POLITECNICO DI TORINO

Mater's Degree in Engineering and Management

Master's Degree Thesis

Service Network Design with Packing Considerations

Supervisors

Prof. Guido PERBOLI

Prof. Sara KHODAPARASTI

Candidate

Samuele MARSANO

April 2025

Acknowledgements

I would want to especially thank you to Professor Sara Khodaparasti for her exceptional politeness, professionalism, and availability. Her honest approach and continuous support made this research trip very enriching, tearing down any distance between teacher and student and creating an exciting and cooperative learning environment.

A special thanks also goes to Professor Guido Perboli for his availability and support. Thanks to him, I had the opportunity to work on a large-scale project of extreme importance, allowing me to tackle stimulating challenges and test myself in a meaningful way.

Thanks to their guidance, I acquired valuable skills and knowledge in the field of optimization techniques and their real-world applications.

Abstract

This thesis is part of a broader research project aimed at developing a mathematical model that integrates two NP-hard problems: the Service Network Design (SND) and the Bin Packing Problem (BPP). The objective is to analyze the economic and logistical benefits of such integration. The existing literature includes only a limited number of studies on the integration of Scheduled Service Network Design (SSND) with BPP, highlighting a significant research gap. Recent contributions have begun to explore the impact of demand loading rules on vehicle utilization and packing constraints in intermodal transport (Kienzle et al., 2024; Morganti et al., 2020), as well as the challenges and advantages of integrating such constraints into network capacity planning (Bruni et al., 2023). However, crucial aspects such as revenue management and capacity selection in SSND remain underexplored. This study aims to bridge this gap by proposing a unified problem formulation that incorporates these elements, providing a framework applicable to contract negotiations and tactical planning in transportation and logistics systems. As part of this research project, the work carried out in this thesis represents a functional yet partial contribution. Specifically, realistic test instances were randomly generated and validated by a VBA script to ensure their compatibility with the corresponding network topology. Subsequently, computational experiments were conducted to examine the integrated model's behaviour across different scenarios, with a particular focus on two key parameters. Finally the same scenarios are tested on the non-integrated model, where the classical SND problem is solved first, followed by the BPP model. The numerical experiments show that the integrated model yields superior outcomes, underscoring the benefits of a method that simultaneously takes the problem's tactical and operational aspects into account.

Table of Contents

List of Tables

List of Figures

Acronyms

SND
: service network design

BPP
: bin packing problem

LTL
: less-than-truckload

OR
: operation research

LP
: liinear programming

ILP
: integer linear programming

OD
: origin-destination

3D-BPP
: three dimensional bin packing problem

NLP
: non liinear programming

MILP
: mixed-integer linear programming

Introduction

Modern economies depend on effective freight transportation since it affects societal well-being, regional growth, and international trade. The difficulty of balancing service quality, environmental sustainability, and economic success is at the heart of this intricate system. The Service Network Design (SND) methodology offers a solution by focusing on optimized planning of transportation networks for carriers employing consolidation strategies. SND is built on consolidation networks, which are essential for lowering operational expenses and enhancing logistical effectiveness.

Consolidation is a widely spread strategy aims to increase operational and economic efficiency for shippers and carriers by consolidating freight with different origins and destinations into the same units (vehicles, containers, etc.) for their entire or partial journeys. The unit shipping cost and the travel duration should thus be reduced, which is beneficial to all parties involved. Railroad, Less-than-Truckload (LTL) motor carriers, shipping companies moving containers on oceans, seas, rivers, and canals, postal services and express couriers, logistics service providers, as well as synchro modal, city logistics, and physical internet systems are prime examples of consolidation-based carriers moving a large and valuable part of the world trade (consumer goods in particular) over short, medium, long, and intercontinental distances [1].

SND acts as a link between operational planning, which oversees day-to-day operations, and strategic planning, which establishes long-term goals, in an increasingly intricate logistics system. By addressing the intricate decisions associated with these operations, SND plays a pivotal role in ensuring that carriers achieve both profitability and service reliability. The significance of SND is further amplified in today's unpredictable environment. Political instability, fluctuating demand patterns, and supply chain disruptions, such as those caused by global health crises, underscore the need for resilient and adaptable transportation systems. SND provides the tools to design such systems, enabling carriers to respond effectively to changing conditions while maintaining operational stability.

The SND methodology is grounded in Operations Research (OR), utilizing combinatorial optimization techniques to solve large-scale problems. These problems are typically characterized by the interdependence of decisions across the network,

such as service scheduling, resource allocation, and freight routing. The ability to manage these interconnected dynamics is essential for improving efficiency, reducing costs, and ensuring an adequate level of service.

The models used in Supply Network Design can be classified along two main dimensions: time (static vs. time-dependent models) and uncertainty (deterministic vs. stochastic models):

1. Static Models: system factors like demand, operating expenses, and resource availability are assumed to stay constant over time by static models. Long-term assessments where temporal changes have little effect on strategic choices are best suited for these models. For instance, a static model may be used to figure out where industrial facilities or warehouses should be placed in an existing logistics network.

2. Time-dependent models: on the other hand, consider temporal variations in system parameters. These are particularly useful for problems where parameters such as demand, resource availability, or costs vary significantly over time. Time-dependent models are essential for managing perishable goods transportation, fleet operation planning, or distribution systems with strict time constraints, such as "just-in-time" deliveries.

3. Deterministic models: use fixed, known input values for parameters such as demand, costs, capacities, and delivery times. These models are relatively simpler to solve, as they do not account for uncertainty, but they may be less realistic in complex scenarios. They are often used for scenario analyses to evaluate the impact of strategic decisions under well-defined assumptions.

4. Stochastic models: take into account the uncertainty around one or more important characteristics, including resource availability, transportation costs, or future demand. In order to find solutions that function well even in uncertain situations, these models frequently employ methods like Monte Carlo simulations, robust optimization, or stochastic programming. They also use probability distributions to describe uncertainty. Stochastic models are particularly critical in dynamic and unpredictable environments, such as global transportation networks affected by external events, weather conditions, or economic fluctuations.

Solving SND problems relies on applying advanced optimization techniques, often derived from combinatorial optimization. Some of the most commonly used techniques are:

- LP and ILP: problems involving decisions represented by continuous or integer variables, such as the number of distribution centers to open or the quantities

of items to move, are solved using linear programming (LP) and integer linear programming (ILP).

- Dynamic Programming: problems requiring sequential decision-making across time are addressed by dynamic programming.
- Metaheuristic Algorithms: techniques like genetic algorithms, simulated annealing, and tabu search are frequently employed to solve optimization problems that are too complex to accurately solve in a reasonable amount of time.
- Multi-objective optimization: makes it possible to reconcile competing goals, such minimizing costs while enhancing service quality.

Chapter 1

Problem Description

1.1 Service Network Design

SND is a methodology focused on designing and optimizing service networks for freight transportation. This involves making crucial decisions regarding route selection, resource planning, and management of freight flows. The primary objective is to minimize the total system costs, including both fixed costs for activating services and variable costs associated with freight transportation. In practical terms, SND addresses complex problems related to managing transportation networks. Among the main challenges are:

- Service network design: determining connections between origins and destinations, including potential intermediate stops for load consolidation.
- Resource allocation: optimizing the use of vehicles, terminals, and other infrastructure.
- Flow optimization: planning the most efficient routes to meet demand while balancing costs and service levels.

A key characteristic of SND is its integrative approach: it does not focus solely on optimizing a single aspect of transportation but considers the entire system as an interconnected whole. This approach allows for effectively managing the interactions among various system components, such as physical infrastructure, transportation demand, and the services offered. The design and optimization of a service network revolve around three fundamental components: the physical network, the demand, and the service network. Each component plays a crucial role in ensuring the efficiency and reliability of the system as a whole.

1.1.1 Physical Network

The physical network represents the underlying infrastructure on which carriers operate. It encompasses all the tangible elements of the system, providing the framework for transportation and logistics activities. The physical network can be broken down into two primary elements:

1. Nodes: serve as critical points within the network where freight is handled, consolidated, or transferred. These include:

   - Terminals: regional terminals act as points of origin or destination for freight. Additionally, they may function as transfer points where goods are unloaded and reloaded for further transportation.
   - Hubs: designed to combine flows from several terminals, hubs are facilities positioned in key locations. Hubs enable economies of scale, particularly in long-haul transportation, by consolidating freight from multiple locations.. For example, combining smaller shipments into larger loads at hubs improves efficiency and lowers overall transportation costs.

2. Arcs: represent the connections between nodes and the transportation links within the network. These connections can include:

   - Highways: connecting hubs and terminals, highways are essential for long-distance vehicle freight transportation.
   - Railways: railways provide cost-effective and sustainable options for bulk commodities or long-distance freight.
   - Maritime Routes: crucial to international trade, these routes connect ports across continents and countries.

   Each arc is defined by several attributes that influence its performance, such as:

   - Capacity: the maximum freight volume that can be transported along a specific arc. Bottlenecks and delays may result from limited capacity.
   - Travel Time: the time required to move goods between nodes. This is influenced by factors such as distance, speed limits, and route conditions.
   - Costs: the expenses associated with using the arc, which may include fuel costs, tolls, or maintenance fees.

There is a physical network $\mathcal{G}^{PH} = (\mathcal{N}^{PH}, \mathcal{A}^{PH})$ that shows the physical infrastructure system. Each node $\eta \in \mathcal{N}^{PH}$ is a terminal, a hub, or a region,

and each arc $a \in \mathcal{A}^{PH}$ shows the possibility of a direct connection between two terminals, which is shown by its starting and ending nodes several attributes are associated with the network $\mathcal{G}^{PH}$, including capacities and costs expressed in terms of vehicles, convoys, containers, or freight volumes. The arc length and the infrastructure quality status are also considered [2].

The physical organizational model is the so called "hub and spoke" system, also known as the star network. The hubs (primary nodes) act as collection points and are directly connected to each other. Additionally, all secondary nodes provide a connection (arc) to their respective hub. The figure 1.1 shows a graphical representation of the "hub and spoke" network.

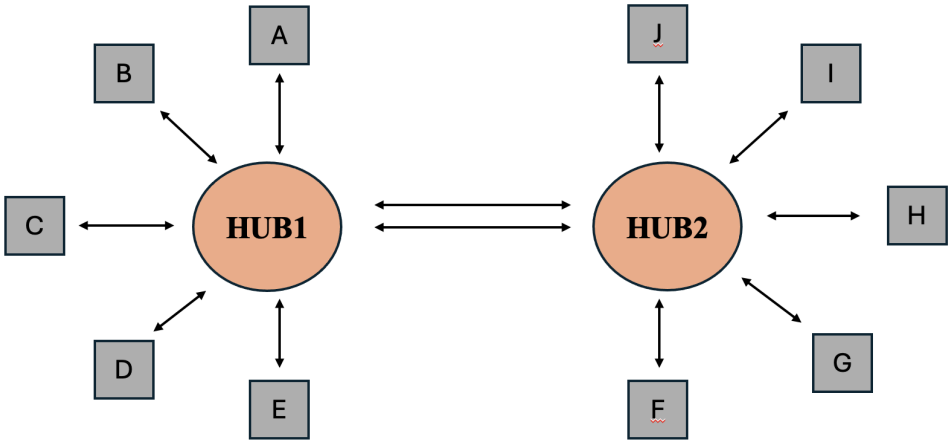


Figure 1.1: Hub and Spoke service network

This approach allows for the centralization of operations, reducing costs and improving the efficiency of connections. This configuration reduces direct routes between nodes, concentrating operations at the central hub to enhance efficiency.

1.1.2 Demand

Demand plays a crucial role as it refers to the transportation needs of shippers, which are represented by origin-destination (OD) flows. These flows indicate the movement of goods between specific points, outlining the required transportation services for various types of freight. Each demand instance not only includes the quantity of goods to be transported but also details the type, characteristics, and specific conditions associated with the freight. For example, the demand could

include bulk materials, perishable goods, hazardous substances, or specialized freight that requires particular care in handling and transport.

By evaluating trends in demand, planners can optimize routes, minimize costs, and allocate the right type of resources to meet these needs effectively. In practice, demands can be aggregated to simplify the modeling process. Similar demands, such as those originating from the same geographical area or involving similar types of freight, can be combined into a single commodity flow. This aggregation reduces the complexity of the transportation network, making it easier to design and optimize. Demand is thus generally defined as the requests for transportation of a set k of OD commodities, each commodity being an aggregation of shippers with similar characteristics in terms of origin, destination, timing, type, product, handling cost, and fare.

The demand characterization of most SND contributions in the literature concerns a single category of customers, which are strongly believed to require service regularly during the coming season. We identify this category as contract-based, with associated commodity set $\mathcal{K}^C$. This demand must be satisfied by the designed service network. Consequently, the total associated revenue R^C is assumed to be a given constant, to be ignored in planning and the SND minimization of the total operation costs. To illustrate the capability of the SND methodology to account for demand segmentation and service differentiation, we define a second demand category, identified as irregular potential, with associated commodity set $\mathcal{K}^I$. This category represents the aggregated volume of estimated demand the carrier receives on a regular basis, while the plan is executed, from shippers without formal understandings. This type of demand may be accepted or not. Let cat(k) stand for the category of demand, cat(k) = C or I when $k \in \mathcal{K}^C$ or $\mathcal{K}^I$, respectively. The entire demand set is then $K = \mathcal{K}^C \cup \mathcal{K}^I$, each commodity $k \in \mathcal{K}$ meaning a request to move a quantity of freight vol_k from its origin O(k) to its destination D(k) [2].

1.1.3 Service Network

The service network represents the operational framework through which carriers fulfill transportation demands. It encompasses all the potential services that can be activated to move goods across the network efficiently. Each service is characterized by specific attributes that determine its suitability for varying types of freight, routes, and cost structures. These attributes include:

1. Mode of Transportation: refers to the type of transportation that is used, such as air, sea, train, or road. Each mode has specific advantages and limitations. For example, rail and maritime modes are more economical for long-haul or bulk goods, while road transportation is flexible and best for short distances.

2. Route Specification: the route specifies the path connecting the origin and destination nodes. Routes can be direct or include intermediate stops for loading, unloading, or cargo transfers. Route selection is affected by a number of variables, including cost, network congestion, and distance. Route optimization can drastically cut down on operating costs and transit times.

3. Capacity: the maximum weight or volume that a service can manage is referred to as its capacity. Capacity constraints must be carefully managed, as exceeding limits can cause delays, additional costs, or the need for extra services.

4. Cost Structure: each service incurs both fixed and variable costs. Fixed costs include expenses such as activating a specific service, maintaining vehicles, or hiring crews. Variable costs depend on the volume or weight of goods transported, fuel consumption, and distance traveled.

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be the potential service network, defined based on the physical nodes of the system $\mathcal{N}^{PH}$ and the set of potential services Σ within the context of the carrier resources, operation rules, economics, and service goals. A service $\sigma \in \Sigma$ follows a path in the physical network from its origin $\mathrm{O}(\sigma)$ to its destination $\mathrm{D}(\sigma)$. Several other terminals may be located along this path. A direct service passes by these terminals without stopping. The service is then represented as a single arc $a \in \mathcal{A}$, and $\mathcal{A} = \Sigma$ when all potential services are single-leg. A multi-leg service halts at intermediary terminals on its route to drop and pick up loads. When convoys are involved (e.g. rail, road, and barge trains), the service may also stop to pick up or drop off individual or groups of vehicles (e.g., car or blocks for railroads and trailers for LTL motor carriers operating multi-trailer road trains). The service route is then described by the sequences of $n(\sigma)$ terminal stops and $n(\sigma)$ - 1 service legs connecting them [2].

