on the same day. This was expected and the assumption is that when an auditor returns to his depot, he will perform the desk audits. In terms of total throughput, the simulation model was able to schedule more audits than the auditors at the DGA. However, upon inspection of the on-site routing, it was noted that auditors were routed throughout Denmark. This is in stark contrast to the predefined districts of the DGA, which offer more dense and compact geographic auditing areas. Therefore, it remains to be seen whether a higher value of ϕ will provide more effective routes; if the model needs to be revised; or if the problem needs to be solved at the local or semi-global level. In terms of efficiency, the simulation model had a run-time of around 9 hours, but it is expected that a higher value of ϕ will result in a higher run-time.

As should be clear from this thesis, the ASRP represents an intriguing and complex problem, which should aid the DGA in scheduling audits. However, this project is only the first step towards a fully data-driven auditing approach, and there is still much work to be done before the problem can be considered solved.

Bibliography

[1] AIMMS. Miller-tucker-zemlin formulation. https://how-to.aimms.com/Articles/332/332-Miller-Tucker-Zemlin-formulation.html, 2020. (accessed: 02-23-2024).

[2] Archetti, C., Jabali, O., and Speranza, M. G. Multi-period vehicle routing problem with due dates. *Computers & Operations Research 61* (2015), 122–134.

[3] Bakker, J. D. *Applied Multivariate Statistics in R.* University of Washington, 2014, ch. Properties of Distance Measures.

[4] Baldacci, R., Battarra, M., and Vigo, D. Routing a heterogeneous fleet of vehicles. In *The Vehicle Routing Problem: Latest Advances and New Challenges.* Springer, 2008.

[5] Bazrafshan, R., Zolfani, H. S., and Mirzapour, S. M. J. A. Comparison of the sub-tour elimination methods for the asymmetric traveling salesman problem applying the seca method. *Axioms 10* (2021).

[6] Bektas, T., and Gouveia, L. Requiem for the miller–tucker–zemlin subtour elimination constraints? *European Journal of Operational Research 236* (2014), 820–832.

[7] Chakraborty, A., Chandru, V., and Rao, M. R. A linear programming primer: from fourier to karmarkar. *Annals of Operations Research 287* (2020), 593–616.

[8] Codd, E. F. A relational model of data for large shared data banks. *Research Report / RJ / IBM / San Jose, California RJ909*, 6 (1970).

[9] Codd, E. F. Further normalization of the data base relational model. *Research Report / RJ / IBM / San Jose, California RJ909* (1971).

[10] Conforti, M., Cornuéjols, G., and Zambelli, G. *Integer Programming*, 1 ed. Springer, 2014, ch. Methods for Solving Integer Programs.

[11] Dantzig, G. B., Fulkerson, R., and Johnson, S. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America 2*, 4 (1954), 393–410.

[12] Dantzig, G. B., and Ramser, J. H. The truck dispatching problem. *Management Science 6*, 1 (1959), 80–91.

[13] Dodge, Y. *The Concise Encyclopedia of Statistics.* Springer New York, 2008, ch. Mean Absolute Deviation, pp. 336–337.

[14] Dutot, P., Laugier, A., and Bustos, A. Technicians and interventions scheduling for telecommunications. *ROADEF* (2006).

[15] Gamrath, G., Reuther, M., Schlechte, T., and Swarat, E. An lp-based heuristc for inspector scheduling. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling - PATAT 2021: Volume I* (2021).

[16] GeoPy. Geopy. https://github.com/geopy/geopy, n.d. (version 2.4.0).

[17] Glomb, L., Liers, F., and Rösel, F. A rolling-horizon approach for multi-period optimization. *European Journal of Operational Research 300*, 1 (2022), 189–206.

[18] Golden, B., Raghavan, S., and Wasil, E. Preface. In *The Vehicle Routing Problem: Latest Advances and New Challenges.* Springer, 2008.

[19] Golden, B., Raghavan, S., and Wasil, E. *The Vehicle Routing Problem: Latest Advances and New Challenges.* Springer, 2008.

[20] Goldfarb, D. *Mathematics and Its Applications: Advances in Optimization and Numerical Analysis.* Springer, 1994, ch. On the Complexity of the Simplex Method.

[21] Gurobi Optimization LLC. gurobipy 10.0.2. https://pypi.org/project/gurobipy/, n.d. (accessed: 08-07-2023).

[22] Knaflic, C. N. *Storytelling with Data: A Data Visualization Guide for Business Professionals*, 1 ed. John Wiley & Sons, Inc., 2015.

[23] Laporte, G., Ropke, S., and Vidal, T. *Vehicle Routing: Problems, Methods, and Applications*, 2 ed. Mathematical Optimization Society and the Society for Industrial and Applied Mathematics, 2014, ch. Heuristics for the Vehicle Routing Problem, pp. 87–88.

[24] Law, A. M. *Simulation modeling and analysis*, 5 ed. McGraw-Hill Education, 2015, ch. Basic Simulation Modelling.

[25] Law, A. M. *Simulation modeling and analysis*, 5 ed. McGraw-Hill Education, 2015, ch. Simulation Software.

[26] Miller, C. E., Tucker, A. W., and Zemlin, R. A. Integer proramming formulation of the travelling salesman problems. *Journal of Association for Computing Machinery 7*, 4 (1960).

[27] Mogensen, H. O., and Dalsgård, A. L. *Antropologiske Projekter*. Samfunds litteratur, 2018, ch. At være tilstede: Deltagelse og observation.

[28] Mokotoff, E. Parallel machine scheduling problems: A survey. *Asia-Pacific Journal of Operational Research 18* (2001).

[29] Mor, A., and Speranza, M. G. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering 79* (2015), 115–129.

[30] Mor, A., and Speranza, M. G. Vehicle routing problems over time: A survey. *4OR 18* (2020), 129–149.

[31] Natural Ressources Canada. Learn the facts: Emissions from your vehicle. https://natural-resources.canada.ca/sites/www.nrcan.gc.ca/files/oee/pdf/transportation/fuel-efficient-technologies/autosmart_factsheet_9_e.pdf, 2014. (accessed: 03-25-2024).

[32] Ngubiri, J., and van Vliet, M. A metric of fairness for parallel job schedulers. *Concurrency and Computation: Practice and Experience 21*, 12 (2009), 1525–1546.

[33] O'Regan, G. *Mathematical Foundations of Software Engineering: A Practical Guide to Essentials*, 1 ed. Springer, 2023, ch. Introduction to Operations Research.

[34] Pandas. About pandas. https://pandas.pydata.org/about/, n.d. (accessed: 08-07-2023).

[35] Paraskevopoulos, D. C., Laporte, G., and Repoussis, P. P. Resource constrained routing and scheduling: Review and research prospects. *European Journal of Operational Research 263* (2017), 737–754.

[36] Pedregosa, F., et al. Scikit-learn: Machine learning in python. *Journal of machine learning research 12* (2011), 2825–2830.

[37] Petropoulos, F., et al. Operational research: methods and applications. *Journal of the Operational Research Society 0*, 0 (2023), 1–195.

[38] Pillac, V., Guéret, C., and Medaglia, A. On the technician routing and scheduling problem. *Proceedings of the 9th Metaheuristics Conference (MIC 2011)* (2011), 675–678.

[39] Pillac, V., Guéret, C., and Medaglia, A. L. On the dynamic technician routing and scheduling problem. *Proceedings of the 5th International Workshop on Freight Transportation and Logistics (ODYSSEUS 2012)* (2012).