The entire effectiveness and financial success of the transportation system depend heavily on the planning and administration of the service network. A well-designed network can handle changes in freight volume, seasonal demand, and service interruptions while guaranteeing that demand is satisfied with the fewest possible delays and expenses. Carriers must constantly assess and modify their service networks to meet changing consumer demands, legal specifications, and technical breakthroughs. Formally, then, the basic arc-based SND formulation seek to:

$$\min \quad \sum_{\sigma \in \Sigma} f_\sigma y_\sigma + \sum_{k \in \mathcal{K}} \sum_{a \in \mathcal{A}} c_a^k x_a^k \tag{1.1}$$

$$\text{s.t.} \quad \sum_{a \in \mathcal{A}_\eta^+} x_a^k - \sum_{a \in \mathcal{A}_\eta^-} x_a^k = d_k, \qquad \eta \in \mathcal{N}, k \in \mathcal{K}, \tag{1.2}$$

$$\sum_{k \in \mathcal{K}} x_a^k \leq u_a y_{\sigma(a)}, \qquad a \in \mathcal{A}, \tag{1.3}$$

$$x_a^k \leq u_a^k y_{\sigma(a)}, \qquad a \in \mathcal{A}, k \in \mathcal{K}, \tag{1.4}$$

$$y_\sigma \in \{0,1\}, \qquad \sigma \in \Sigma, \tag{1.5}$$

$$x_a^k \geq 0, \qquad a \in \mathcal{A}, k \in \mathcal{K}. \tag{1.6}$$

Where $\mathcal{A}_\eta^+ = \{(\eta, \eta') \in \mathcal{A}\}$ and $\mathcal{A}_\eta^- = \{(\eta', \eta) \in \mathcal{A}\}$. Define the sets of outgoing and incoming arcs for node $\eta \in \mathcal{N}$, respectively, while $d_k = vol_k$ at the demand origin $\eta = O(k) - vol_k$ at the demand destination $\eta = D(k)$, and zero at all other nodes [1].

The objective of the SND minimizes the total cost of operating the system, computed as the sum of the fixed costs associated with selecting the service network and the variable cost of transporting commodities using the selected services. Equations 1.2 are often referred to as flow-balance constraints and ensure that all of a commodity's demand departs from its origin, arrives at its destination, and departs from any other locations at which it arrives. The expression on the left-hand side of the linking constraints 1.3 computes the total flow traveling on arc $a \in \mathcal{A}$, whereas the expression on the right-hand side gives the global arc capacity provided by the corresponding service (selected or not). The commodity-disaggregated linking constraints are given by 1.4. Constraints 1.5 and 1.6 define the variable domains [1].

1.2 Bin Packing Problem

The BPP is a combinatorial optimization problem that has been studied greatly due to its theoretical complexity and practical usefulness in many different disciplines. At its core, the problem is a collection of objects, each with a weight, and a set of bins with a fixed capacity. In addition to ensuring that the total weight of the goods in each bin does not exceed its capacity, the assignment should minimize the number of bins used. This seemingly simple problem has deep implications and several real-world applications. Its relevance to computer science and operations research stems from its role in modeling resource allocation, storage optimization, and logistics planning.

1.2.1 Mathematical Formulation

The mathematical formulation below corresponds to that of Martello and Toth, who in 1990 decided to compile all knapsack problems in a book, as well as the bin packing problem, which is not usually included in the knapsack area, but can be interpreted as a multiple subset sum problem where all containers have the same capacity c, all items must be selected, and it's desired to minimize the number of containers used. Given n items and n knapsacks (or bins) with

$$w_j = \text{weight of item j},$$

$$c = \text{capacity of each bin}$$

assign each item to one bin such that the total weight of the items in each bin, does not exceed c and the number of bins used is a minimum. A possible mathematical formulation of the problem is:

Minimize

$$z = \sum_{i=1}^{n} y_i \tag{1.7}$$

Subject to

$$\sum_{j=1}^{n} w_j x_{ij} \leq c y_i, \quad i \in N = \{1, \ldots, n\} \tag{1.8}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j \in N \tag{1.9}$$

$$y_i \in \{0,1\}, \quad i \in N \tag{1.10}$$

$$x_{ij} \in \{0,1\}, \quad i \in N, j \in N \tag{1.11}$$

where

$$y_i = \begin{cases} 1 & \text{if bin } i \text{ is used;} \\ 0 & \text{otherwise,} \end{cases}$$

$$x_{ij} = \begin{cases} 1 & \text{if item } j \text{ is assigned to bin } i; \\ 0 & \text{otherwise.} \end{cases}$$

We will suppose, as is usual that the weight w_j are positive integers. Hence, without loss of generality, we will also assume that:

- c is a positive integer
- $w_j \leq c$ for $j \in \mathcal{N}$ [3].

1.2.2 Importance and Applications

The BPP presents a fundamental trade-off between simplicity and computational complexity. Even though the rules of the problem are simple to grasp, finding the best solution gets much more difficult as the number of items goes up. This complexity has led to many years of research aimed at finding effective solution methods. One interesting thing about the BPP is how it is connected to other combinatorial optimization problems, like the knapsack problem and scheduling problems. These connections have resulted in insights from different fields, where methods created for one issue can often be changed or used for another. For example, the BPP has a lot in common with packing and cutting problems, which also focus on optimizing resource use while following strict rules. The BPP has been studied a lot since it was first formally defined. Much of the literature focuses on approximation algorithms due to the difficulty of solving large instances exactly.

Heuristic and metaheuristic methods, like First Fit, Best Fit, and genetic algorithms, are commonly used to find near-optimal solutions in an efficient way. These methods give up some accuracy for faster results, which makes them really useful in situations where time is important, such as making quick decisions in logistics. Beyond heuristic methods, researchers have looked into the theoretical parts of the problem, like finding lower bounds and recognizing structural properties. The goal of these efforts is to help us understand how difficult the problem is and to create standards for measuring how well algorithms perform. Exact algorithms are not very common because they require a lot of computing power, but they are still an important topic of research for small or very limited cases.

The significance of the BPP lies in its versatility, as it is applicable across various industries.

1. Logistics and Transportation: design of packaging is an application area for the BPP. The success of IKEA in minimizing inventory space is attributed to its clever “flat” package designs. This example demonstrates the fact that, instead of trying to store predetermined size and shape packages in a fixed area of storage, the decision makers of the supply chain may come up with optimal designs for packaging of items in order to maximize space utilization [4].

2. Manufacturing: has applications in optimizing raw material cutting. For instance, in steel manufacturing, large sheets are cut into custom-sized pieces for use in construction, automotive, and machinery industries. Similarly,

in textile production, fabric rolls are cut into patterns for making clothes, upholstery, or other items. In these situations, the challenge is not just to reduce waste but also to improve the cutting process. This helps to lower machine idle times, stick to production schedules, and consider other factors like the grain direction in fabrics or areas with defects in raw materials.

3. Project scheduling: is an issue in all stages of a supply chain. It is concerned with the efficient utilization of given resources (time, people, budget) to complete a sequence of activities, which have precedence relations among themselves. Obviously, the decision-makers in all stages of the supply chain, starting from suppliers' tier reaching to the customers, have to deal with their own projects. Project scheduling resembles the BPP in many aspects. The problem of task allocation along a timeline can be modeled and solved as a multi-dimensional BPP. The specific algorithms developed for the BPP may be adapted effortlessly in this case. Manufacturing involves cutting raw materials like steel or fabric into smaller pieces to reduce waste [4].

These applications extend beyond considering only weight or volume.

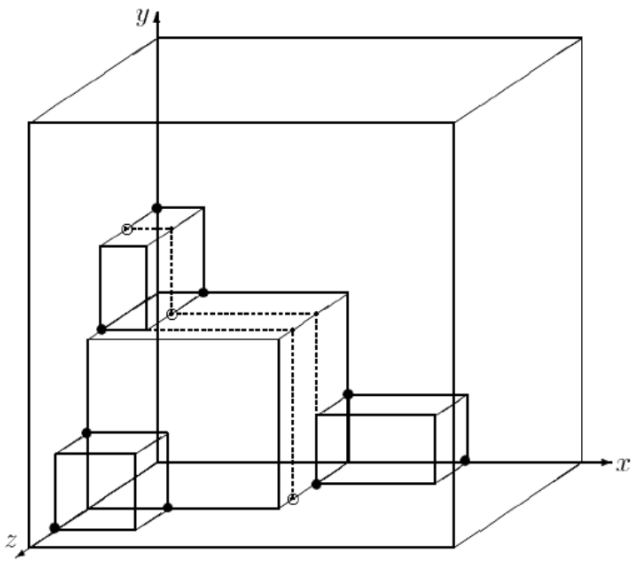


Figure 1.2: Three dimensional single bin filling [5]

Multidimensional variants introduce additional constraints, such as item dimensions, shapes, or orientation, which mirror the complexities faced in real-world scenarios. Figure 1.2 graphically illustrates this complexity by depicting the Three-Dimensional Bin Packing Problem (3D-BPP). In summary, the BPP is not only an

abstract mathematical challenge. This is a practical and important problem that helps improve how resources are used and how efficient different industries can be. The theoretical basis is based on mathematics, but its use in real-life situations like logistics, manufacturing, transportation, and scheduling shows how important it is in practice. Efficient resource packing, whether for physical goods, raw materials, or shipments, directly impacts cost-effectiveness, sustainability, and operational efficiency in these industries.

The adaptability of BPP shows interesting opportunities for combining it with more complicated logistical issues, like SND. As seen, in SND, resources must be distributed across various services within a network. This is often limited by how much capacity we have, the level of demand, and the costs of operating. The SND includes many important factors like routing, scheduling, and service-level needs. However, the main ideas of BPP as resource allocation, optimization, and constraint satisfaction fit nicely with what SND aims to achieve. So, by using BPP optimization methods, it's possible to think of solutions that not only solve regular resource-packing problems but also make service networks work better, which can lower costs and improve how the system performs.

The ongoing progress in BPP research, such as creating new algorithms and looking into multidimensional or irregular packing methods, offers even more possibilities for addressing the complicated issues found in SND. As technology and computing power improve, combining BPP techniques with service network design could lead to more efficient, sustainable, and cost-effective solutions. The integration of these optimization methods shows how theoretical knowledge connects with practical uses. It provides new ways for industries to use resources better, reduce waste, and improve the overall efficiency of service networks.

1.3 The integration of Bin Packing Problem in Service Network Design

As integrated vehicle routing problems combine optimization problems that are usually NP-hard by themselves, the prevailing attitude among operations researchers has been, until recently, to tackle each problem independently, at the expense of global optimization. On the other side, combining two, or more, hard problems causes a significant increase of the computational burden required, but tends to provide considerably better solutions than solving optimally each problem, often even if the integrated problem is solved with a heuristic [6].

Actaully, in today's literature, there is very little available on the integration of the two problems. Few studies have explored the experimentation of such integration, but those who have, even if with some differences compared to the model that will be presented in this thesis, have shown that combining the two

problems brings clear advantages of various kinds.

Flamand et al., propose a new variant of the transportation problem with multiple types of commodities to be transported from multiple supply nodes to multiple demand nodes via several vehicles of different types that may be available at supply nodes with different quantities. These vehicles have different capacities and each type of commodity accommodates different amount of space (weight) on them. It is noteworthy that if the quantities of the various commodities to be shipped by each supplier are fixed, the problem reduces to the well-known variable size BPP (or equivalently, the multiple length cutting stock problem) [7].

Coté et al. state that loading issues are closely related to multi-dimensional packing problems, especially extensions of the classical (one dimensional) BPP. In their paper they have considered the capacitated vehicle routing problem with two-dimensional loading constraints, which is an integrated problem where the capacitated vehicle routing problem is combined with the problem of finding a feasible loading pattern for a set of rectangular-shaped items. It's proposed a solution approach that addresses the integrated problem by means of an exact algorithm and compared such approach with three not integrated approaches that consider the routing and the loading aspects of the problem separately. It's shown that the cost of a solution obtained with a not integrated approach may be as large as twice the cost of an optimal integrated solution [6].

Moreover, Hewitt et Lehuédé estabilish that formulating with consolidations also facilitates modeling issues that have not yet been addressed in the literature, such as bin-packing considerations when computing vehicle capacity needs. In addition, the proposed modeling technique for bin-packing considerations in a consolidation-based formulation yields instances that are easier to solve than those wherein capacity is modeled in an aggregate sense [8].

Including packaging considerations, like Bin Packing, into distribution networks is an important innovative idea in solving complicated logistical issues. This method allows us to define consolidations directly on the physical network, which removes the need for formulations that rely on space-time networks. The result is a compact formulation particularly suited for small-scale problems. It works better than the usual space-time network formulations that have knapsack-style capacity limits. The importance of this integration is that it can overcome the traditional way of doing things, where tactical and operational modeling are handled separately. This separation can cause problems with cost estimation and finding ways to consolidate effectively.

From the first evidence in the literature, the integrated approach provides:

- Better consolidation opportunities: by combining the two problems, it helps to optimize routes and demand allocation at the same time, which reduces waste and makes better use of resources.

- More realistic costs: taking into account the operational complexities during the tactical modeling phase allows for more precise and relevant cost estimates.
- Lower computational complexity: the compact formulation makes it easier to solve small problems without solving in an independent manner the SND problem and the BPP.

The two models have been tested in this thesis and the results obtained from the comparison of the two models will be presented in the last section. he findings indicate that the integrated model is not only computationally less expensive, since it eliminates the need to solve an additional problem, but also enhances profit maximization through improved consolidation.

Chapter 2

SND integrated with BPP

In this chapter, the integrated model is presented as an improved and more efficient version of the non-integrated and independent models found in the literature. In the model, carriers are part of the supply side, offering transportation capacity through what is called scheduled services.

2.1 Model

Each service has its own features:

- Starting and ending terminals: showing the physical nodes of departure and arrival.
- Departure time and arrival time: respectively from the starting terminal to the destination terminal.
- Service duration: the time needed to travel directly from the starting point to the destination.

From the perspective of an individual service, transportation is considered unimodal, as each service uses a single mode of transport for the connection between the origin and destination. As a result, the physical network representation maintains this unimodal structure. However, from the perspective of the carriers, transportation can take on a multimodal nature, especially when carriers act as intermediaries and combine multiple modes to move loads between terminals. Each service is uniquely labeled and associated with a specific mode of transport, determined by its nature (e.g., land, maritime, or air) and the corresponding infrastructure (e.g., railway, road, etc.). The capacity offered by a service is expressed in terms of pre-reserved space blocks (also known as allotments). These blocks correspond to standardized load units, each characterized by a specific type, such as 20- or 40-foot containers,

railcars, or vehicles with multiple loading spaces. Two additional characteristics of load units are their costs and capacities, measured in terms of volume, weight, or length. Each service has a limited number of load units, which belong to a specific mode of transport. However, not all types of load units are compatible with every service. Clearly, a single carrier can perform multiple services. Before defining the service cost, it is crucial to first explain what contracts are and how they are managed.

Carriers offer service contracts to shippers, who give all important information about their transportation needs to the intermediary platform. All requests for supply and demand are provided in the system before the planning period begins. The platform is important for managing the system because it makes important decisions, as:

1. Acceptance or rejection requests from shippers that are not part of a contract, making sure that the goods are picked up from the origin after the release time.

2. Delivery occurs before the deadline. Moreover, all contractual demands, i.e., those for which a contract with the shippers exists, must be fulfilled.

Figure 2.1 shows the interactions between shippers and carriers from the perspective of this intermediary. The intermediary platform leverages spatial and temporal consolidation of multi-shipper requests to design an optimized service network. This network offers several advantages, primarily increased efficiency by eliminating the need to explicitly model each time interval, thereby simplifying both problem comprehension and management. The approach also offers flexibility in incorporating constraints, allowing for the inclusion of complex rules related to vehicle capacity and packing requirements. Unlike traditional methods based on time-space networks, the proposed compact representation is both simpler and more efficient, particularly for smaller-scale problems. However, for larger problems, a hybrid model is introduced, combining consolidation-based modeling with a traditional time-space representation, enabling the management of various problem configurations depending on the number of consolidations possible within each time period.

The pysical network is similar to the one presented in section 1.1 of chapter 1 with some updates. Operations at terminals in different periods are modelled as time-stamped nodes of the form $(n, t) \in \mathcal{N}$ and time-stamped arcs. There are two types of arcs in $\mathcal{A}$, the sets of service arcs $\mathcal{A}^{\Sigma}$ and the sets of holding arcs $\mathcal{A}^{H}$. $\mathcal{A} = \mathcal{A}^{\Sigma} \cup \mathcal{A}^{H}$. A service arc, joining nodes (n, t) and (n', t'), models the operation of a single-leg service $\sigma \in \sum$ between its origin $o(\sigma) = n$ and destination $d(\sigma) = n'$, starting at time $\alpha(\sigma) = t$ and arriving at time $\beta(\sigma) = t'$. A holding arc, joining nodes (n, t) and $(n, t + 1)$, models the possibility of holding the freight

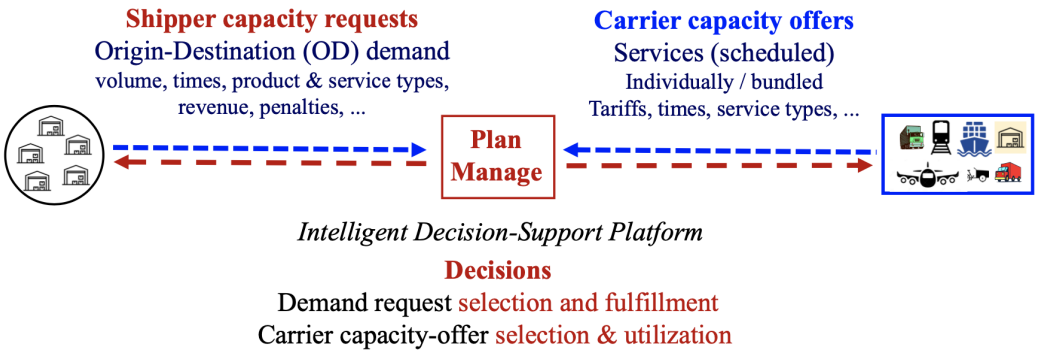


Figure 2.1: M1M system [9]

at node n from period t to $t+1$. Each potential service has a fixed service cost f_σ. Each service σ has associated a set of loading units J_σ. In what follows, we apply the generic term bin to refer to the loading unit associated with services. Let $J = \bigcup_{\sigma\in\Sigma} J_\sigma \; J_\sigma \cap J'_\sigma = \emptyset, \forall\sigma \neq \sigma' \in \Sigma$ be the set of loading units available in the service network. The set of bin types is denoted by $\Pi = \{1, 2, \ldots, \pi, \ldots, n_\Pi\}$ and each bin $j \in J$ has a type ($\phi(j) \in \Pi$) that is characterised by capacity Q_π , fixed cost, c^F_π and variable unitary cost c^a_π . Each service σ has a global capacity U_σ, that is defined on the basis of the characteristics of the system considered [10].

Demand is categorized in two types: contract demand K^C, which must be satisfied and non contract demand K^{NC} which could be dissatisfied. So $K = K^C \cup K^{NC}$. Each item has a size v_i (expressed in the same unit as the container capacity), while the size of the shipper's request is the sum of the volumes of its items, indicated as d_i. All items associated with the same request are available simultaneously (availability time) at the origin and must be delivered to the final destination by the common due date. Each potential request from non-contract customers can be accepted or rejected based on its revenue. In contrast, contract customer requests must be accepted. Clearly, accepting a request implies delivering all its items.

2.1.1 Mathematical Formulation

We consider the following sets of decision variables: $y_\sigma \in \{0,1\}, \sigma \in \Sigma$, for the selection of service σ; $z_j \in \{0,1\}, j \in J$, selects or not bin j; $x^i_{aj} \in \{0, 1\}$, represents the possible assignment of item $i \in I(k)$ to bin j of service σ, and it is defined for given demand $k \in \mathcal{K}$ $\forall a \in \mathcal{A}^\Sigma, i \in I(k), j \in J_{\sigma_a} | \phi(j) \in \Pi_k$. The binary variables $w^i_a \in \{0, 1\}, a \in \mathcal{A}^H$, $i \in I$, indicate if item i is held on arc a, $\lambda_k \in \{0, 1\}, k \in \mathcal{K}$, whether or not demand k is accepted.

The objective function (2.1) maximizes the total profit expressed as a difference between revenues and costs. The total cost of selecting services, securing bins, holding items at terminals, transporting items through the designed system is considered. Constraints (2.2) ensure that each item is routed from its origin node to its destination node, respecting the temporal constraints. Constraints (2.3) enforce a feasible assignment of items to bins, respecting the bin capacity. Constraints (2.4) express the service maximum capacity in terms of the total capacity of bins operating on that service. Constraints (2.5) link the z and y variables ensuring that only if a service is opened, then we can use the bins of that service. Constraints (2.6) require that the demand of all regular customers are totally accepted and delivered. Finally, constraints (2.7)-(2.11) express the nature of the variables.

$$\text{Maximize} \sum_{k\in\mathcal{K}} p_k\lambda_k - \Big(\sum_{\sigma\in\Sigma} f_\sigma y_\sigma + \sum_{j\in J} c^F_{\phi(j)} z_j + \sum_{a\in\mathcal{A}^H}\sum_{k\in\mathcal{K}} h_k\Big(\sum_{i\in I(k)} w^i_a\Big) + \sum_{a\in\mathcal{A}^\Sigma}\sum_{i\in I}\sum_{j\in J} c^a_{\phi(j)} v_i x^i_{aj}\Big) \tag{2.1}$$

Subject to

$$\sum_{a\in\mathcal{A}^+_{(n,t)}}\sum_{j\in J_{\sigma_a}} x^i_{aj} + \sum_{a\in\mathcal{A}^+_{(n,t)}} w^i_a - \Big(\sum_{a\in\mathcal{A}^-_{(n,t)}}\sum_{j\in J_{\sigma_a}} x^i_{aj} + \sum_{a\in\mathcal{A}^-_{(n,t)}} w^i_a\Big)$$
$$= \begin{cases} \lambda_k, \text{if } (n,t) = \Big(o(k),\alpha(k)\Big) \\ -\lambda_k, \text{if } (n,t) = \Big(d(k),\beta(k)\Big) \\ 0, \text{otherwise} \end{cases} \forall (n,t)\in\mathcal{N}, \forall k\in\mathcal{K}, \forall i\in I(k) \tag{2.2}$$

$$\sum_{i\in I} v_i x^i_{aj} \le Q_{\phi(j)} z_j, \forall a\in\mathcal{A}^\Sigma, \forall j\in J_{\sigma_a} \tag{2.3}$$

$$\sum_{j\in J_\sigma} Q_{\phi(j)} z_j \le U_\sigma y_\sigma, \forall a\in\mathcal{A}^\Sigma \tag{2.4}$$

$$\sum_{j\in J_\sigma} z_j \le |J\sigma| y_\sigma, \forall \sigma\in\Sigma \tag{2.5}$$

$$\lambda_k = 1, \forall k\in\mathcal{K}^C \tag{2.6}$$

$$y_\sigma \in \{0,1\}, \forall \sigma\in\Sigma \tag{2.7}$$

$$z_j \in \{0,1\}, \forall j\in J \tag{2.8}$$

$$x^i_{aj} \in \{0,1\}, \forall k\in\mathcal{K}, \forall a\in\mathcal{A}^\Sigma, \forall i\in I(k), j\in J_{\sigma_a} | \phi(j)\in\Pi_k \tag{2.9}$$

$$w^i_a \in \{0,1\}, \forall a\in\mathcal{A}^H, \forall i\in I \tag{2.10}$$

$$\lambda_k \in \{0,1\}, \forall k\in\mathcal{K} \tag{2.11}$$

where $\mathcal{A}^+_{(n,t)} = \{a = \Big((n'',t''),(n',t')\Big) \in \mathcal{A} | n''=n,\ t''=t\}$ and $\mathcal{A}^-_{(n,t)} = \{a = \Big((n',t'),(n'',t'')\Big) \in \mathcal{A} | n''=n,\ t''=t\}$, for each $(n,t)\in\mathcal{N}$.

The objective of function 2.1 is to maximize total profit, calculated as the difference between revenues and costs. The model considers all costs related to

service selection, bin allocation, item storage at terminals, and transportation through the designed system. Constraints 2.2 ensure that each item is routed from its origin node to its destination node while respecting time constraints. Constraints 2.3 guarantee that the assignment of items to bins is feasible without exceeding their capacity. Constraints 2.4 define the maximum service capacity based on the total capacity of the bins operating within that service. Constraints 2.5 establish the relationship between the z and y variables, ensuring that bins assigned to a service can only be used if that service is activated. Constraints 2.6 require that all requests from regular customers are fully accepted and delivered. Finally, constraints 2.7 to 2.11 define the nature of the model variables.

2.1.2 Space-Time Network

Figure 2.2 illustrates an example on a physical network of five terminals, over a schedule length of seven periods. Each node is uniquely numbered, resulting in a total of 35 nodes in the network. This means that in the specific example showed in the figure a possible service 9 to 22 resulting in a service which has the following features:

- Departure terminal: 4.
- Departure time: 2.
- Arrival terminal: 2.
- Arrival time: 5.

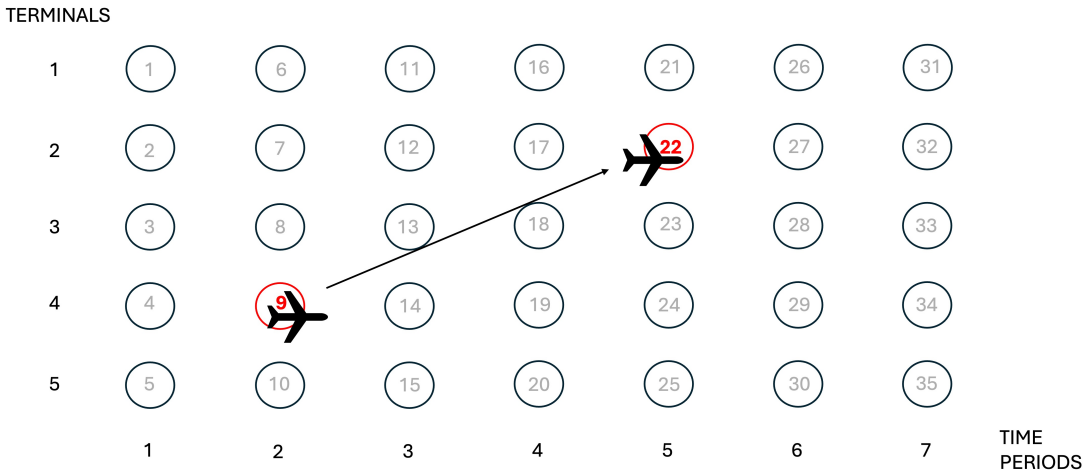


Figure 2.2: Space-Time Network

It is important to note that a commodity can remain at the same terminal for multiple time periods to facilitate consolidation when it is the optimal solution.

Based on the idea of consolidation on physical links at particular dispatching moments, the authors offer a novel formulation. According to this method, a collection of demands that can all travel simultaneously along the same physical link is represented by a consolidation connected to an arc or service. This idea makes it possible to allocate resources in a more organised and effective manner, maximising demand flow and reducing duplication and inefficiencies.

A key advantage of this formulation is its ability to predefine the set of consolidations associated with each physical link due to the relatively simple service structure. By generating these consolidations a priori, the model allows for the incorporation of more complex packing considerations and constraints without too much increasing computational complexity. This enhances the overall feasibility and flexibility of the model, making it particularly valuable for network optimization. Furthermore, a key advantage of this framework is that it does not require a time-space network formulation because consolidations are established directly on the actual network. This leads to a more compact and scalable solution by streamlining the modelling process and increasing computational efficiency. For larger issue instances, traditional time-space network formulations, which frequently depend on knapsack capacity constraints become computationally costly and challenging to scale. On the other hand, the suggested model provides a more efficient and performance-oriented alternative that is more appropriate for practical applications with limited dimensions.

The formulation greatly improves operational efficiency and decision-making in transportation and logistics networks by utilising this aggregation mechanism. It offers a reliable and computationally effective substitute for traditional models, guaranteeing improved physical link utilisation while preserving a high standard of service quality. This method shows exceptional efficacy in managing intricate logistical problems, marking a significant advancement in the field of network optimisation.

Chapter 3

Instances Generation

3.1 Demand Generation

Demand scenarios are systematically generated using Microsoft Excel to guarantee a diverse and comprehensive dataset for study. This phase entails the stochastic production of critical parameters that define each demand instance within the simulation framework. For each test scenario, the subsequent items are produced randomly.

Origin and destination nodes: each demand instance is linked to a randomly designated starting node and an ending node, signifying the source and destination points within the transportation network. This randomisation guarantees that the model is evaluated over diverse spatial distributions. Departure and Arrival Times: The temporal elements of each demand are assigned arbitrarily. The departure time indicates when demand activates inside the system, whereas the arrival time denotes the anticipated conclusion of the transportation operation. These factors introduce temporal variability, enabling the model to be assessed under various scheduling restrictions.

Request volume (k): the quantity or volume linked to each demand is another essential variable that is created randomly. This parameter affects capacity limitations and overall resource distribution inside the optimisation framework, rendering it crucial for performance evaluation. After generating a sufficiently large number of demand instances in Excel, these structured datasets, typically comprising several test cases, are exported and supplied as input to AIMMS, a sophisticated modelling environment for mathematical optimisation. The details of AIMMS and its function in the simulation process will be studied in the subsequent chapter.

AIMMS processes datasets supplied by Excel to produce structured .data files, which function as the standardised input format for executing the optimisation model. These files .data encompass many test scenarios, facilitating the methodical

assessment of the model's efficiency and correctness under diverse settings. This method converts unrefined demand data into organised computational scenarios, enabling thorough performance evaluation and model verification.

3.1.1 Consolidation: potential issue in generating commodities randomly

The M1M model assumes that commodities can be consolidated, meaning combined, to maximize the capacity of arcs (e.g., trucks or railway routes). When there are different commodities sharing the same origin, destination, departure time, and arrival time, the model treats these commodities as independent. As a result, consolidation may be handled incorrectly, leading to redundant utilization of arc capacity. Consequently, the capacity of an arc might be artificially filled due to duplication, preventing the consolidation of other genuinely distinct commodities. This leads to inefficient consolidation, increasing the number of transports required and, consequently, the costs. Additionally, having duplicated commodities with identical attributes contradicts physical logic, as two identical flows cannot exist simultaneously. It is equivalent to representing two shipments that are the same thing. This compromises the data consistency and the validity of the model. To prevent these issues, it is essential to implement validation checks and clean the dataset before using it in the M1M model.

The dataset represents a set of instances of commodities, each characterized by a set of attributes describing specific transport operations within a logistics network. Operationally, validation involves creating a unique key in the script based on origin, destination, departure time, and service duration, and identifying duplicate keys, meaning all commodities that share these four identical attributes. Departure times or service durations are then adjusted to generate different keys. Adopting a clean and accurate dataset is crucial to fully exploit the potential of the M1M model and to make informed and effective strategic decisions in managing the logistics network.