[40] Pinedo, M. L. *Scheduling: Theory, Algorithms, and Systems*, 5 ed. Springer, 2016, ch. Introduction.

[41] Pinedo, M. L. *Scheduling: Theory, Algorithms, and Systems*, 5 ed. Springer, 2016, ch. Deterministic Models: Preliminaries.

[42] Rubow, C., Bundgaard, H., and Mogensen, H. O. *Antropologiske Projekter*. Samfunds litteratur, 2018, ch. Introduktion.

[43] Rytter, M., and Olwig, K. F. *Antropologiske Projekter*. Samfunds litteratur, 2018, ch. At snakke om det: Måder at interviewe på.

[44] Scikit Learn. sklearn.metrics.pairwise.haversine_distances. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.haversine_distances/, n.d. (accessed: 02-01-2024).

[45] Shneiderman, B. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE Symposium on Visual Languages* (1996), pp. 336–343.

[46] Spillemyndigheden. Compliance assessment and reponses. https://www.spillemyndigheden.dk/en/compliance-assessment-and-reponses, n.d. (accessed: 08-05-2023).

[47] Spillemyndigheden. Mål- og resultatplan. https://www.spillemyndigheden.dk/uploads/2023-01/Spillemyndighedens%20m%C3%A5l-%20og%20resultatplan%202023.pdf, n.d. (accessed: 08-05-2023).

[48] Tufte, E. R. *The Visual Display of Quantitative Information*, 2 ed. Graphic Press, 2005.

[49] Unwin, A. Why is data visualization important? what is important in data visualization? *Harvard Data Science Review 2*, 1 (2020).

[50] Vanderbei, R. J. *Linear Programming: Foundations and Extensions*, 5 ed. Springer, 2020, ch. The Simplex Method.

[51] Vanderbei, R. J. *Linear Programming: Foundations and Extensions*, 5 ed. Springer, 2020, ch. Duality Theory.

[52] Vanderbei, R. J. *Linear Programming: Foundations and Extensions*, 5 ed. Springer, 2020, ch. Integer Programming.

[53] Widom, G. U. *Database Systems: The Complete Book*, 2 ed. Pearson, 2014.

[54] Wolsey, L. A. *Integer Programming*, 2 ed. John Wiley & Sons, Inc., 2021, ch. Branch and Bound.

Appendix A

Code for Retrieving Sets and Parameters

```
###########
# Imports #
###########
# Data retrieval and wrangling
import pandas as pd
import sqlite3
from contextlib import closing

# Dates
import modules.utils.date_utils as date_utils

########################
# Auxilliary functions #
########################

# ----- Returns data frames ------

def get_all_audits(con: sqlite3.Connection) -> pd.DataFrame:
    """
    Returns a dataframe containing all audits and geographic information
    """
    return pd.read_sql("""SELECT
                                all_tasks.ID,
                                all_tasks.facility_id,
                                all_tasks.release_date_id,
                                all_tasks.audit_date_id,
                                all_tasks.due_date_id,
                                all_tasks.duration,
                                all_tasks.audit_type_id,
                                all_tasks.required_skill_level,
                                all_tasks.priority_before_audit,
                                all_tasks.employee_id,
                                facilities.zip_code,
                                facilities.lat,
                                facilities.long,
                                audit_types.on_site_audit
                           FROM all_tasks
                           INNER JOIN audit_types ON all_tasks.audit_type_id = audit_types.ID
```

```python
                            INNER JOIN facilities ON all_tasks.facility_id = facilities.ID
                            """, con)


def get_daily_audits(date: str,
                     con: sqlite3.Connection,
                     task_tbl: pd.DataFrame) -> pd.DataFrame:
    """
    Takes the dataframe from get_all_audits and returns the audits from a specific date ID
    """
    date_id = date_utils.convert_date_to_id(date, con)
    return task_tbl[task_tbl["release_date_id"].astype(int) == date_id]



# ----- Returns dictionaries -----

def get_daily_vehicle_capacity(date_id: int,
                               con: sqlite3.Connection,
                               start_hour: int = 7,
                               end_hour: int = 17) -> pd.DataFrame:
    time_slots = date_utils.get_date_time_slots(date_id, con)
    """
    Returns a dictionary that stores how many time slots a vehicle is available between a start hour
                                                         and an end hour
    """
    time_slot_range = [time_slots[i] for i in range(start_hour, end_hour)]

    availability = pd.read_sql(f"SELECT * FROM vehicle_availability WHERE time_slot_id in {tuple(
                                                         time_slot_range)}", con)
    availability = availability.groupby("vehicle_id", as_index=False)["available"].sum()
    vehicles = availability["vehicle_id"].to_list()
    capacity = availability["available"].to_list()

    return_dict = {
    }
    for vehicle, capacity in zip(vehicles, capacity):
        return_dict[vehicle] = capacity
    return return_dict
```

```
def get_depots_and_vehicles(con: sqlite3.Connection) -> dict[int, list[int]]:
    """
    Returns a dictionary containing a list of all vehicles associated with a depot.
    """
    with closing(con.cursor()) as cur:
        vehicle_depots = cur.execute("""SELECT
                                            facilities.ID AS facility_id,
                                            vehicles.ID AS vehicle_id
                                        FROM facilities
                                        INNER JOIN vehicles ON facilities.ID = vehicles.depot_id""").
                                                                                            fetchall()
        return_dict = {}
        for depot, _ in vehicle_depots:
            return_dict[depot] = []

        for depot, vehicle in vehicle_depots:
            return_dict[depot].append(vehicle)
    return return_dict

################################
# Retrieve Sets and Parameters #
################################

# ----- Sets -----
def get_employees(con:sqlite3.Connection) -> list[int]:
    """
    Returns a list of auditors, which represent E
    """
    with closing(con.cursor()) as cur:
        employees = cur.execute("SELECT ID FROM employees").fetchall()
    return [em[0] for em in employees]


def get_vehicles(con:sqlite3.Connection) -> list[int]:
    """
    Returns a list of vehicles
    """
    with closing(con.cursor()) as cur:
        vehicles = cur.execute("SELECT ID FROM vehicles").fetchall()
    return [vehicle[0] for vehicle in vehicles]


def get_depots(con: sqlite3.Connection) -> list[int]:
    """
    Returns a list of depots, which represents L
    """
    with closing(con.cursor()) as cur:
        depots = cur.execute("""SELECT
                                    facilities.ID AS facility_id
                                FROM facilities
                                WHERE facilities.facility_type_id = 15""").fetchall()
    return [depot[0] for depot in depots]


def get_on_site_audits(audits: pd.DataFrame) -> list[int]:
    """
    Returns a list of on-site audits, which represent O
    """
    return [*audits[audits["on_site_audit"] == 1]["ID"]]


def get_audits_as_list(audits: pd.DataFrame) -> list[int]:
    """
    Returns a list of on-site audits, which represents V
    """
    return [*audits["ID"]]


# ----- Parameters -----

def get_processing_times(audits: pd.DataFrame) -> dict[int, int]:
    """
    Returns a dictionary of audit durations which represent p_i
    """
    durations = audits["duration"].to_list()
    audits = audits["ID"].to_list()
    p = {}
    for _, i in enumerate(audits):
        p[i] = durations[_]
    return p


def get_due_dates(audits: pd.DataFrame) -> dict[int, int]:
    """
    Returns a dictionary of due dates which represent d_i
    """
    tasks = audits["ID"].to_list()
    due_dates = audits["due_date_id"].astype(int).to_list()

    d = {}
    for task, due_date in zip(tasks, due_dates):
        d[task] = due_date
    return d


def get_daily_employee_capacity(date_id: str,
                                con: sqlite3.Connection) -> pd.DataFrame:
    """
    Returns a dictionary of daily auditor capacities, which represent q_{et}
    """
    time_slots = date_utils.get_date_time_slots(date_id, con)
    availability = pd.read_sql(f"SELECT * FROM employee_availability WHERE time_slot_id in {tuple(
                                                  time_slots)}", con)

    availability = availability.groupby("employee_id", as_index=False)["available"].sum()
    employees = availability["employee_id"].to_list()
    capacity = availability["available"].to_list()

    q = {}
    for employee, capacity in zip(employees, capacity):
        q[employee] = capacity
    return q


def get_objective_val(due_dates: list[int], t: int) -> dict[int, int]:
    """
    Returns a dictionary which contains the difference between a list of due dates and the current date
                                                   t.
    This represents u_i
    """
    u = {}
    for key, val in due_dates.items():
        u[key] = val - t
    return u


def get_n_vehicles(date_id: int,
                   con: sqlite3.Connection,
                   start_hour: int = 6,
                   end_hour: int = 18) -> dict[int, int]:
    """
    Returns a dictionary containing how many vehicles are available at each depot.
    This represents k_{lt}
    """
    h = get_daily_vehicle_capacity(date_id,con, start_hour, end_hour)
    K = {}
    for depot, vehicles in get_depots_and_vehicles(con).items():
        K[depot] = 0
        for vehicle in vehicles:
            if h[vehicle] >= (end_hour - start_hour):
                K[depot] += 1
    return K
```