The original Macros included two scripts for managing non-cyclic cases and two for cyclic cases; moreover, there was the fifth script named "Generate random values" was designed to generate random values between 1 and 14 in column J of the worksheet. While functional, this script can be easily replaced with a straightforward formula directly in the worksheet. To achieve the same result without running the macro, you can use the following formula in column J: ROUND(RAND()*13+1,0).

All the others four scripts have been updated, and the most significant improvement across all of them is the introduction of a new data structure: the dictionary. This change has drastically reduced computational costs and improved efficiency. In fact, the two scripts that were able to modify the duplicate keys found had too high computational costs and could provide a solution with times greater than 5

minutes (for each sheet). So the old version was very slow and heavy, while the new one is faster and lighter. The following code is part of all 4 scripts, they were based on this structure:

```
For i = 2 To lastRow
currentKey = ws.Cells(i, "B").Value & ws.Cells(i, "F").Value & ws.
    Cells(i, "H").Value & ws.Cells(i, "I").Value

For j = 2 To lastRow
If i <> j Then
comparisonKey = ws.Cells(j, "B").Value & ws.Cells(j, "F").Value &  ws
    .Cells(j, "H").Value & ws.Cells(j, "I").Value

If currentKey = comparisonKey
   ws.Cells(i, "H").Calculate
   ws.Cells(i, "I").Calculate
```

The code used two loops, one inside the other, to compare every row with all other rows in the sheet. This process was repeated until no more duplicates were found. Because of the two loops, the code had to check every row multiple times. This resulted in a computational cost of $O(n^2)$, where "n" is the number of rows. For large datasets, this made the macro extremely inefficient. Moreover, when a duplicate was found, the macro updated certain cells, but it didn't handle these updates in the best way. It repeated calculations that could have been avoided.

3.1.2 Optimized VBScripts

The new code uses a dictionary to manage data more efficiently and performs extra checks on things like spaces and uppercase letters. A dictionary is like a table where each entry has two parts: a key and a value. The key is a unique identifier, and the value is the associated data. In this macro, the key is a combination of values from columns B, F, H, and I (after cleaning and standardizing them). The value is the row number where the key was first found. When processing a row, the macro checks if the key already exists in the dictionary:

- If the key exists, it means the row is a duplicate, and the code updates the relevant cells.
- If the key does not exist, it is added to the dictionary with the current row number as its value.

The dictionary allows the macro to find duplicates without checking every row against all the others. This reduces the computational cost from $O(n^2)$ to $O(n)$, which is a significant improvement for large datasets. Before adding information to the dictionary, the new code cleans up the data by:

- Removing extra spaces.
- Making everything uppercase so it's not case sensitive (even if in the 4 scripts this check is superfluous since the key only contains numbers and "-")

Instead of just comparing rows, the new code also checks for duplicates more carefully and updates certain cells with random values to refresh the data. As previously described, there are four scripts: two handle non-cyclic cases, and two handle cyclic cases. The code of the two pairs is almost identical, with the only difference being that in the cyclic case, the (key) cells on which the checks and potential modifications are performed differ from the cells in the non-cyclic case. Below, the two scripts for the non-cyclic case are shown, and then the small differences with the two scripts for the cyclic case will be highlighted. The following script is named CheckDuplicated-NonCyclic() and it aims to search and to find the number of total duplicated rows but also their position in the sheet, so their row number.

```
1  Sub CheckDuplicated_NonCyclic()
2      Dim ws As Worksheet
3      Dim lastRow As Long
4      Dim i As Long
5      Dim currentKey As String
6      Dim dict As Object
7      Dim duplicates As String
8      Dim count As Long
9      Set ws = ThisWorkbook.ActiveSheet
10
11     ' To find the last not empty row
12     lastRow = ws.Cells(ws.Rows.count, "B").End(xlUp).Row
13
14     ' To Initialize the dictionary
15     Set dict = CreateObject("Scripting.Dictionary")
16
17     ' output variables
18     duplicates = "Duplicated rows: " & vbCrLf
19     count = 0
20
21     ' Cycle on all lines and generate a key by concatenating the
       right values
22     For i = 2 To lastRow
23         currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-"
       & _
24                      UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
       & _
25                      UCase(Trim(CStr(ws.Cells(i, "H").Value))) & "-"
       & _
```

```
                    UCase(Trim(CStr(ws.Cells(i, "I").Value)))

        ' Existence check
        If dict.exists(currentKey) Then
            duplicates = duplicates & "Row " & i & " (duplicate of row
    " & dict(currentKey) & ")" & vbCrLf
            count = count + 1
    Else
             dict.Add currentKey, i
        End If
    Next i

  ' To show results
    If count > 0 Then
        MsgBox duplicates & vbCrLf & "Total number of duplicate rows:
    " & count, vbInformation
    Else
        MsgBox "There are no duplicate rows.", vbInformation
    End If
End Sub
```

In VBA, dictionaries can be accessed through the "Scripting.Dictionary" object, which is part of the Microsoft Scripting Runtime library. However, this library is not natively available on MacOS systems, as it relies on ActiveX components that are exclusive to Windows. As a result, when working on Mac, it is necessary to create a custom class to replicate the functionality of a dictionary, allowing cross-platform compatibility for VBA scripts. Will be shown after how to readapt this code to make it work on MacOS systems. It is important to emphasize that all four codes are based on the structure of the existing sheets, which contain the following attributes in the respective columns:

- Column B: starting node.
- Column F: destination node.
- Column H: departure time (non-cyclic case).
- Column I: service duration (non-cyclic case).
- Column L: departure time (cyclic case).
- Column M: service duration (cyclic case).

As shown in the image, this specific code performs checks on columns B-F-H-I, as it is specifically designed for the control of the non-cyclic case.

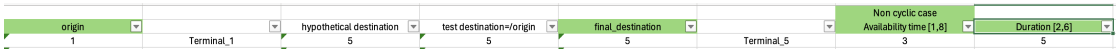


Figure 3.1: Excel structure: commodity generation in non-cyclic case

To ensure that the code was functioning properly, a simple verification system for the script was implemented. This system is based on the creation of four new columns (attributes): two to verify the correct functioning of the scripts related to the non-cyclic case, and two to verify the correct functioning of the two codes for the cyclic case. As shown in the figures for the non-cyclic case, a key is created in one cell by concatenating the values from columns B-F-H-I (the previously mentioned attributes). In the adjacent cell, a simple "COUNTIF" formula is implemented to count how many times that key appears in the entire column of keys. Here the concatenation formula and its result:

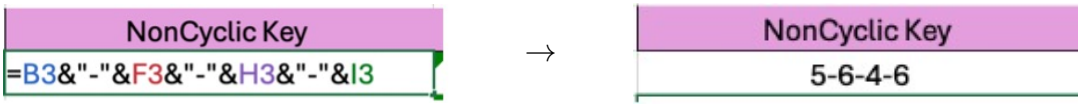


Figure 3.2: Concatenation Formula

In the following figure the “countif” formula and the overall output:

NonCyclic Key	NonCyclic Check
1-5-3-5	=CONTA.SE($Q:$Q;Q3)
2-5-5-4	1
6-7-5-6	1
6-7-7-2	1
2-6-7-3	1
1-4-6-4	1
7-2-3-4	1
2-1-8-4	1
5-4-5-3	1
4-5-5-4	2
6-7-4-3	1

Figure 3.3: CountIf Formula

Essentially, this checks whether each key generated for each row has a duplicate, i.e., whether it is repeated. If so, the value "2" will appear in the "check" column. By filtering all "2" values, it is immediately clear how many and which rows are duplicated.

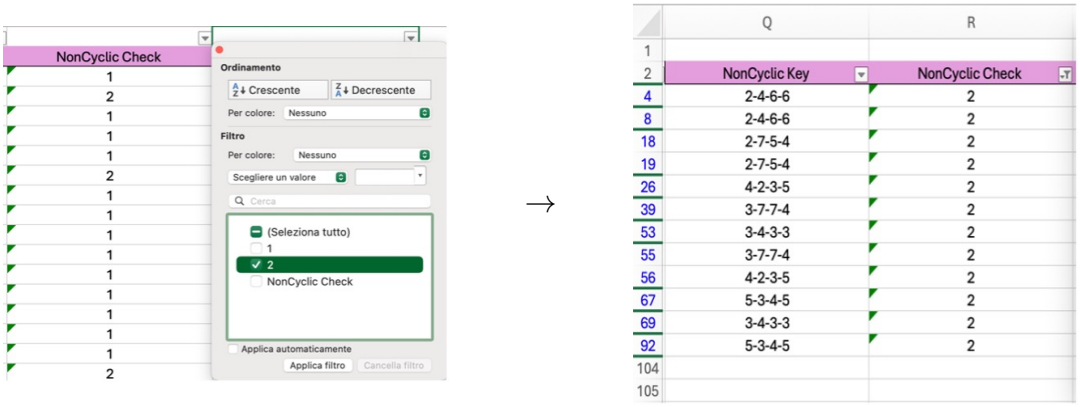


Figure 3.4: Filtering process

In this way, by running the script, it is possible to verify whether the script's output matches the easily implemented cross-check in the spreadsheet.

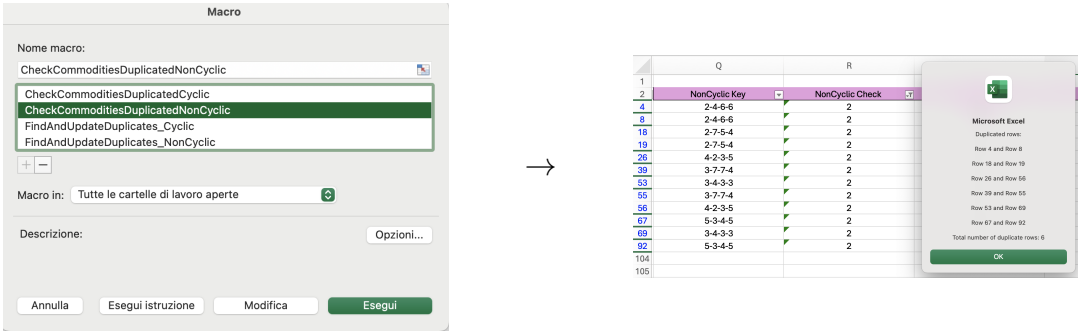


Figure 3.5: Macro Check Duplicated Commodities: non-cyclic

It's evident from last 2 figures that the script's output matches the cross-check in the spreadsheet. Once the duplicate keys (i.e., the commodities) have been identified, it is necessary to run the second script, "FindAndUpdateDuplicatesNonCyclic". The following piece of script is a part of the full algorithm which is visible in full in the appendix section.

```
1
2   ' Check if key exists
3       If dict.exists(currentKey) Then
```

```
4            ' if exists, updates values of availabilty time and
    duration (coloumn H and I)
5            ws.Cells(i, "H").Value = WorksheetFunction.Round(Rnd() *
    7 + 1, 0)
6            ws.Cells(i, "I").Value = WorksheetFunction.Round(2 + Rnd
    () * 4, 0)
```

The algorithm is essentially similar in the use of the dictionary as a data structure. However, it does not stop at identifying duplicates but also modifies the values in columns H and I by generating random values, as shown in the code above.

Very rarely, during the regeneration of the two key attributes, it might happen that the newly generated keys become duplicates of already existing keys. For this reason, it is highly recommended to rerun the previous script, "CheckDuplicatedNonCyclic", after executing this one to ensure that the output confirms there are no duplicate rows remaining. If duplicates are still found, simply rerun the "FindAndUpdateDuplicatesNonCyclic" script again, and the issue will be resolved. Nonetheless, this remains a very rare occurrence.

The figure 3.6 shows what happens in the excel file.

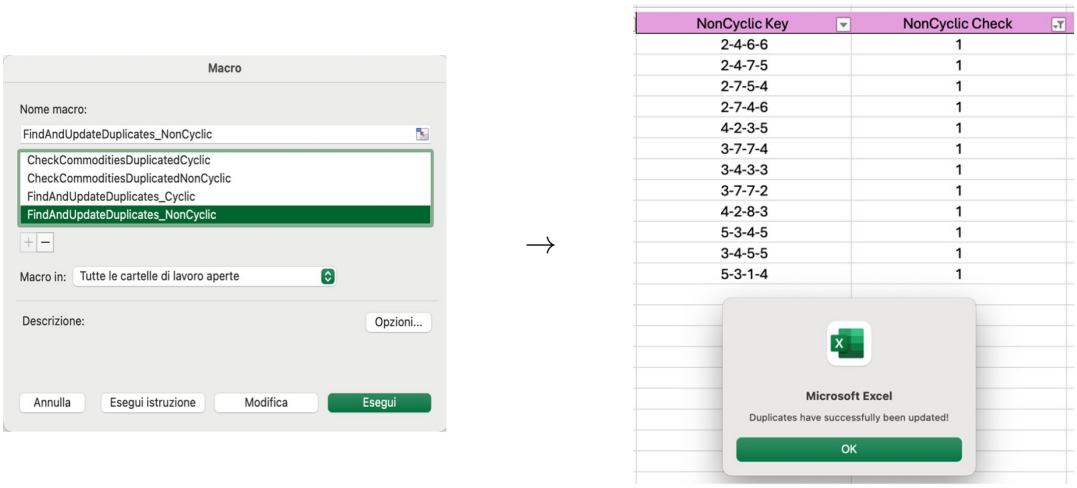


Figure 3.6: Macro Update Duplicated Commodities: non-cyclic

As seen in the figure, all keys are now different from each other. As a result, the "NonCyclicCheck" coloumn only contains "1". In conclusion this is the operational model of the script combined with the cross-verification process implemented. However, a user might want to apply the code in scenarios where time cyclicity is considered. For example, in cases where a service starts at time 12, has a duration of 5 periods, and finishes at time 3, assuming a period length of 14.

For this second case, where cyclicity exists, two additional scripts have been

designed. These are structurally identical to the two previously discussed scripts, with the only difference being the cells used to form the key in this cyclicity scenario.

origin	final_destination	Cyclic case Availability time [1,8]	Duration [1,10]
1	5	2	4

Figure 3.7: Cyclic case

Since, as mentioned earlier, the codes were designed based on the existing sheet structure, as shown in the previous image, the key in this case will be constructed using columns B-F-L-M. In the same way as seen before, two columns for verifying the correct functioning of the scripts have also been created for the cyclicity case.

As a result, the concatenation formula for the cyclic key is:

```
1
2 =B3&"-"&F3&"-"&L3%"-"&M3
```

The process of executing macros is the same but in this case the cyclic case macros are chosen that's illustrated in the following teo figures.