Appendix B

Code for Retrieving Matrices

```
##########
# Import #
##########
# Data retrieval and wrangling
import pandas as pd
import modules.data_retrieval.retrieve_sets_params as get_sets_params
import sqlite3
from contextlib import closing

# Travel time matrix
from sklearn.metrics.pairwise import haversine_distances
from math import radians
import modules.utils.utils as utils


#############
# Functions #
#############

def get_auditor_depot_matrix(con: sqlite3.Connection) -> dict[int, dict[int, int]]:
    """
    Returns the auditor-depot matrix.
    """
    with closing(con.cursor()) as cur:
        employee_depots = cur.execute("""SELECT
                                            facilities.ID AS facility_id,
                                            employees.ID AS employee_id
                                        FROM facilities
                                        INNER JOIN employees ON facilities.ID = employees.depot_id""").
                                                                                                   fetchall()
        depots = get_sets_params.get_depots(con)
        return_dict = {}
        for depot in depots:
            return_dict[depot] = {}

        for dep, empl in employee_depots:
            for depot in depots:
                if depot == dep:
                    return_dict[depot][empl] = 1
                else:
                    return_dict[depot][empl] = 0
    return return_dict


def get_travel_time_matrix(audits: pd.DataFrame,
                           depots: list[int],
                           con: sqlite3.Connection,
                           km_pr_hour: int = 80) -> dict[dict[float]]:
    """
    Returns the travel-time matrix.
    """
    physical_audits = audits[audits["on_site_audit"] == 1]
    locations = []
    for depot in depots:
        for _, coords in utils.get_lat_long(con, depot).items():
            locations.append([radians(coord) for coord in coords])


    for i, row in physical_audits.iterrows():
        locations.append([radians(row["long"]), radians(row["lat"])])

    distance_matrix = {
    }

    distances = (haversine_distances(locations) * 6371) / km_pr_hour
    for i, id_1 in enumerate([*depots, *physical_audits["ID"]]):
        distance_matrix[id_1]= {}
        for j, id_2 in enumerate([*depots, *physical_audits["ID"]]):
            distance_matrix[id_1][id_2] = distances[i][j]

    return distance_matrix


def get_accomplice_matrix(audits: pd.DataFrame,
                          con: sqlite3.Connection) -> dict[int, dict[int, int]]:
    """
    Returns the accomplice matrix
    """
    employees_tbl = pd.read_sql("SELECT * FROM employees", con)
    skills_tbl = pd.read_sql("SELECT * FROM skills", con)

    employees = employees_tbl["ID"].to_list()
    accomplice_matrix = {}

    for empl in employees:
        accomplice_matrix[empl] = {}

    for i, row in audits.iterrows():
        audit_id = row["ID"]
        audit_type = row["audit_type_id"]
        skill_req = row["required_skill_level"]
        for empl in employees:
            accomplice_matrix[empl][audit_id] = []
            skill_level = skills_tbl[(skills_tbl["employee_id"].astype(int) == int(empl)) & (skills_tbl
                                                                    ["audit_type_id"].astype(int) == int(
                                                                    audit_type))]["skill_level"].to_list()[
                                                                    0]
            if skill_level >= skill_req:
                accomplice_matrix[empl][audit_id] = 1
            else:
                accomplice_matrix[empl][audit_id] = 0
    return accomplice_matrix
```

Appendix C

Code for Date Utility Functions

```
###########
# Imports #
###########
import sqlite3
from contextlib import closing
import modules.data_retrieval.retrieve_sets_params as get_sets_params

#############
# Functions #
#############
def convert_id_to_date(date_id: int, con: sqlite3.Connection) -> str:
    """
    Takes an integer, which represents the ID of a date and returns a string representing that date.
    """
    with closing(con.cursor()) as cur:
        return cur.execute(f"SELECT date FROM dates WHERE ID == {date_id}").fetchone()[0]


def convert_date_to_id(date: str, con: sqlite3.Connection) -> int:
    """
    Takes a string, which represents a date and returns an integer, which represents the ID of that
                                                    date
    """
    with closing(con.cursor()) as cur:
        return cur.execute(f"SELECT ID FROM dates WHERE date == '{date}'").fetchone()[0]


def get_date_time_slots(date_id: int, con: sqlite3.Connection) -> list[int]:
    """
    Takes a date ID and returns a list of integers representing the IDs of the time slots associated
                                                    with that date.
    """
    with closing(con.cursor()) as cur:
        time_slots = cur.execute(f"SELECT ID from time_slots WHERE date_id == {date_id}").fetchall()
    return [time_slot[0] for time_slot in time_slots]


def get_day_type(date_id: int, con: sqlite3.Connection) -> int:
    """
    Takes a date ID and returns a string representing the day type.
    """
    with closing(con.cursor()) as cur:
        return cur.execute(f"SELECT day_type FROM dates WHERE ID == {date_id}").fetchone()[0]


def get_day_types_in_range(start_day: int, end_day: int, con:sqlite3.Connection) -> list[int]:
    """
    Returns a list of day types between a start date and an end date.
    """
    day_types = []
    for day in range(start_day, end_day):
        day_types.append(get_day_type(day, con))
    return day_types


def no_holidays(start_day: int, end_day: int, con: sqlite3.Connection) -> bool:
    """
    Returns a boolean indicating if there is any holidays between a start date and an end date.
    """
    return "holiday" not in get_day_types_in_range(start_day, end_day, con)


def find_day_range(earliest_day: int, latest_day: int, n_days: int, con: sqlite3.Connection):
    """
    Searches for a range of n_days between a start date and an end day, where there is no holidays.
    """
    for day in range(earliest_day, latest_day, n_days):
        if no_holidays(day, day + n_days, con):
            return (day, day + n_days)
    return None


def create_auditor_holidays(first_day: int, last_day: int, con: sqlite3.Connection, n_days = 42) ->
                                                            dict[int, list[int]]:
    """
    Returns a dictionary containing lists of auditor holidays
    """
    auditor_holidays = {}
    for auditor in get_sets_params.get_employees(con):
        if 1 <= auditor <= 5:
            auditor_holidays[auditor] = find_day_range(first_day + 90, last_day, n_days, con)
        elif 6 <= auditor <= 10:
            auditor_holidays[auditor] = find_day_range(first_day + 180, last_day, n_days, con)
        else:
            auditor_holidays[auditor] = find_day_range(first_day + 250, last_day, n_days, con)
    return auditor_holidays
```

Appendix D

Code for Output Utility Functions

```
##########
# Import #
##########

# Data retrieval and wrangling
import sqlite3
import pandas as pd
import numpy as np
import modules.data_retrieval.retrieve_sets_params as get_sets_params

# Relax release dates
from modules.relax_release_dates import relax_release_dates


##############################################
# Helper Functions for The output dictionary #
##############################################
def create_results_dict(con: sqlite3.Connection,
                        first_day: int,
                        last_day:int) -> dict:
    """
    Returns a dictionary, which stores the results of the simulation model
    """
    results_dict = {}
    for t in range(first_day, last_day + 1):
        results_dict[t] = {}
        for e in get_sets_params.get_employees(con):
            results_dict[t][e] = {}
            results_dict[t][e]["audits"] = []
    return results_dict


def update_res_dict(results_dict: dict[int],
                    route_dict: dict,
                    audit_dict: dict,
                    day: int):
    """
    Updates the dictionary, which stores the results of the simulation model
    """
    for auditor, route in route_dict.items():
        results_dict[day][auditor]["route"] = route

    for auditor, audits in audit_dict.items():
        results_dict[day][auditor]["audits"] = audits

    return results_dict


###################
# Output CSV-file #
###################
def generate_simulation_dataframe(con: sqlite3.Connection,
                                  phi: int,
                                  epsilon: int,
                                  first_day: int,
                                  last_day:int) -> pd.DataFrame:
    """
    Is a wrapper for the function relax_release_dates.
    """
    all_audits = get_sets_params.get_all_audits(con)
    all_audits = all_audits[~((all_audits["zip_code"] >= 3700) & (all_audits["zip_code"] <= 3799))]
    all_audits = all_audits[all_audits["audit_type_id"] != 9]
    sim_audits = all_audits.copy()
    sim_audits["audit_date_id"] = -1
    sim_audits["employee_id"] = np.nan
    return relax_release_dates(phi, epsilon, sim_audits, first_day, last_day, con)
```

Appendix E

Code for General Utility Functions

```
###########
# Imports #
###########
import sqlite3
from contextlib import closing

#########################
# Utility functions #
#########################
def get_max_dict_value(my_dict: dict) -> float:
    """
    Returns the highest numerical value in a dictionary
    """
    return max([val for _, val in my_dict.items()])


def get_audit_facility(audit_id: int, con: sqlite3.Connection) -> int:
    """
    Returns the facility ID from an audit ID
    """
    with closing(con.cursor()) as cur:
        return cur.execute(f"SELECT facility_id FROM all_tasks WHERE ID == {audit_id}").fetchone()[0]


def get_lat_long(con: sqlite3.Connection, facility_id: int) -> dict[list[float]]:
    """
    Returns a dictionary which contains the latitude and longitude of a facility.
    The key is the facility ID
    The return value is a list of floats.
    The first list value is the longitude, the last list value is the latitude
    """
    with closing(con.cursor()) as cur:
        facility_locations = cur.execute(f"""
                                 SELECT
                                    facilities.ID AS facility_id,
                                    facilities.lat,
                                    facilities.long
                                 FROM facilities
                                 WHERE facility_id = {facility_id}""").fetchall()
        return_dict = {}
        for facility_location in facility_locations:
            facility_id = facility_location[0]
            lat = facility_location[1]
            long = facility_location[2]
            return_dict[facility_id] = [long, lat]
    return return_dict
```

Appendix F

Code for Generating Bar Charts

```python
import pandas as pd
import sqlite3
import plotly.express as px

def generate_barchart_of_release_dates(df: pd.DataFrame,
                                       y_axis_range: int,
                                       first_day: int,
                                       last_day: int,
                                       con:sqlite3.Connection) -> px.bar:
    """
    Takes a dataframe containing all audits and generates a bar chart showing the distribution of
                                                       release dates.
    The barchart only displays dates between first_day and last day.
    The return value is a plotly figure.
    """

    df["release_date_id"] = df["release_date_id"].astype(int)
    dates_tbl = pd.read_sql("SELECT * FROM dates", con)
    dates_tbl["ID"] = dates_tbl["ID"].astype(int)

    df = df[(df["release_date_id"] >= first_day) & (df["release_date_id"] <= last_day)]
    df = df.merge(dates_tbl, how="left", left_on="release_date_id", right_on="ID")
    df = df.groupby("date").size().reset_index()
    df["date"] = pd.to_datetime(df["date"])
    df = df.rename(columns={0: "phi"})

    fig = px.bar(df, x = "date", y = "phi")
    fig = fig.update_layout(yaxis_range=[0, y_axis_range])
    return fig
```