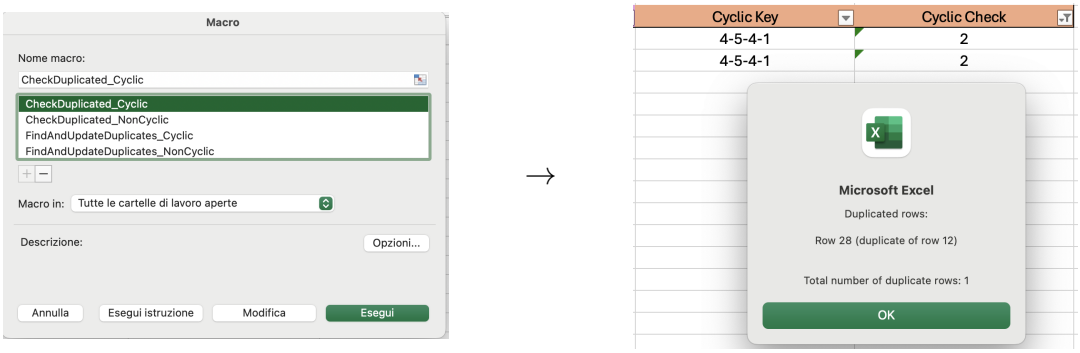


Figure 3.8: Macro Check Duplicated Commodities: cyclic

The code pair is basically the same of NonCyclic case. The only difference is the key which is made of the two attributes of cyclic case (coloumn L and M) in addition to the first two unchanged coloumns B and F representing starting and arrival node. The following code is only a part of CheckDuplicated-Cyclic to show the only change in the key. The entire code for both Check and Updates macros is attached in the appendix section.

```
1
2 For i = 2 To lastRow
```

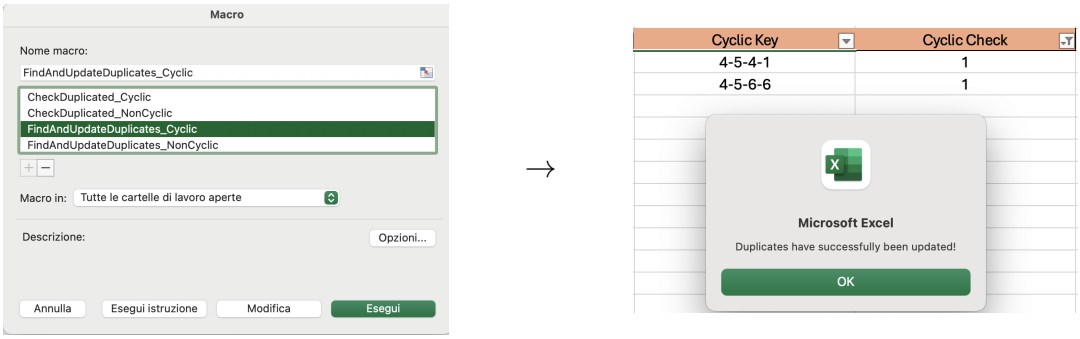


Figure 3.9: Macro Update Duplicated Commodities: cyclic

```
3         currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-"
    & _
4                      UCase(Trim(CStr(ws.Cells(i, "L").Value))) & "-"
    & _
5                      UCase(Trim(CStr(ws.Cells(i, "M").Value)))
6                      UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
    & _
```

Another small difference with respect the NonCyclic case is depicted in the next part of code. The row: ws.Cells(i, "M").Value = WorksheetFunction.Round(1+Rnd()*9,0) in which the random values are generated between 1 and 10, while in the other case they were been generated between 2 and 6. Anyway this difference is basically given by the constraints of the problem.

```
1
2         ws.Cells(i, "L").Value = WorksheetFunction.Round(Rnd() * 7 +
    1, 0)
3         ws.Cells(i, "M").Value = WorksheetFunction.Round(1 + Rnd() *
    9, 0)
```

Scripts in MacOS

As previously said In VBA, dictionaries are typically implemented using the Scripting.Dictionary object, part of the Microsoft Scripting Runtime library. This approach works seamlessly on Windows systems, but it poses a challenge on macOS because ActiveX components, such as the Scripting Runtime library, are not supported. Consequently, Scripting.Dictionary cannot be used natively on Mac. To address this limitation, a custom class module was created to replicate the functionality of a dictionary. This custom solution allows the script to handle key-value pairs, check for duplicate keys, and retrieve or update stored values ensuring

compatibility with macOS. The custom dictionary class, named clsDictionary, provides methods to:

- Add: insert a new key-value pair.
- Remove: delete a specific key and its associated value.
- Exists: check if a key already exists.
- Retrieve the value associated with a given key.

As can be seen in the following algorithm, the implementation of this class required the use of two internal collections:

- dictKeys: to store the list of keys.
- dictItems: to store the list of corresponding values.

```
Private dictKeys As Collection
Private dictItems As Collection
Private Sub Class_Initialize()
    Set dictKeys = New Collection
    Set dictItems = New Collection
End Sub
' to add an element
Public Sub Add(key As String, Item As Variant)
    Dim index As Long
    On Error Resume Next
    ' to find thekey
    index = dictKeys(key)
    If Err.Number = 0 Then
        Err.Raise 457, , "Chiave già esistente"
    End If
    On Error GoTo 0
   'if  there is not error, add the key and the element in the
    collection
    dictKeys.Add key, key
    dictItems.Add Item, key
End Sub
' to verify if key exists
Public Function Exists(key As String) As Boolean
    Dim k As Variant
    On Error Resume Next
    For Each k In dictKeys
        If k = key Then
            Exists = True
            On Error GoTo 0
```

```
            Exit Function
        End If
    Next k
    Exists = False
    On Error GoTo 0
End Function

' to get a value with key
Public Function Item(key As String) As Variant
    Item = dictItems(key)
End Function

' to remove a key and an element
Public Sub Remove(key As String)
    dictKeys.Remove key
    dictItems.Remove key
End Sub
```

With the clsDictionary class implemented, the original VBA scripts relied on Scripting.Dictionary needed modifications. Below are the key adjustments:

```
    Dim dict As Object

    Set dict = CreateObject("Scripting.Dictionary")
```

Moreover, on Windows, the syntax dict(key) = value could directly add or update a key-value pair. In the macOS-compatible script, this is replaced with:

```
If dict.Exists(currentKey) Then
        dict.Remove currentKey
End If
        dict.Add currentKey, True
```

Finally on Windows, dict(key) retrieves the value associated with a key. In the custom class, the equivalent is the following:

```
    dict.Item(currentKey)
```

Finally the full codes are in appendix section. In conclusion, by implementing a custom class module, the lack of scripting. Dictionary support on macOS was successfully addressed. These changes ensure that the VBA scripts maintain their functionality while remaining compatible across both Windows and Mac environments. This approach demonstrates the importance of adaptability when dealing with platform-specific limitations in VBA development.

Chapter 4

Modeling with AIMMS

The mathematical model was developed using the AIMMS software. AIMMS enabled the creation and translation of the mathematical formulation, as well as the generation of results from various simulations.

Optimising challenges are developed and solved using the modelling tool AIMMS. Like other tools such as AMPL and OPL, it has its own proprietary language that lets users specify goals, constraints, and variables. AIMMS uses specialised tools called solvers, rather than solving problems straight forwardly. A solver is a program designed to determine the best answer from a mathematical model entered into it. Various kinds of issues are solved with solvers including:

- LP: with a linear objective function and constraints.
- ILP : with integer or mixed-integer variables (MILP).
- Non-Linear Programming (NLP): with nonlinear constraints or objectives.
- Stochastic Programming: with uncertain parameters.

AIMMS offers a number of advanced modeling concepts not found in other languages, as well as a full graphical user interface both for developers and end-users. Aimms includes world-class solvers (and solver links) for linear, mixed-integer, and nonlinear programming such as baron, cplex, conopt, gurobi, knitro, path, snopt and xa, and can be readily extended to incorporate other advanced commercial solvers available on the market today. In addition, concepts as stochastic programming and robust optimization are available to include data uncertainty in your models. The multidimensional modeling language in Aimms offers a powerful index notation which enables you to capture the complexity of real-world problems in an intuitive manner. In addition, the language allows you to express very complex relationships in a compact manner without the need to worry about memory management or

sparse data storage considerations. The combined declarations and procedures using these multidimensional structures can be organized, edited and displayed using an advanced interactive model editor [11].

As a result, AIMMS lets users pick a solver for their resolution and streamlines the building of optimisation models. The usual procedure follows this:

- The model is written in AIMMS using its modeling language.
- The user chooses a solver (e.g., CPLEX).
- AIMMS formats the model such that the solver may understand it.
- The solver processes the data and returns the optimal solution.

This approach allows users to focus on problem formulation rather than the programming of the solving algorithm.

4.1 Identifiers

Obviously, this is not the place to explain all the tools and capabilities of AIMMS, as there are various detailed guides for this purpose. Instead, only the most commonly used methods will be presented to carry out this thesis project.

All data in AIMMS is kept in identifiers, which are essential components for managing and defining a model. In addition to storing data, identifiers are essential for functions and procedures because they allow the computation of new values from preexisting data. Through dynamic value updates and system-wide consistency, this structure enables AIMMS to manage mathematical models effectively.

AIMMS offers a variety of identifier types, each of which has a distinct purpose inside a model:

AIMMS provides several types of identifiers, each serving a specific function within a model:

- Set: define discrete domains over which other identifiers operate. Sets are essential for structuring models and organizing data efficiently.
- Parameters: store numerical values that can be fixed constants or dynamically derived from external inputs. Parameters allow the integration of real-world data into the model.
- Variables: represent the key decisions that need to be optimized. They can be continuous, integer, or binary, depending on the problem.

- Constraints: impose restrictions on the model, ensuring that the solution remains feasible and adheres to real-world limitations, such as capacity or budget constraints.

- Objective: specifies the criterion to be optimized, whether it be minimization (e.g., cost reduction) or maximization (e.g., profit or efficiency increase).

- Procedures and Functions: enable custom operations and advanced calculations. These elements manipulate identifiers to automate tasks, update data, and implement algorithms tailored to specific optimization needs.

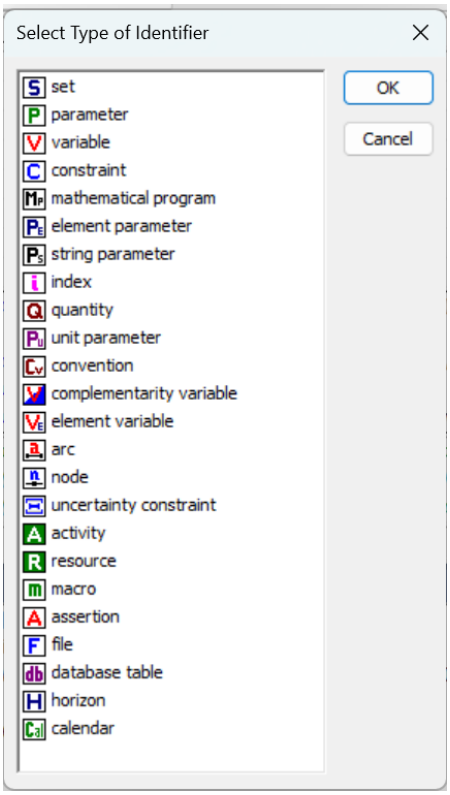


Figure 4.1: Identifiers on AIMMS

The figure 4.1 shows the identifiers window of the AIMMS software. It displays all the identifiers available within the software. The following figures will similarly be taken as screenshots from AIMMS, based on the project in question.

All the identifiers are accesibles in the model explorer. The Aimms Model Explorer provides you with a simple graphical model representation. The model tree lets you store information of different types, such as identifier declarations,

procedures, functions, and model sections. Each piece of information is stored as a separate node in the model tree, where each node has its own type-dependent icon.

As shown in the figure 4.2, the structuring nodes allow you to subdivide the information in your model into a logical framework of sections with clear and descriptive names. This is one of the major advantages of the Aimms model tree over a straightforward text model representation, as imposing such a logical subdivision makes it much easier to locate the relevant information when needed later on. This helps to reduce the maintenance cost of Aimms applications drastically. [11]

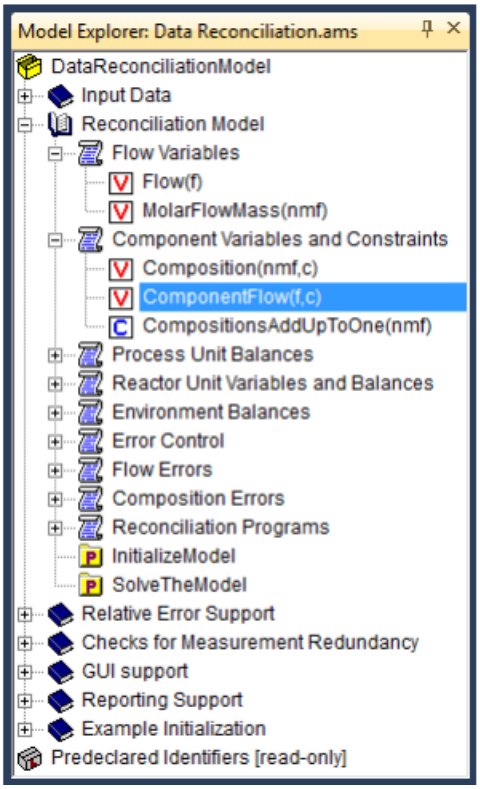


Figure 4.2: The AIMMS Model Explorer [11]

4.1.1 Sets, Parameters and Variables

The model's structural foundation is defined by sets, which are finite collections of elements that function as references for other identifiers, including parameters, variables, and constraints. They facilitate the generalisation of operations within the model by enabling the aggregation of similar elements, thereby establishing a hierarchical structure.

A set can be either simple, consisting of a single group of elements, or multidimensional, incorporating multiple sets to represent relationships between different categories. Specific principles can be applied to specific groups of elements with greater flexibility by defining subsets within a broader set. Additionally, sets may be either static, which maintains a consistent set of elements throughout the optimisation process, or dynamic, which changes in response to the model's conditions. As an example, only a few identifiers from the model presented in the chapter will be shown.

For privacy reasons, sensitive data from the AIMMS model cannot be fully disclosed as the publication has not yet been finalized and these data are subject to confidentiality agreements.

Figure 4.3 shows one of the sets of the model: the demand set. It includes all the requests received from customers to be handled by the system. They come from the random generation decribed in chapter 2.1.2.

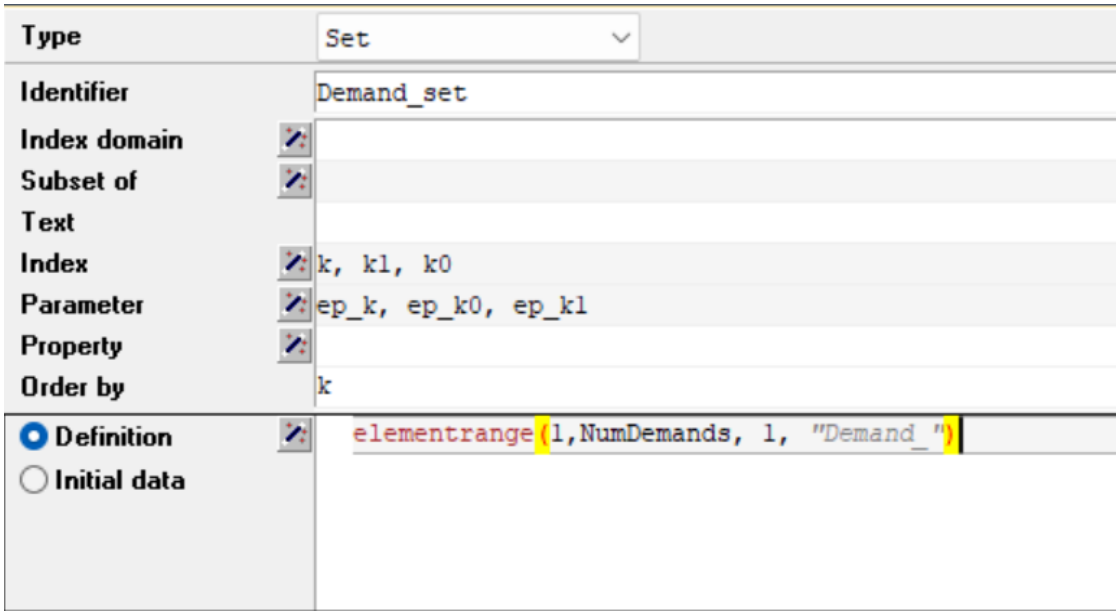


Figure 4.3: Set: Demand

Clear and scalable modelling necessitates the utilisation of sets. These are employed to index other identifiers, thereby eliminating the necessity to explicitly define each relationship between variables and enhancing the model's readability and maintainability. For example one identifier that could be indexed is the parameter. Parameters are numerical values that provide essential information within the model and are used to specify properties, constraints, or conditions that influence the decision-making process. Unlike decision variables, parameters are not modified by the solver but serve as input data for computational operations. Parameters can

be assigned explicitly as fixed values, computed based on formulas, or dependent on other variables in the model. They can also be dynamically updated throughout iterations, enabling simulations based on different circumstances. Thanks to the indexing feature It's feasible the definition of data matrices that vary based on the elements of the reference sets, structuring information in a tabular format and optimizing the computation process.