Appendix G

Code for Data Visualizations

```
###########
# Imports #
###########
from modules.utils.output_utils import generate_simulation_dataframe
from modules.utils.date_utils import convert_date_to_id
from modules.utils.date_utils import convert_id_to_date
from modules.visualizations.generate_visualizations import generate_barchart_of_release_dates
from modules.data_retrieval.retrieve_sets_params import get_all_audits
import sqlite3
import pandas as pd
import numpy as np
import plotly.express as px
import plotly.graph_objects as go


#######################
# Database connection #
#######################
print("Connecting to Database")
con = sqlite3.connect("final_database_master_thesis.db")
cur = con.cursor()

###############################
# Barchart with release dates #
###############################
print("--------------- Bar Chart with release dates-------------------")
# ---- Initialization -----
epsilon = 5
first_day = convert_date_to_id("2022-01-01", con)
last_day = convert_date_to_id("2022-12-31", con)


# ----- Barchart: Original release dates -----
print("Generating Barchart with release dates: original")
generate_barchart_of_release_dates(df = get_all_audits(con),
                                   y_axis_range = 461,
                                   first_day = first_day,
                                   last_day = last_day,
                                   con = con).write_html("outputs/imgs/barchart_original_release_dates.
```

```
                                                                        html")

# ----- Barchart: phi = 100 -----
print("Generating Barchart with release dates: phi = 100")
sim_df = generate_simulation_dataframe(con = con,
                                       phi = 100,
                                       epsilon = epsilon,
                                       first_day = first_day,
                                       last_day = last_day)

fig = generate_barchart_of_release_dates(df = sim_df,
                                   y_axis_range = 461,
                                   first_day = first_day,
                                   last_day = last_day,
                                   con = con)
fig.add_hline(y=100)
fig.write_html("outputs/imgs/barchart_phi_100.html")

# ----- Barchart: phi = 50 -----
print("Generating Barchart with release dates: phi = 50")
sim_df = generate_simulation_dataframe(con = con,
                                       phi = 50,
                                       epsilon = epsilon,
                                       first_day = first_day,
                                       last_day = last_day)

fig = generate_barchart_of_release_dates(df = sim_df,
                                   y_axis_range = 461,
                                   first_day = first_day,
                                   last_day = last_day,
                                   con = con)
fig.add_hline(y=50)
fig.write_html("outputs/imgs/barchart_phi_50.html")

# ----- Barchart: phi = 15 -----
print("Generating Barchart with release dates: phi = 15")
sim_df = generate_simulation_dataframe(con = con,
                                       phi = 15,
                                       epsilon = epsilon,
                                       first_day = first_day,
                                       last_day = last_day)

fig = generate_barchart_of_release_dates(df = sim_df,
                                   y_axis_range = 461,
                                   first_day = first_day,
                                   last_day = last_day,
                                   con = con)
fig.add_hline(y=15)
fig.write_html("outputs/imgs/barchart_phi_15.html")

# ----- Barchart: phi = 6 -----
print("Generating Barchart with release dates: phi = 6")
sim_df = generate_simulation_dataframe(con = con,
                                       phi = 6,
                                       epsilon = epsilon,
                                       first_day = first_day,
                                       last_day = last_day)

fig = generate_barchart_of_release_dates(df = sim_df,
                                   y_axis_range = 461,
                                   first_day = first_day,
                                   last_day = last_day,
                                   con = con)
fig.add_hline(y=6)
fig.write_html("outputs/imgs/barchart_phi_6.html")

########
# Maps #
########

print("--------------- Routing maps -------------------")

# ----- Get Data -----
print("Getting data for maps")
facilities_tbl = pd.read_sql("""SELECT
                                facilities.ID,
                                facilities.name,
                                facilities.n_machines,
                                facilities.n_betting,
                                facilities.facility_type_id,
                                facilities.active,
                                facilities.active,
                                facilities.address,
                                facilities.zip_code,
                                facilities.city,
                                facilities.country,
                                facilities.lat,
                                facilities.long,
                                districts.employee_id,
                                employees.name AS employee_name,
                                employees.depot_id
                             FROM facilities
                             INNER JOIN districts
                                ON (facilities.zip_code = districts.zip_code
                                AND facilities.facility_type_id = districts.facility_type_id)
                             INNER JOIN employees
                                ON employees.ID = districts.employee_id""", con)


depots = pd.read_sql("SELECT * FROM facilities WHERE facilities.facility_type_id = 15", con)

depots["Depot"] = ""

depots.loc[depots["ID"] == 1, "Depot"] = "DGA Odense"
depots.loc[depots["ID"] == 2, "Depot"] = "DGA Trekroner"
depots.loc[depots["ID"] == 3, "Depot"] = "DGA Fredensborg"
depots.loc[depots["ID"] == 4, "Depot"] = "DGA Aalborg"
depots.loc[depots["ID"] == 5, "Depot"] = "DGA H jbjerg"

facilities_tbl["Depot"] = ""

facilities_tbl.loc[facilities_tbl["depot_id"] == 1, "Depot"] = "DGA Odense"
facilities_tbl.loc[facilities_tbl["depot_id"] == 2, "Depot"] = "DGA Trekroner"
facilities_tbl.loc[facilities_tbl["depot_id"] == 3, "Depot"] = "DGA Fredensborg"
facilities_tbl.loc[facilities_tbl["depot_id"] == 4, "Depot"] = "DGA Aalborg"
facilities_tbl.loc[facilities_tbl["depot_id"] == 5, "Depot"] = "DGA H jbjerg"


# ----- Global Routing -----
print("Generating map of global routing")
```

```
fig = px.scatter_mapbox(facilities_tbl,
                        lat="lat",
                        lon="long",
                        opacity=0.7,
                        hover_name= "ID",
                        hover_data={"ID":True,
                                    "lat":False,
                                    "long":False,
                                    },
                        zoom=8)


fig.add_trace(go.Scattermapbox(
    lat = depots["lat"],
    lon = depots["long"],
    mode='markers',
    marker=go.scattermapbox.Marker(
        size=15,
        color='rgb(0, 0, 0)',
        opacity=1
        ),
    name = "Depot"
    ))

fig.update_layout(mapbox_style="carto-positron")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.write_html("outputs/imgs/global_routing.html")


# ----- Local Routing -----
print("Generating map of local routing")
local_dist_df = facilities_tbl.copy()
local_dist_df = local_dist_df.sort_values(by = "employee_id", ascending=True)
local_dist_df["Auditor"] = local_dist_df["employee_id"].astype(str)
fig = px.scatter_mapbox(local_dist_df,
                        lat="lat",
                        lon="long",
                        opacity=0.7,
                        hover_name= "ID",
                        hover_data={"Auditor":True,
                                "lat":False,
                                "long":False,
                                },
                        color="Auditor",
                        color_discrete_sequence=px.colors.qualitative.Light24,
                        zoom=8)


fig.add_trace(go.Scattermapbox(
    lat = depots["lat"],
    lon = depots["long"],
    mode='markers',
    marker=go.scattermapbox.Marker(
        size=15,
        color='rgb(0, 0, 0)',
        opacity=1
        ),
    name = "Depot"
    ))

fig.update_layout(mapbox_style="carto-positron")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.write_html("outputs/imgs/local_routing.html")


# ----- Semi-global Routing -----
print("Generating map of semi-global routing")
depot_viz = facilities_tbl[facilities_tbl["depot_id"].notna()]
fig = px.scatter_mapbox(depot_viz,
                        lat="lat",
                        lon="long",
                        opacity=0.7,
                        hover_name= "ID",
                        hover_data={"depot_id":True,
                                "lat":False,
                                "long":False,
                                },
                        color="Depot",
                        color_discrete_sequence=px.colors.qualitative.Light24,
                        zoom=8)
```

```
fig.add_trace(go.Scattermapbox(
    lat = depots["lat"],
    lon = depots["long"],
    mode='markers',

    marker=go.scattermapbox.Marker(
        size=15,
        color='rgb(0, 0, 0)',
        opacity=1
        ),
    name = "Depot"

    ))

fig.update_layout(mapbox_style="carto-positron")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.write_html("outputs/imgs/semi_global_routing.html")


# ----- Results 2022 -----
print("Cleaning data for map with results")
first_day = convert_date_to_id("2022-01-01", con)
last_day = convert_date_to_id("2022-12-31", con)

# Get Data
results_df = pd.read_csv("outputs/results/model_holidays.csv")
results_df = results_df[results_df["on_site_audit"] == 1]
results_2022 = results_df[(results_df["release_date_id"] >= first_day) &
                      (results_df["release_date_id"] <= last_day)]
results_2022 = results_2022[~((results_2022["zip_code"] >= 3700) & (results_2022["zip_code"] <= 3799))]
depots = pd.read_sql("SELECT * FROM facilities WHERE facilities.facility_type_id = 15", con)

# Get Depots
depots["Depot"] = ""
depots.loc[depots["ID"] == 1, "Depot"] = "DGA Odense"
depots.loc[depots["ID"] == 2, "Depot"] = "DGA Trekroner"
depots.loc[depots["ID"] == 3, "Depot"] = "DGA Fredensborg"
depots.loc[depots["ID"] == 4, "Depot"] = "DGA Aalborg"
depots.loc[depots["ID"] == 5, "Depot"] = "DGA H jbjerg"

# Remove NaN
results_2022 = results_2022.sort_values(by = "employee_id", ascending=True)
results_2022 = results_2022[results_2022["employee_id"].notna()]
results_2022["Auditor"] = results_2022["employee_id"].astype(str)

# Create map
print("Generating map of global results")
fig = px.scatter_mapbox(results_2022,
                        lat="lat",
                        lon="long",
                        opacity=0.7,
                        color="Auditor",
                        hover_name= "facility_id",
                        hover_data={"Auditor":True,
                                    "lat":False,
                                    "long":False,
                                    },
                        color_discrete_sequence=px.colors.qualitative.Light24,
                        zoom=8)


fig.add_trace(go.Scattermapbox(
    lat = depots["lat"],
    lon = depots["long"],
    mode='markers',
    marker=go.scattermapbox.Marker(
        size=15,
        color='rgb(0, 0, 0)',
        opacity=1
        ),
    name = "Depot"
    ))

fig.update_layout(mapbox_style="carto-positron")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.write_html("outputs/imgs/model_routing.html")

# ----- DGA districts 2022 -----
```

```
print("Getting and cleaning data of DGA districts 2022")
first_day = convert_date_to_id("2022-01-01", con)
last_day = convert_date_to_id("2022-12-31", con)

# Get Data
tasks = pd.read_sql("""SELECT
                                all_tasks.ID,
                                all_tasks.facility_id,
                                all_tasks.release_date_id,
                                all_tasks.audit_type_id,
                                facilities.zip_code,
                                facilities.facility_type_id,
                                facilities.lat,
                                facilities.long,
                                audit_types.on_site_audit,
                                districts.employee_id
                           FROM all_tasks
                           INNER JOIN audit_types ON all_tasks.audit_type_id = audit_types.ID
                           INNER JOIN facilities ON all_tasks.facility_id = facilities.ID
                           INNER JOIN districts ON facilities.facility_type_id = districts.
                                                                               facility_type_id AND
                                                                               facilities.zip_code =
                                                                               districts.zip_code
                           """, con)
tasks_2022 = tasks[(tasks["release_date_id"] >= first_day) &
                      (tasks["release_date_id"] <= last_day)]
tasks_2022 = tasks_2022[~((tasks_2022["zip_code"] >= 3700) & (tasks_2022["zip_code"] <= 3799))]
depots = pd.read_sql("SELECT * FROM facilities WHERE facilities.facility_type_id = 15", con)

depots["Depot"] = ""

depots.loc[depots["ID"] == 1, "Depot"] = "DGA Odense"
depots.loc[depots["ID"] == 2, "Depot"] = "DGA Trekroner"
depots.loc[depots["ID"] == 3, "Depot"] = "DGA Fredensborg"
depots.loc[depots["ID"] == 4, "Depot"] = "DGA Aalborg"
depots.loc[depots["ID"] == 5, "Depot"] = "DGA H jbjerg"


# Clean Data
tasks_2022 = tasks_2022.sort_values(by = "employee_id", ascending=True)
```

```
tasks_2022 = tasks_2022[tasks_2022["employee_id"].notna()]
tasks_2022["Auditor"] = tasks_2022["employee_id"].astype(str)


# Visulize map
print("Generating map of dga routing")
fig = px.scatter_mapbox(tasks_2022,
                        lat="lat",
                        lon="long",
                        opacity=0.7,
                        color="Auditor",
                        hover_name= "facility_id",
                        hover_data={"Auditor":True,
                                    "lat":False,
                                    "long":False,
                                    },
                        color_discrete_sequence=px.colors.qualitative.Light24,
                        zoom=8)


fig.add_trace(go.Scattermapbox(
    lat = depots["lat"],
    lon = depots["long"],
    mode='markers',
    marker=go.scattermapbox.Marker(
        size=15,
        color='rgb(0, 0, 0)',
        opacity=1
        ),
    name = "Depot"
    ))

fig.update_layout(mapbox_style="carto-positron")
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.write_html("outputs/imgs/dga_routing.html")

###############
# Gantt Chart #
###############
print("--------------- Gantt Chart -------------------")
print("Getting and cleaning data")
# Get Data and convert it to datetime
all_audits = get_all_audits(con)
all_audits = all_audits.sort_values( "release_date_id").reset_index(drop = True)
all_audits = all_audits[(all_audits["release_date_id"] >= convert_date_to_id("2020-01-01", con)) & (
                                                    all_audits["release_date_id"] <= convert_date_to_id
                                                    ("2024-12-31", con))]
all_audits["release_date"] =  all_audits["release_date_id"].apply(convert_id_to_date, con = con)
all_audits["release_date"] = pd.to_datetime(all_audits["release_date"])
all_audits["due_date"] =  all_audits["due_date_id"].apply(convert_id_to_date, con = con)
all_audits["due_date"] = pd.to_datetime(all_audits["due_date"])

# Visualize Gant chart
print("Generating Gant Chart")
all_audits["Audit"] = all_audits.index
fig = px.timeline(all_audits,
                  x_start="release_date",
                  opacity=1,
                  x_end="due_date",
                  y= "Audit")
fig.update_yaxes(autorange="reversed") # otherwise tasks are listed from the bottom up
fig.update_layout(
    plot_bgcolor='white'
)
fig.write_html("outputs/imgs/gantt_chart.html")


################################################
# Barchart with total audit duration each year #
################################################
print("--------------- Duration barchart -------------------")
print("Getting and cleaning data")
# Get Data and convert it to datetime
all_audits = get_all_audits(con)
all_audits = all_audits.sort_values( "release_date_id").reset_index(drop = True)
all_audits = all_audits[(all_audits["release_date_id"] >= convert_date_to_id("2020-01-01", con)) & (
                                                    all_audits["release_date_id"] <= convert_date_to_id
                                                    ("2025-12-31", con))]
all_audits["release_date"] =  all_audits["release_date_id"].apply(convert_id_to_date, con = con)
all_audits["release_date"] = pd.to_datetime(all_audits["release_date"])
```

```
all_audits["due_date"] =  all_audits["due_date_id"].apply(convert_id_to_date, con = con)
all_audits["due_date"] = pd.to_datetime(all_audits["due_date"])

# Get total audit time for each year
all_audits = all_audits.groupby(all_audits.release_date.dt.year)["duration"].agg(['sum']).reset_index()
all_audits = all_audits.rename(columns={

    "release_date": "year"
})

# Visualize barchart
print("Generating chart")
all_audits = all_audits[all_audits["year"] >= 2020]
all_audits = all_audits[all_audits["year"] <= 2025]
fig = px.bar(all_audits, y = "sum", x="year", text_auto=True)
fig.update_layout(yaxis_range=[0, 25000])
fig.add_hline(y=19800) # 19800 is the maximum capacity
fig.write_html("outputs/imgs/yearly_audit_duration_barchart.html")


################################
# Histogram of audit durations #
################################
print("--------------- Histogram -------------------")
print("Getting and cleaning data")
all_audits = get_all_audits(con)
all_audits = all_audits.groupby(all_audits["duration"])['duration'].agg(['count']).reset_index()

print("Generating Histogram")
fig = px.histogram(all_audits, y = "count", x="duration", nbins=25, text_auto=True)
fig.write_html("outputs/imgs/durations_histogram.html")
```