Variables are foundational components that represent the decisions to be made within an optimization model. Unlike parameters, which contain fixed numerical values, variables are the elements whose values are determined by the solver during the optimization process. Their role is crucial, as they define the possible courses of action that the model can take to accomplish the specified objective while respecting the imposed constraints. They are often subject to constraints that restrict their feasible range and ensure that the solution remains realistic and aligned with the problem's requirements.

The parameter "acceptance rate", shown in the following figure, serves as a simple example of the interconnection between variables, parameters, and sets. In reality, what is taken from the Demand set is its cardinality, meaning the number of elements (requests) it contains.

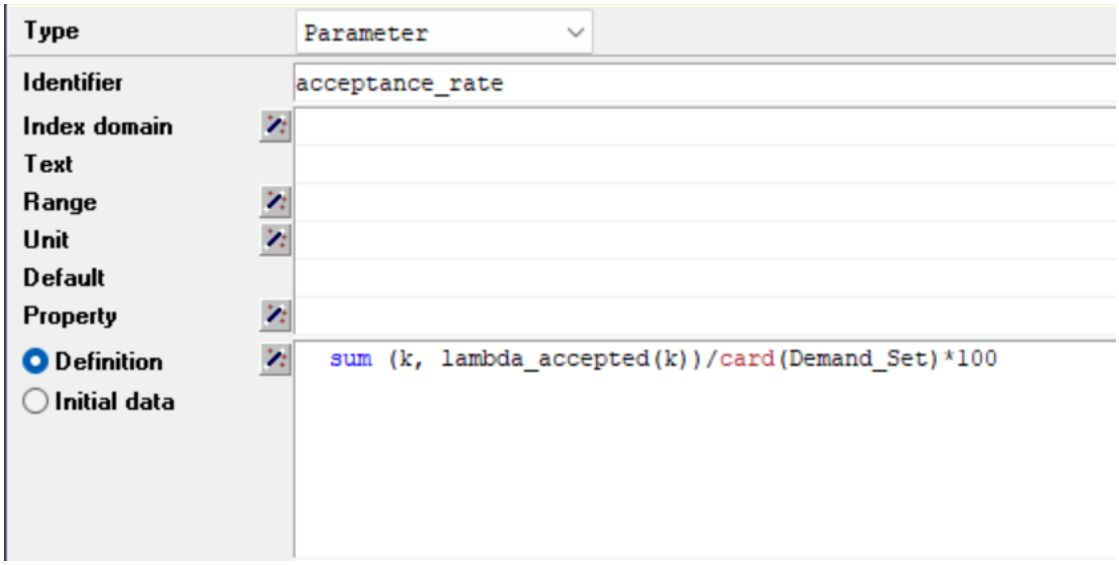


Figure 4.4: Parameter: Acceptance Rate

This parameter measures, as a percentage, how many requests have been satisfied out of the total received. It is a key metric that helps determine the appropriate sizing of the contractual request set based on the model's performance.

The variable lambdaAccepted is a binary variable, as shown in the figure 4.5, and is defined within the index domain k, which corresponds to the indexing of the

Demand set. The fact that it is binary means that it can only take values of 0 or 1.

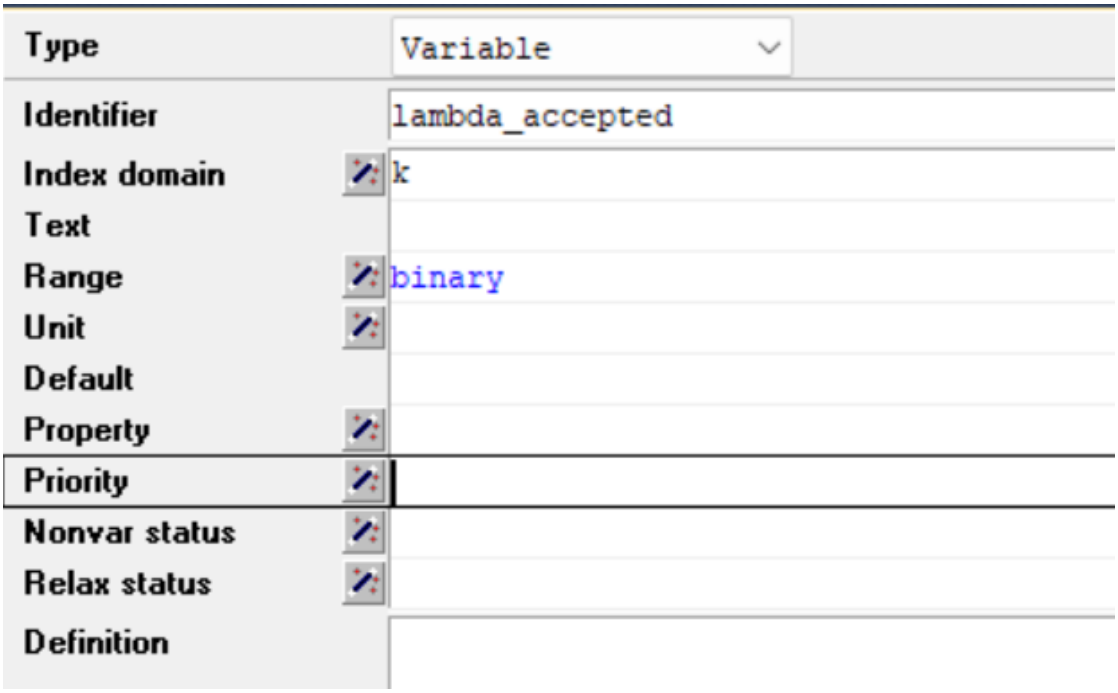


Figure 4.5: Variable: Lambda Accepted

Essentially, lambdaAccepted functions as a vector, where each request k is assigned either 0 or 1, depending on whether it has been accepted or not. The figure below specifically displays the vector containing only the positive values, corresponding to the accepted requests in that particular scenario.

Hence, when you right-click on an identifier and choose "Data," a specific data table or dialogue box with the values currently associated with that identifier is displayed. This feature is very helpful for understanding the structure of a model and confirming that variables, sets, and parameters contain the expected values. This acts as a strong tool for inspecting, entering, and changing data inside a model. Users can work directly with the model's data without changing the fundamental mathematical formulation thanks to this functionality, which offers an organised method of visualising, editing, and analysing the values linked to various identifiers.

When working with large-scale models, where programmatic data inspection might be difficult, this functionality is very helpful. Through the graphical interface, users may easily access and change values without having to write extra code to extract information.

There are also others identifier of parameter type which are more specific and suitable for some special situations.The Element Parameter, which accepts values from a predetermined set. An element parameter, as opposed to a typical numerical

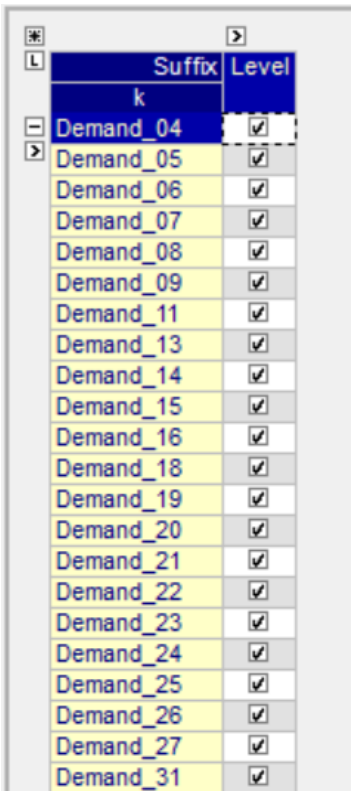


Figure 4.6: Lambda Accepted: Data

parameter, refers to a particular element within a set rather than having a numerical value. This feature, which enables more modularity and flexibility, is especially helpful when a model calls for dynamic reference of set elements. It can be used for instance, to monitor the selected or active element of a given set.

Instead of storing numerical data, the String Parameter is made to hold textual values. Because of this, it is very helpful for documentation, labelling, and dynamically specifying attributes connected to output. In real-world applications, a String Parameter can be used to hold category names, descriptions, or other metadata that improves the model's interpretability. Since the main purpose of AIMMS is numerical optimisation, the String Parameter only plays a supporting role in improving the model's documentation and clarity rather than having a direct impact on mathematical calculations.

A measurement unit to be connected to variables, constraints, or other parameters in the model is specified by the Unit Parameter. Unit consistency is supported by AIMMS by default, guaranteeing that all computations adhere to the proper dimensional analysis. By specifying a Unit Parameter, users may guarantee that model operations adhere to accurate unit conversions, lowering the possibility of

mistakes and enhancing computation accuracy.

4.1.2 Procedures and Math Program Inspector

Procedures in AIMMS are essential for managing the flow of execution, carrying out calculations, automating processes, and dynamically changing data inside a model. Procedures use an imperative programming paradigm, which means they carry out a series of operations in a predetermined order, in contrast to declarative elements like sets, parameters, and variables, which specify the mathematical structure of the optimisation model. They are therefore crucial for managing intricate reasoning that cannot be adequately conveyed by equations and constraints alone.

In AIMMS, a procedure is a collection of statements that specify the order in which certain calculations or operations must be carried out. These statements, which enable users to construct logic that manipulates data, regulates execution, or communicates with the solver, can include assignments, loops, conditional structures (if-else), and function calls. Modular and reusable coding structures are made possible by procedures' ability to call other procedures or functions.

Aimms supports several methods to initiate procedural model execution. More specifically, you can run procedures:

- from within another procedure of your model,
- from within the graphical user interface by pressing a button, or when changing a particular identifier value, or
- by selecting the Run procedure item from the right-mouse menu for any procedure selected in the Model Explorer.

The first two methods of running a procedure are applicable to both developers and end-users. Running a procedure from within the Model Explorer a useful method for testing the correct operation of a newly added or modified procedure [11].

Procedures also make it possible for AIMMS to use user-defined algorithms. Some models need extra logic that cannot be effectively represented by constraints alone, even while the solver manages the fundamental mathematical optimisation. In these situations, procedures can apply decomposition techniques, metaheuristic methodologies (such simulated annealing or genetic algorithms), or unique heuristics to more effectively handle large-scale or computationally complicated problems.

Figure 4.7 shows a simple procedure which aims to load multiple case files and for each of them it runs another prodcedure: Solve SND wiht profit.

A case file is a file that includes all of the information about sets, parameters, variables, and solution outcomes as well as a snapshot of the model's current state. With the use of this file, It's possible to save and then to reload a particular set of

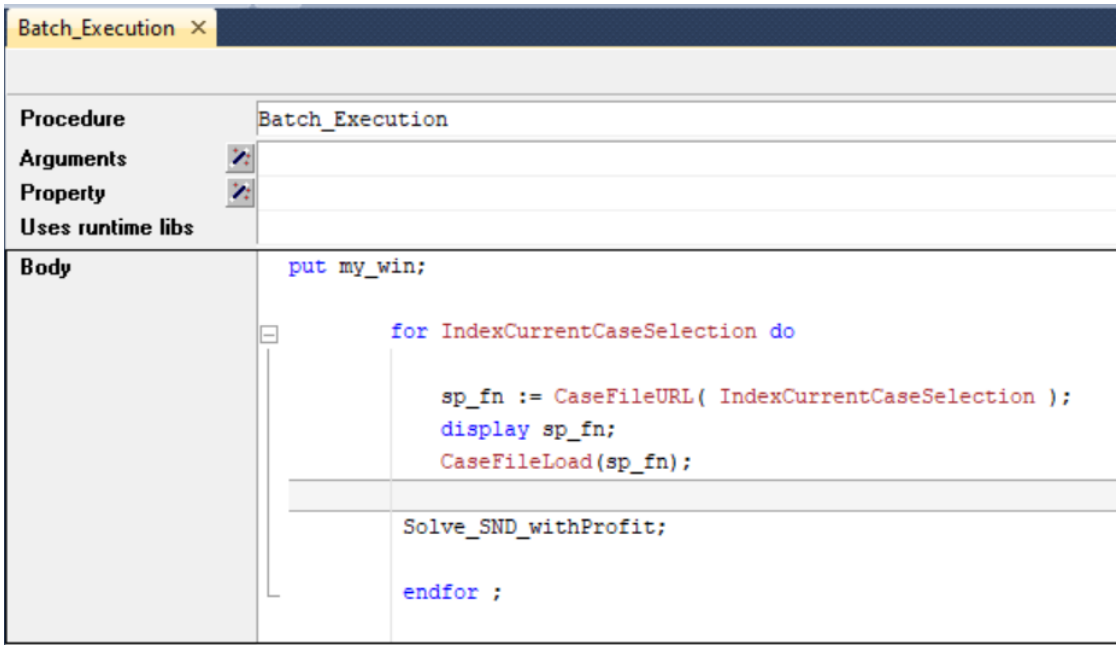


Figure 4.7: Procedure: Batch Creation

inputs and outputs without having to manually re-enter data or recalculate the entire model.

To load a case in AIMMS It's necessary to use the proper menu, as shown in the figure below.

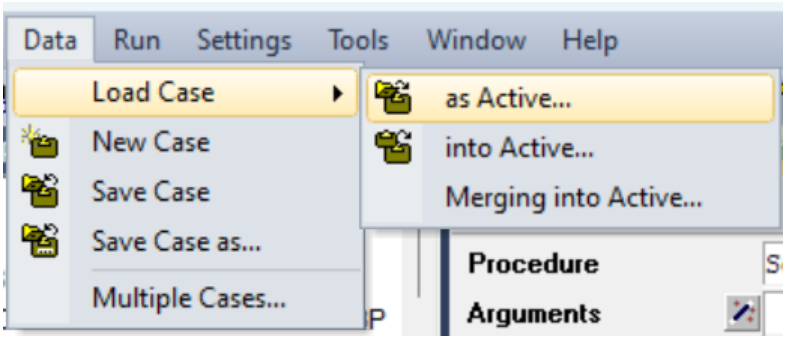


Figure 4.8: Load Case

Once the appropriate button in the drop-down menu is clicked, the desired case must be selected from the list as appears in the figure.

Batch execution which is visible in figure 4.7 is made possible by AIMMS's ability to choose numerous cases at once. This functionality is very helpful when working with large-scale models that need to automatically evaluate various situations.