Appendix H

Code for Relax Release Dates

```python
import pandas as pd
import sqlite3
import modules.utils.utils as utils
import modules.utils.date_utils as date_utils
import modules.data_retrieval.retrieve_sets_params as get_sets_params


def relax_release_dates(phi: int,
                        epsilon: int,
                        all_audits: pd.DataFrame,
                        first_day: int,
                        last_day: int,
                        con: sqlite3.Connection) -> pd.DataFrame:
    """
    Takes a dataframe which contains all audits an returns a dataframe.
    The returned data have had its release dates redistributed according to
    episolon and phi.
    """
    for day in range(first_day, last_day + 1):

        # Sort audits and get audits on the given day
        all_audits = all_audits.sort_values(by = ["due_date_id", "duration", "priority_before_audit"],
                                            ascending= [True, False, False])
        all_audits = all_audits.reset_index(drop = True)
        day_audits = all_audits[(all_audits["release_date_id"] == day)].reset_index(drop = True).copy()
        all_audits = all_audits[(all_audits["release_date_id"] != day)].reset_index(drop = True)

        # If it is not a work day push release dates until nearest work day
        day_type = date_utils.get_day_type(day, con)
        if day_type != "workday":
            i = 1
            day_type = date_utils.get_day_type(day, con)
            while day_type != "workday":
                day_type = date_utils.get_day_type(day + i, con)
                i += 1


            non_edit = day_audits[(day_audits["due_date_id"] + i <= (day - epsilon)) | (day_audits["
                release_date_id"] + i > last_day)].copy
```

```
                                                                ()
            edit_day = day_audits[~((day_audits["due_date_id"] + i <= (day - epsilon)) | (day_audits["
                                                                release_date_id"] + i > last_day))].
                                                                copy()
            edit_day = edit_day.reset_index(drop = True)
            non_edit = non_edit.reset_index(drop = True)

            edit_day["release_date_id"] = edit_day["release_date_id"] + i
            all_audits = pd.concat([all_audits, edit_day, non_edit])

        # If it is a work day push audits release date to next date
        else:
            non_edit = day_audits[day_audits["due_date_id"] <= (day - epsilon)].copy()
            edit_day = day_audits[day_audits["due_date_id"] > (day - epsilon)].copy()
            edit_day = edit_day.reset_index(drop = True)
            non_edit = non_edit.reset_index(drop = True)

            if edit_day.shape[0] <= phi:
                all_audits = pd.concat([all_audits, non_edit, edit_day])

            else:
                edit_day.loc[phi : , "release_date_id"] = edit_day.loc[phi : , "release_date_id"].apply
                                                                    (lambda x : x+1)
                all_audits = pd.concat([all_audits, non_edit, edit_day])
    return all_audits
```

Appendix I

Code for Simulation Model

```
###########
# Imports #
###########
# Data
import numpy as np
import sqlite3
from modules.data_retrieval.retrieve_sets_params import get_all_audits

# Output data
from modules.utils.output_utils import generate_simulation_dataframe
from modules.utils.output_utils import create_results_dict
from modules.utils.output_utils import update_res_dict
import pickle

# Sets and parameters
from modules.data_retrieval.retrieve_sets_params import get_audits_as_list
from modules.data_retrieval.retrieve_sets_params import get_on_site_audits
from modules.data_retrieval.retrieve_sets_params import get_depots
from modules.data_retrieval.retrieve_sets_params import get_employees
from modules.data_retrieval.retrieve_sets_params import get_due_dates
from modules.data_retrieval.retrieve_sets_params import get_objective_val
from modules.data_retrieval.retrieve_sets_params import get_processing_times
from modules.data_retrieval.retrieve_sets_params import get_n_vehicles
from modules.data_retrieval.retrieve_sets_params import get_daily_employee_capacity

# Matrices
from modules.data_retrieval.retrieve_matrices import get_auditor_depot_matrix
from modules.data_retrieval.retrieve_matrices import get_accomplice_matrix
from modules.data_retrieval.retrieve_matrices import get_travel_time_matrix

# Dates
from modules.utils.date_utils import convert_date_to_id
from modules.utils.date_utils import create_auditor_holidays
from modules.utils.date_utils import get_day_type

# Timing
import time

# Optimization
```

```
import gurobipy as gp
from gurobipy import GRB

import copy

#######################
# Database Connection #
#######################
con = sqlite3.connect("final_database_master_thesis.db")
con.execute("PRAGMA foreign_keys = 1")
con.commit()

####################
# Simulation model #
####################
# ------ Initialization Routine -----
# Start parameters
vehicle_start_hour = 6
vehicle_end_hour = 18
km_pr_hour = 80
phi = 15
epsilon = 5
first_day = convert_date_to_id("2022-01-01", con)
last_day = convert_date_to_id("2022-12-31", con)

# Bornholm, Greenland, and holidays
green_land_start, green_land_end = (first_day + 30, first_day + 37)
bornholm_start, bornholn_end = (first_day + 30, first_day + 31)
auditor_holidays = create_auditor_holidays(first_day, last_day, con, 42)

# Get and clean data
all_audits = get_all_audits(con)
all_audits = all_audits[~((all_audits["zip_code"] >= 3700) & (all_audits["zip_code"] <= 3799))] #
                                                    Remove Bornholm
all_audits = all_audits[all_audits["audit_type_id"] != 9] # Remove Greenland

# Dataframe for simulation
sim_audits = all_audits.copy()
sim_audits["audit_date_id"] = np.nan
sim_audits["employee_id"] = np.nan
```

```
sim_audits = generate_simulation_dataframe(con, phi, epsilon, first_day, last_day)

# Multi-day audits
long_audits = sim_audits[sim_audits["duration"] > 8]["ID"].to_list()
long_audits_last_audit = {}
long_audits_auditors = {}
for long_audit in long_audits:
    long_audits_auditors[long_audit] = None
    long_audits_last_audit[long_audit] = 0

# Dictionary for output
results_dict = create_results_dict(con, first_day, last_day)

# ----- Timing routine -----
start_time = time.time()
for day in range(first_day, last_day + 1):

    # Get audits
    daily_audits = sim_audits[(sim_audits["release_date_id"] <= day) &
                              (sim_audits["release_date_id"] >= first_day) &
                              (sim_audits["audit_date_id"] < 0)]

    # Is it a workday and is there any audits?
    if daily_audits.shape[0] == 0:
        print(f"\n ############################ Day {day} ################################### \n")
        print(f"""\n No Audits Available \n""")
        continue

    elif get_day_type(day, con) != "workday":
        print(f"\n ############################ Day {day} ################################### \n")
        print(f"""\n Day is a {get_day_type(day, con)} \n""")
        continue
    else:
        print(f"\n ############################ Day {day} ################################### \n")
        print(f"""\n N AUDITS: {daily_audits.shape[0]} \n""")

    # ----- Event Routine -----
    # Get Data
    t = day
    O = get_on_site_audits(daily_audits)
    V = get_audits_as_list(daily_audits)
    L = get_depots(con)
    E = get_employees(con)
    d = get_due_dates(daily_audits)
    u = get_objective_val(d, t)
    b = get_auditor_depot_matrix(con)
    g = get_accomplice_matrix(daily_audits, con)
    p = get_processing_times(daily_audits)
    K = get_n_vehicles(t, con, vehicle_start_hour, vehicle_end_hour)
    q = get_daily_employee_capacity(t, con)
    c = get_travel_time_matrix(daily_audits, L, con, km_pr_hour)

    # Split multi-day Audits
    for i in V:
        if i in long_audits:
            if p[i] - 5 > 0:
                p[i] = 5
            else:
                long_audits_last_audit[i] = 1

            # Update Accomplice matrix so it is the auditor who performs the multi-day audit
            if long_audits_auditors[i] is not None:
                assigned_auditor = long_audits_auditors[i]
                for e in E:
                    if e == assigned_auditor:
                        g[e][i] = 1
                    else:
                        g[e][i] = 0

    # Set Employee 10 and 11 as unavailable for Greenland expedition
    if green_land_start <= day <= green_land_end:
        q[10] = 0
        q[11] = 0

    if bornholm_start <= day <= bornholn_end:
        q[2] = 0
        q[3] = 0


    # Create Holidays #
    for e in E:
        start_holiday, end_holiday = auditor_holidays[e]
        if start_holiday <= day <= end_holiday:
            q[e] = 0


    # Create Model
    m = gp.Model(f"Danzig-fuller-day-{day}")
    a = m.addVars(E, vtype=GRB.BINARY, name="a")
    y = m.addVars(V, E, vtype=GRB.BINARY, name="y")
    x = m.addVars([*O, *L],[*O, *L], E, vtype=GRB.BINARY, name="x")

    # Not necessary but improves performance
    for e in E:
        for l in L:
            for i in [*O, *L]:
                if b[l][e] == 0:
                    x[l, i, e].ub = 0
                    x[i, l, e].ub = 0

    # Add constraints
    print("Constraint 1: Only leave once from designated depot")
    for e in E:
        for l in L:
            m.addConstr(gp.quicksum(x[l, i, e] for i in O) == b[l][e] * a[e])

    print("Constraint 2: Only return once from designated depot")
    for e in E:
        for l in L:
            m.addConstr(gp.quicksum(x[i, l, e] for i in O) == b[l][e] * a[e])

    print("Constraint 3: Respect vehicle capacity when leaving")
    for l in L:
        m.addConstr(gp.quicksum(x[i, l, e] for e in E for i in O) <= K[l])

    print("Constraint 4: Respect vehicle capacity when returning")
    for l in L:
        m.addConstr(gp.quicksum(x[l, i, e] for e in E for i in O) <= K[l])

    print("Constraint 5: Respect skill levels")
    for i in V:
        for e in E:
            m.addConstr(g[e][i] >= y[i, e])

    print("Constraint 6: No more than 1 audit")
    for i in V:
        m.addConstr(gp.quicksum(y[i, e] for e in E) <= 1)

    print("Constraint 6: Create Tour From --> To")
    for i in O:
        for e in E:
            m.addConstr(gp.quicksum(x[i, j, e] for j in [*O, *L]) == y[i, e])

    print("Constraint 7: Create Tour To --> From")
    for i in O:
        for e in E:
            m.addConstr(gp.quicksum(x[j, i, e] for j in [*O, *L]) == y[i, e])

    print("Constraint 8: Do not go above employee capacity")
    for e in E:
        m.addConstr(gp.quicksum(c[i][j] * x[i, j, e] for i in [*O, *L] for j in [*O, *L]) + gp.quicksum
                                                          (p[i] * y[i, e] for i in V) <= q[e])

    print("Constraint 9: Create assignment variable")
    for e in E:
        for i in O:
         m.addConstr(y[i, e] <= a[e])

    print("Constraint 10: Force y to 1")
    for i in V:
        m.addConstr(d[i] - t >= gp.quicksum(y[i, e] for e in E))

    print("Constraint 11: Conserve flow")
    for j in [*O, *L]:
        for e in E:
            m.addConstr(gp.quicksum(x[i, j, e] for i in [*O, *L]) - gp.quicksum(x[j, i, e] for i in [*O
                                                              , *L]) == 0)

    # Objective function
    m.setObjective(gp.quicksum(y[i, e] * (1/u[i]) for i in V for e in E), GRB.MAXIMIZE)
```

```
# Subtour elimination
def subtour_elimination_callback(model, where):
    if where == GRB.Callback.MIPSOL:
        # Get the solution values
        vals = model.cbGetSolution(model._vars)
        selected_edges = [(i, j, e) for i in [*L, *O] for j in [*L, *O] for e in E if vals[i, j, e]
                                                             > 0.6]
        # Find subtours in the selected edges
        S = get_subtours(selected_edges)

        # Add subtour elimination constraints
        if len(S) != 1:
            for e in E:
                for i in range(len(S)):
                    model.cbLazy(gp.quicksum(model._vars[S[i][j][0], S[i][j][1], e] for j in range(
                                                                    len(S[i]))) <= len(S[i])-1)

def get_subtours(r0):
    """
    This code is adapted from the subtour elimination procedure at this website:
    https://medium.com/swlh/techniques-for-subtour-elimination-in-traveling-salesman-problem-theory
                                                         -and-implementation-in-71942e0baf0c
    """
    r=copy.copy(r0)
    route = []
    while len(r) != 0:
        plan = [r[0]]
        del (r[0])
        l = 0
        while len(plan) > l:
            l = len(plan)
            for i, j in enumerate(r):
                if plan[-1][1] == j[0]:
                    plan.append(j)
                    del (r[i])
        route.append(plan)
    return(route)

# Solve Model
```

```
    m._vars = x
    m.Params.lazyConstraints = 1
    m.Params.NoRelHeurTime = 100
    m.Params.TimeLimit = 2700
    m.optimize(subtour_elimination_callback)

    # ----- Report Generator -----
    # Get Routes
    route_dict = {}
    for auditor in E:
        route_dict[auditor] = [(i, j) for i in [*O, *L] for j in [*O, *L] if x[i, j, auditor].X >= 0.6]

    # Get Audits
    audit_dict = {}
    for auditor in E:
        audit_dict[auditor] = [(i) for i in V if y[i, auditor].X >= 0.6]

    # Print and Write Results
    print(f"""\n - Objective value: {m.objVal} \n\n""")
    for auditor, audits in audit_dict.items():
        audit_time = sum([p[audit] for audit in audits])
        travel_time = sum([c[i][j] for i, j in route_dict[auditor]])
        print(f"##### AUDITOR: {auditor} ######")
        print(f"Availability: {q[auditor]}")
        print(f"Total Audit Time: {audit_time}")
        print(f"Total Travel Time: {travel_time}")
        print(f"Total Time: {audit_time + travel_time}")
        print()

        print(" - AUDITS")
        for audit in audits:
            print(f"\t - Audit: {audit}")
        print()

        route = route_dict[auditor]
        print(f"\n - ROUTE\n")
        if len(route) > 0:
            for i, j in route:
                print(f"\t - FROM {i} --> {j}")
            print()
    # Update  Audits
    for auditor, audits in audit_dict.items():
        for i in audits:
            if i in long_audits:
                sim_audits.loc[sim_audits["ID"] == i, "duration"] = sim_audits.loc[sim_audits["ID"] ==
                                                                    i, "duration"] - p[i]

                if long_audits_auditors[i] is None:
                    long_audits_auditors[i] = auditor

                if long_audits_last_audit[i]  == 1:
                    sim_audits.loc[sim_audits["ID"] == i, "audit_date_id"] = day
                    sim_audits.loc[sim_audits["ID"] == i, "employee_id"] = auditor

            elif i not in long_audits:
                sim_audits.loc[sim_audits["ID"] == i, "audit_date_id"] = day
                sim_audits.loc[sim_audits["ID"] == i, "employee_id"] = auditor
    results_dict = update_res_dict(results_dict, route_dict, audit_dict, day)

print()
print("--- \%s seconds to run ---" \% (time.time() - start_time))

# Save Final results
sim_audits.to_csv("outputs/results/model_holidays.csv", index = False)
with open('outputs/results/results_dict_holidays.pkl', 'wb') as fp:
    pickle.dump(results_dict, fp)
```