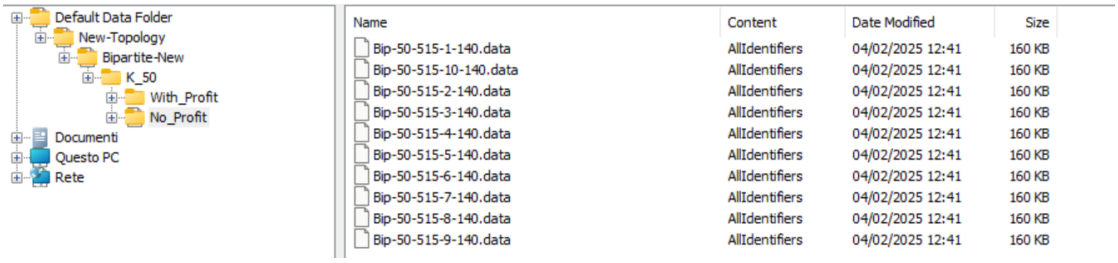


Figure 4.9: Case Files

Multiple case selection allows AIMMS to analyse them in a sequential manner without user intervention, greatly increasing workflow efficiency for scenario analysis and optimisation. This enables users to compare various input sets and see how they affect the model's results. Furthermore, batch execution makes it easier to assess results and improve the model based on methodical observations by guaranteeing consistency in performing preset scenarios under identical computational parameters.

Another important tool usable in AIMMS is the Math Program Inspector. It is used for examining, troubleshooting, and improving mathematical models both before and after they have been solved. It gives users a thorough understanding of the composition and properties of an optimisation model, enabling them to identify possible problems, evaluate computational performance, and optimise formulations for increased accuracy and efficiency.

Model structure analysis is one of its main features, allowing users to look at important elements including variables, constraints, goal functions, and parameters. Redundancies, inconsistencies, or missing components that could impair the solver's performance can be found by visualising the model's structure. The tool also makes constraint and variable examination easier, enabling users to examine the values of variables and evaluate both active and inactive constraints. This guarantees that variables behave as predicted during the optimisation process and aids in identifying binding constraints that restrict the possible solution space.

The Math Program Inspector must be chosen from the specific "Tool" menu in order to be used as seen in figure 4.10. One of the most often utilised features in this project is "Subtract Causing Infeasibility."

For identifying and fixing infeasibilities in an optimisation model, the "Subtracture Causing Infeasibility" tool is quite helpful. A mathematical model's inability to identify a workable solution frequently indicates that the limitations placed on the problem are too onerous, leaving no workable solution within the specified feasible range. This feature aids in determining whether constraints render the model impracticable.

Upon activation, the Math Program Inspector examines the constraint list and

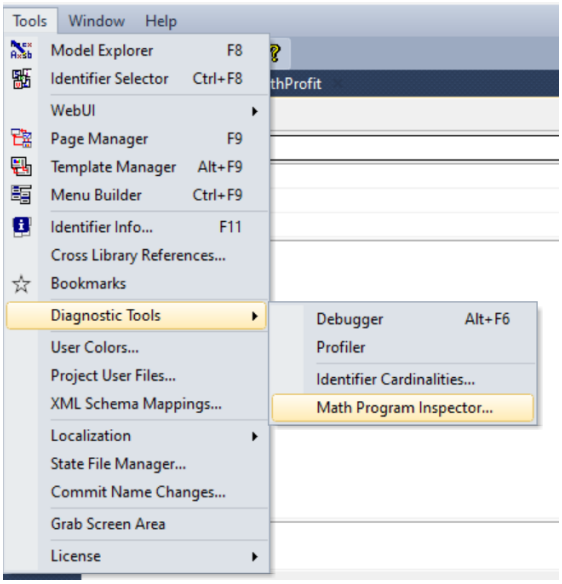


Figure 4.10: Math Program Inspector

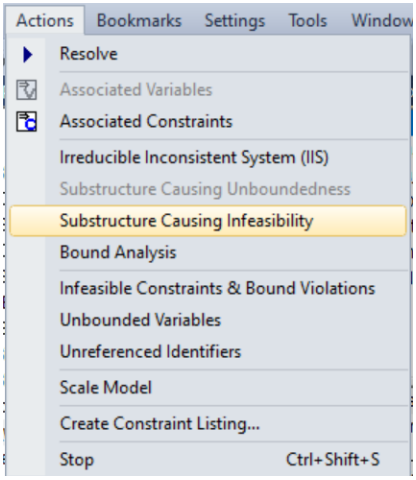


Figure 4.11: Substructure Causing Infeasibility

methodically eliminates any that make it impractical. Figure 4.11 shows how to select the "Substracture Causing Infeasibility" from the right actions menu.

AIMMS's Math Program Inspector and procedures offer strong tools for mathematical model analysis, debugging, and optimisation. Procedures allow users to apply customised operations that improve model flexibility and automation, while the Math Program Inspector helps to uncover infeasibilities, refine restrictions, and improve solver performance. When combined, these features guarantee increased

effectiveness, precision, and command over challenging optimisation issues, which eventually leads to more robust and reliable decision-making procedures.

Chapter 5

Computational Experiments: Discussion and Conclusion

Although they are minor in comparison to the project's total scope, the operational components of the work and the contribution will be discussed in this part. The purpose of this section is to give readers a clear understanding of the strategy used and the knowledge acquired by carrying out specific operations.

As stated in the introduction, the project forming the foundation of this thesis aims to assess the advantages and performance of combining two different optimisation problems: the BPP and the SND. In order to evaluate the impact and efficacy of this integration in a methodical and quantitative way, the research focuses on examining the solution provided by a mathematical model that integrates these two logistical and economic difficulties.

By contrasting the results of the integrated method with those of tackling the two problems independently, the effectiveness of the suggested model is evaluated. The potential benefits of tackling these issues inside a single framework as opposed to considering them as separate optimisation problems can be better understood thanks to the comparative analysis. In the end, the results help assess if this combined strategy can result in better decision-making procedures in pertinent operational and industrial situations.

5.1 Scope of experiments

The work and experiments carried out in this thesis serve solely as a functional contribution to the broader objective of the overall research project. Their purpose is to support and complement the larger study by providing specific insights and analyses that contribute to the final goal.

In particular, it will present the findings from multiple computational experiments that were conducted in order to investigate and examine the model's behaviour in various scenarios. The purpose of the numerical experiments is to first evaluate the integrated model of SND and BPP, analyze the results generated by running this model, and then compare these results with those obtained from the model that does not consider the integration of BPP with SND. As will be shown later, integration provides benefits in terms of optimization, leading to cost reductions and, consequently, increased profits.

Before comparing the two models, a profit estimation analysis was conducted. Specifically, the model was executed in two different ways. In the first run, the objective was to minimize costs, using the obtained result as a proxy for potential profits. Subsequently, in the second execution, the model was run to maximize profits, defined as revenues minus costs. At this stage, various variables and parameters were analyzed with an emphasis on two crucial parameters that were particularly relevant to the analysis.

The first parameter analysed is the acceptance rate, which is crucial for determining the appropriate sizing of the set of contractual requests. Additionally, it provides insight into whether the average obtained value is realistically "good." This means assessing whether the model, as the number of received requests (k) varies over a given time horizon, is capable of delivering reasonable results. If so, this would indicate a potential adaptability of the model to real-world applications. In essence, the acceptance rate serves as an initial indicator of the model's performance, which must then be evaluated alongside its computational cost to fully assess its practical feasibility and efficiency.

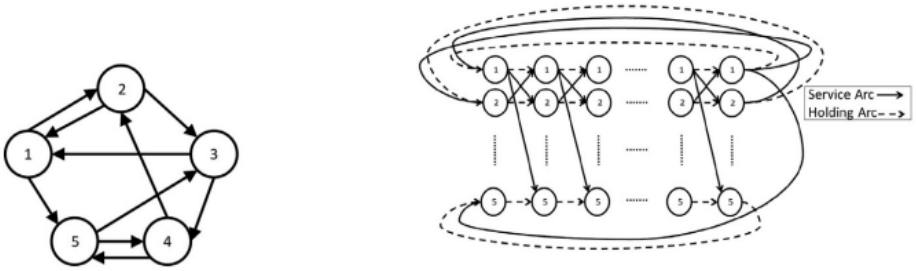
The second step, as said, involved obtaining a preliminary, albeit rudimentary, estimate of the profits associated with each request. To achieve this, it was first necessary to compute the total costs by summing the unit costs of each request. Then, potential profits were estimated under the assumption that they follow the costs according to a uniform distribution. As will be shown, significant profits were obtained in all the tested cases, demonstrating the good performance of the model and confirming its effectiveness.

5.2 Experimental plan

Experiments were conducted on three different network topologies, each subjected to a set of tests on specific instances, 10 for each demand class designed to represent real-world scenarios. These instances included randomly generated services, allowing for the simulation of variable operating conditions and testing the adaptability of the model under different constraints.

The first network topology was taken from Lanza et al. (2021) and is illustrated

in the corresponding figure. This topology is itself inspired by Crainic et al. (2014b). As shown in the figure 5.1, the network consists of five nodes and ten arcs. For this topology, three demand classes were considered, specifically with k values set at 15, 20, and 25. For simplicity, from now on, the instances generated for this topology will be referred to as "Lanza."



(a) Physical service network

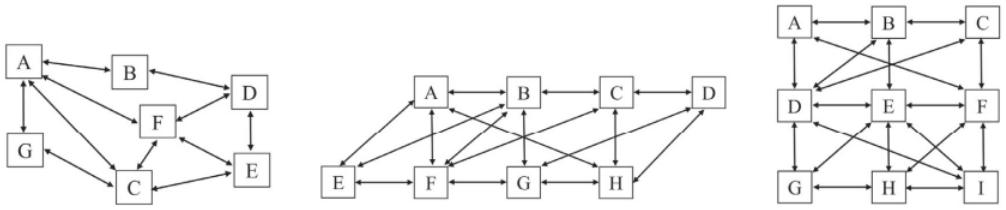
(b) Space-time service network

Figure 5.1: Physical and space-time service networks [12]

The space-time configuration shown in the figure 5.1(b) represents a network that is set up to control both the potential for strategic waiting and the dynamism of flows. The nodes' arrangement demonstrates a well-organised chronological progression, but the existence of holding arcs, particularly those with lengthy, curved connections, indicates a flexible resource management strategy in which availability and route optimisation are the primary constraints in addition to time.

The second and third network topologies, as proposed by Taherkhano et al., correspond to bipartite and grid structures. Additionally, preliminary experiments were conducted on a fourth topology, the hypercorridor network, which is also depicted in the figure. However, the initial results obtained from this network suggested that further investigation might be required before drawing meaningful conclusions. Consequently, its analysis was temporarily set aside in favor of focusing on the other three networks. The second image of 5.2 shows a network with eight nodes and sixteen edges called Bipartite (8,16). With connections mostly between pieces from different sets, the structure emphasises two separate groups of nodes. In assignment models, such as matching problems in optimisation or representations of the interactions between two different groups (e.g., tasks and available resources), this kind of network is common. The presence of cross connections reveals that there is a more flexible distribution of connections rather than a strict one-to-one correlation between the groups, while the orientation of the edges suggests a movement of information or resources between the two sets. A clear structure that is helpful for routing and allocation issues is highlighted by the lack of internal cycles inside the separate sets.

The third image depicts a grid-structured network, so called Grid (9,19), with nine nodes and nineteen edges. This arrangement is typical of spatial models in which every node is linked to its immediate neighbours, resulting in a system that is very regular and structured. Multiple alternate routes between nodes suggest great redundancy and good fault tolerance, which makes this arrangement appropriate for communication networks or transportation models. The existence of diagonal connections in comparison to the basic orthogonal layout suggests greater flexibility in the transfer of resources or the dissemination of information, and the directed edges display connection directions that imply a flow dynamic. Because it reduces the average distance, the grid layout is especially helpful for routing issues in physical networks, such energy distribution or traffic on the roads. For both the bipartite and grid networks, three different sets of shipper demands were considered, each with increasing sizes: 50, 75, and 100. This allowed for an evaluation of how the model behaves under varying levels of demand intensity and provided insights into its scalability and efficiency.



(a) Hyper-corridor (7,11) (b) Bipartite (8,16) (c) Grid (9,19)

Figure 5.2: The physical network topologies [13]

The instances were generated based on the three different network topologies to evaluate their impact on the acceptance rate and total costs. Additionally, the services include all possible routes within each network.

For privacy reasons, the specific characteristics of the services and their associated costs will not be disclosed. For each demand class, 10 instances were generated, except for the topology taken from Lanza et al., where the number of instances per demand set was increased to 20. The computational experiments were performed on a laptop equipped with an Intel i3-2310M CPU @ 2.10 GHz and 6GB of RAM. The mathematical model was solved using the CPLEX 22.1 solver, and all instances were solved to optimality.

In the problem formulation, demand is divided into two types: contractual and non-contractual. Contractual demand must always be satisfied due to imposed constraints, while non-contractual demand has no such requirement and can be accepted or rejected based on optimization criteria. The first objective is to compute all demands that can be satisfied and to get the accepatance rate of each scenario.

To ensure the problem remains feasible, the contract set is initially left unrestricted and empty. If, instead, 70% of the demand set were randomly assigned to the contract set, the problem would likely become infeasible. This is because all contractual demands must be fulfilled, but since we cannot predict in advance whether a particular demand (e.g., demand 2) will be satisfied—depending on factors such as existing services between nodes, bin capacities, and timing constraints, some demands might remain unmet, making the problem unsolvable.

This infeasibility is visually represented in the figure 5.3, where the constraints shown in blue are not satisfied. This issue has been identified using the Math Program Inspector, which allows for a detailed analysis of constraint violations.

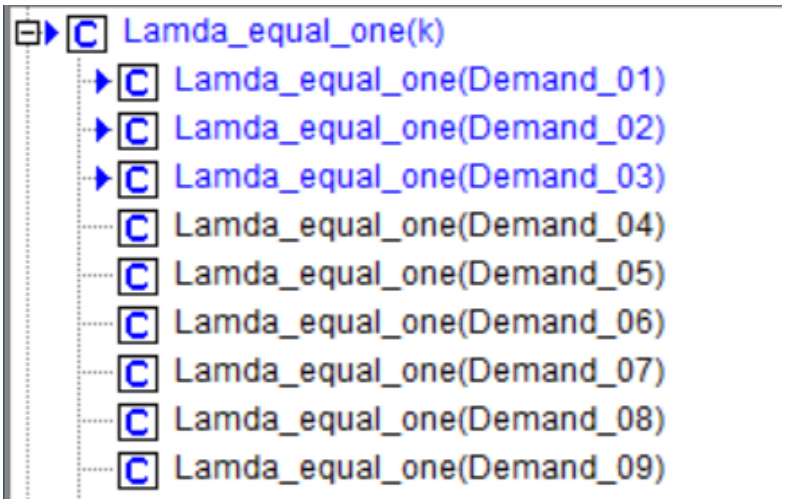


Figure 5.3: Constraints causing infeasibility

By leaving the contract set empty at this stage, an optimal solution can be obtained, represented by the lambda vector. This vector contains the subset of demands selected from the total demand set. Additionally, this phase enables the calculation of total costs, as profits are set to zero, making the objective function negative.

In the second phase, the lambda vector obtained from the first phase is fixed within the original problem. This means that the selection of demands is now assumed to be predetermined and cannot be altered. Essentially, the contract set is now explicitly defined as the exact set of demands that were satisfied in the first phase. Moreover, the total costs computed in the first phase are used as input to estimate the possible profits associated with each demand. This is done through a specific formula which will be shown in the following section, that links the costs to potential revenue generation. By incorporating these profit estimates, the model now captures a more realistic economic perspective of the problem.

At this stage, the true value of the objective function is obtained. Since both the

revenues (derived from the accepted demands) and the total costs are now known, the model can accurately evaluate its financial performance. Additionally, by calculating the ratio of contractual demands to total demands, the model provides a key insight: this ratio represents the percentage of contract demand that can be feasibly sustained under the specific simulation conditions. This percentage serves as an indicator of the model's capacity to integrate contractual commitments while maintaining feasibility and profitability. This approach is essential for estimating the appropriate sizing of the contract set. Specifically, it helps determine whether the percentage of total demand that can be satisfied is sufficient to ensure not only a reliable service level but, more importantly, the profitability of the service based on the fixed costs.

5.3 Output of Integrated Model SND + BPP

First, all available cases were executed using the integrated model. As mentioned, this approach involved an initial cost estimation by minimizing them while setting profits to zero. Subsequently, these estimated costs were used to determine the profits. In this way, during the second execution, the profit for each case was obtained, which corresponded to the objective function value and was defined as revenue minus costs. The instances were executed with a time limit of 3600 seconds. While the actual solving time for each instance was recorded and noted, an in-depth analysis of computational costs was not within the scope of this thesis. The only requirement was to ensure that each instance reached optimality within the given time limit. As previously mentioned, the primary objective of the first execution was to determine precisely which demands were satisfied for each case, measuring both the acceptance rate for each specific scenario and the average acceptance rate for each demand class.

The objective function in this phase always assumed a negative value. This is due to the structure of the function itself: since in this first execution revenues were set to zero, and the problem is formulated as a minimization problem, the solver naturally returned the lowest possible negative objective value. In figure 5.4 is shown a standard output of a specific instance of this first computation. In particular the instance is taken by grid topology set for k=50.

The complete output shows the time taken, the various types of costs identified in the model, and, most importantly, the total costs used to estimate profits. Additionally, the lambda accepted values indicate exactly which requests were accepted by the model. The results provide useful insights for optimizing transport networks, highlighting the benefits of an integrated approach in solving the Service Network Design problem with packing constraints. By gathering data for grid topology from all demand classes (50,75,100) the following table has been got.

GRID_50_Instance1.txt

```
lambda_accepted(k) := data
{ Demand_01 : 1,  Demand_02 : 1,  Demand_03 : 1,  Demand_05 : 1,  Demand_06 : 1,  Demand_10 : 1,  Demand_11 : 1,
  Demand_12 : 1,  Demand_13 : 1,  Demand_14 : 1,  Demand_18 : 1,  Demand_19 : 1,  Demand_20 : 1,  Demand_23 : 1,
  Demand_24 : 1,  Demand_26 : 1,  Demand_27 : 1,  Demand_29 : 1,  Demand_30 : 1,  Demand_31 : 1,  Demand_32 : 1,
  Demand_33 : 1,  Demand_34 : 1,  Demand_35 : 1,  Demand_36 : 1,  Demand_39 : 1,  Demand_40 : 1,  Demand_41 : 1,
  Demand_43 : 1,  Demand_45 : 1,  Demand_47 : 1,  Demand_49 : 1 } ;

profit(k) := data { } ;

contract_rate := 0 ;

acceptance_rate := 64 ;

totalProfit := 0 ;

totalcost := 69855 ;

SND_withProfit.Objective := -69855 ;

SND_withProfit.SolutionTime := 88.453 [s] ;

SND_withProfit.ProgramStatus := 'Optimal' ;

serviceCost := 6670 ;

bincost := 18995 ;

holdcost := 36080 ;

TransportationCost := 8110 ;

SND_withProfit.bestbound := -69855 ;

SND_withProfit.Incumbent := -69855 ;

Gap := 1000000 ;
```

Figure 5.4: Output of first computation

Demand (K)	Average Acceptance Rate (%)
50	69.0
75	68.7
100	69.0

Table 5.1: Grid Topology: acceptance rate

The acceptance rate for the grid topology shown in the table 5.1 is basically the same for the three demand classes. The low standard deviation suggests that there are no significant irregular fluctuations in the acceptance rates.

For each tested topology, the average acceptance rates are shown in the following table. The Lanza cases have a higher average acceptance rate than the other two. This is because the Lanza instances consist of 20 cases for each demand class, while the other two topologies have only 10 cases per class. Additionally, the Lanza topology includes three demand classes with k=15, k=20 and k=25 whereas the other two topologies have demand classes with k=50, k=75 and k=100.

By multiplying the objective function by -1, we obtained the total cost value. This total cost, along with the set of satisfied demands, was then used in the second execution of the instances to derive the profit value for each individual demand. Specifically, the formula used to compute individual profits is as follows.

Demand (K)	Average Acceptance Rate (%)
Grid	69 %
Bipartite	71 %
Lanza	88 %

Table 5.2: Average Acceptance Rate

```
profit(k):=2*ceil((totalcost*(uniform(0.8,1.4)))/card(NumDemands)
```

The current approach uses a random factor with uniform distribution in the interval [0.8,1.4] to calculate profits based on total costs. Simply said, this approach might not be sufficient to capture the reality of the issue. A uniform distribution means that all values inside the interval are equal, unless there is a preference for a particular profit level. However, in the context of an economic optimisation problem, it is reasonable to assume that some profit intervals are more likely than others. For example, in many real-world scenarios, the profits tend to follow a less balanced distribution, with a higher concentration around a central value and a lower likelihood of extremely low or extremely high equilibria.

Analysing past data from comparable situations and creating a distribution based on actual observations is a more accurate option. A function that takes into account pertinent factors like the distance related to the demand, the necessary volume, or other operational parameters could be derived in place of choosing a multiplicative factor at random. Analysis of historical data may show, for instance, that profits often correlate with the value of the transported items or the length of the route. In this instance, a coefficient that evaluates each demand's contribution in light of these factors might be included in the calculation.This method improves the model's adherence to reality and lowers the possibility of producing irrational profits by allowing for the creation of a profit distribution that is not random but rather represents the patterns shown in historical data.However, the results obtained from the second calculation step are examined. This step utilises the demands selected in the first phase along with the total costs. The outcome of this second computation reveals that the average value of the objective function is so positive, indicating that gains are being generated.

Figure 5.5 is the typical output of a Lanza instance with k=25 related to the second run. It can be observed that in this second execution, the profits are no longer zero, and the objective function is no longer negative. This is due to the previous estimation of total costs, which then made it possible to derive the profits in a somewhat rudimentary way. In addition to the individual mini-reports generated for each tested instance, as shown in the figure 5.5, the procedure also created a summary output that compactly gathered all relevant information for

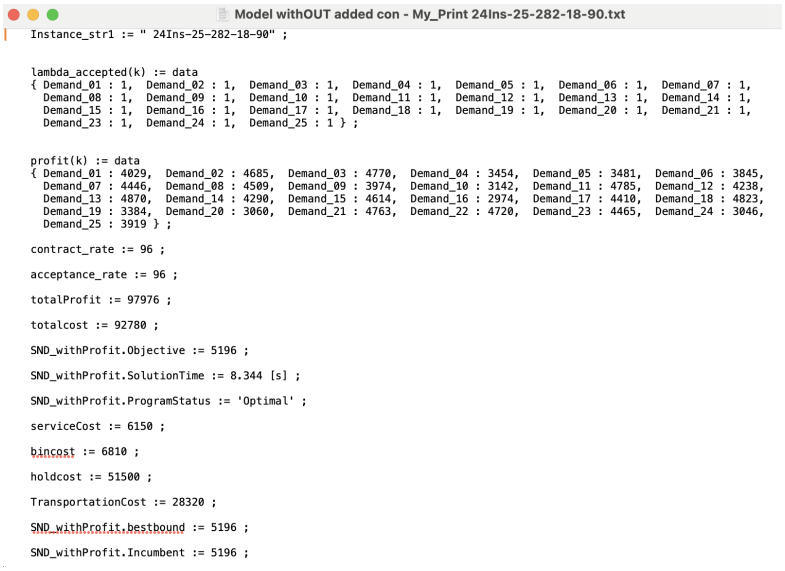
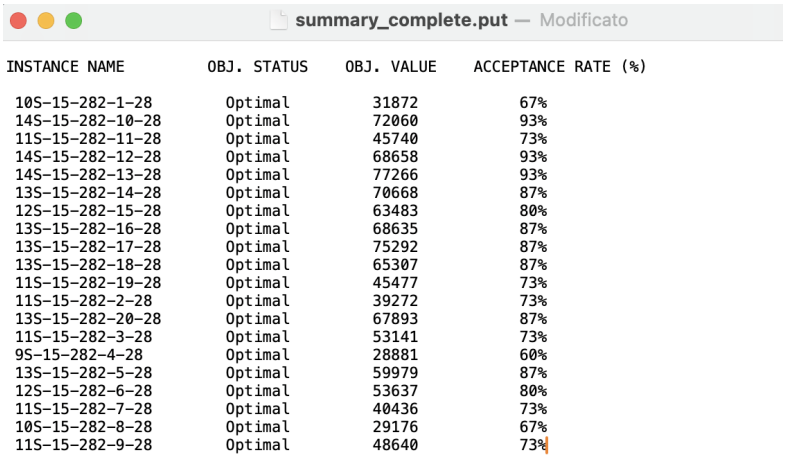


Figure 5.5: Output of second computation

each demand class of each topology. Consequently, the summarized results are visible as shown in the figure 5.6.


summary_complete.put — Modificato

INSTANCE NAME	OBJ. STATUS	OBJ. VALUE	ACCEPTANCE RATE (%)
10S-15-282-1-28	Optimal	31872	67%
14S-15-282-10-28	Optimal	72060	93%
11S-15-282-11-28	Optimal	45740	73%
14S-15-282-12-28	Optimal	68658	93%
14S-15-282-13-28	Optimal	77266	93%
13S-15-282-14-28	Optimal	70668	87%
12S-15-282-15-28	Optimal	63483	80%
13S-15-282-16-28	Optimal	68635	87%
13S-15-282-17-28	Optimal	75292	87%
13S-15-282-18-28	Optimal	65307	87%
11S-15-282-19-28	Optimal	45477	73%
11S-15-282-2-28	Optimal	39272	73%
13S-15-282-20-28	Optimal	67893	87%
11S-15-282-3-28	Optimal	53141	73%
9S-15-282-4-28	Optimal	28881	60%
13S-15-282-5-28	Optimal	59979	87%
12S-15-282-6-28	Optimal	53637	80%
11S-15-282-7-28	Optimal	40436	73%
10S-15-282-8-28	Optimal	29176	67%
11S-15-282-9-28	Optimal	48640	73%

Figure 5.6: Output of integrated model

From this output, it is possible to quickly calculate the average acceptance rate, as done previously, and also obtain information about the optimality of the solution and the value of the objective function. The way an instance's name is structured is not random but follows a specific logic.

- The first number represents the number of accepted requests, meaning those that were satisfied.
- In the first part, "Grid," "Bip," or "Ins" indicate the type of topology being referenced.
- The second number represents the demand class which the instance belongs to.
- The third one is the number of bins.
- Then, a progressive number identifies the instance (first, second, etc.).
- Finally, the last number indicates the total demand for items of that specific instance.

In Appendix B, all the results will be presented in tabular form, as shown in the figure. To enhance visual impact and clarity, and to simplify everything, the instance names will be reformulated as follows:

- Progressive number.
- Topology name.
- Demand class.

As mentioned, table 5.3 shows the positive average net profit results obtained for Grid topology, associated with the corresponding demand class.

Demand (K)	Average Objective Value (euro)
50	73839
75	91047
100	108840

Table 5.3: Grid Topology: objective values for the integrated model

A direct proportional relationship between the two variables is evident: as k increases, the net profits (profits) also increase. In the following table, the results of the integrated model are presented. This model was developed to optimize the decision-making process and its benefits will be shown in the next paragraphs. The values reported provide insights into how different demand levels impact the objective function, hence the total profits. The whole dataset is available in Appendix B for a more thorough examination.

The results of the integrated model clearly reveal an upward trend in profits as demand rises, as the accompanying table illustrates. From a business standpoint,

Topology	Demand (K)	Average Objective Value (euro)
Grid	50	73839
	75	91047
	100	108840
Bipartite	50	71791
	75	88911
	100	110227
Lanza	15	55276
	20	90244
	25	101870

Table 5.4: Average profits for different topologies

this positive association indicates that higher demand levels result in higher overall profitability. These findings demonstrate that the model well reflects the dynamic of increasing revenues while preserving operational efficiency, which is a primary goal in any optimisation process. Furthermore, the model's approach is strengthened by the reality that earnings increase consistently with demand. It shows that there are no major inefficiencies in the system's ability to handle the increased workload and resource allocation. Next, the results of the non-integrated model will be introduced. A comparison between the two approaches will be provided, highlighting the differences in their performance. This analysis aims to demonstrate the optimization achieved through the integrated model by contrasting it with the results obtained from the non-integrated one. The comparison will emphasize the extent to which the integrated approach leads to improved efficiency and better outcomes in the decision-making process.

5.4 Comparison between the 2 models

The second objective was to assess the value of integrating bin packing consideration into the model. We perform this analysis by measuring the difference in the net revenues obtained by solving our integrated model, with respect to a non-integrated approach which solves the problem without considering bin packing constraints and tries to integrate them a posteriori, gradually increasing the flexibility of the decision-making pro- cess. More precisely, we examine two two-step decision processes, with different flexibility in the re-optimization of the resource utilization. In the first case, the service network is devised on the basis of the solution of the problem without bin packing consideration. Keeping fixed the service selection (only the y variables will be fixed) the plan is then adjusted to eventually reassign items to bins over the network of open services. This entails reoptimizing the

itineraries of the items and possibly changing the assignments decision of items to services. In the second policy (case), the service network and the assignment of items to services is devised on the basis of the solution of the problem without bin packing consideration. More in detail, the policy fixes the service and the items itineraries (and consequently the assignment of items to services) and tries, for each service, to accommodate the items into the bins associated to that service [10].

Data has been gathered and arranged in Excel to enable a more thorough study. Finding patterns and trends is made easier using this method since it enables a more organised and transparent analysis of the data. Additional statistics have been calculated and informative visual representations of the data, including graphs and summary tables, have been produced by utilising Excel's features. This kind of data organisation has improved comprehension of the effects of varying demand levels on profitability. To replicate the same example, the figure presents the results of the same instances shown in figure 5.7 of Lanza, with K = 15. However, this time, the results correspond to the non-integrated model. This allows for a direct comparison between the two approaches, highlighting the differences in performance and optimization outcomes. By analyzing these results, it is possible to assess how the absence of integration affects the overall efficiency of the solution and whether deviations emerge when compared to the integrated model.

summary_complete.put

INSTANCE	STATUS	OBJ. VALUE	ACCEPTANCE RATE
fed sol of SND into our model 10S-15-282-1-28	Optimal	29752.00	67%
fed sol of SND into our model 14S-15-282-10-28	Optimal	58679.00	93%
fed sol of SND into our model 11S-15-282-11-28	Optimal	47257.00	73%
fed sol of SND into our model 14S-15-282-12-28	Optimal	56768.00	93%
fed sol of SND into our model 14S-15-282-13-28	Optimal	75419.00	93%
fed sol of SND into our model 13S-15-282-14-28	Optimal	71639.00	87%
fed sol of SND into our model 12S-15-282-15-28	Optimal	62834.00	80%
fed sol of SND into our model 13S-15-282-16-28	Optimal	64380.00	87%
fed sol of SND into our model 13S-15-282-17-28	Optimal	59551.00	87%
fed sol of SND into our model 13S-15-282-18-28	Optimal	57848.00	87%
fed sol of SND into our model 11S-15-282-19-28	Optimal	35324.00	73%
fed sol of SND into our model 11S-15-282-2-28	Optimal	34475.00	73%
fed sol of SND into our model 13S-15-282-20-28	Optimal	64728.00	87%
fed sol of SND into our model 11S-15-282-3-28	Optimal	39950.00	73%
fed sol of SND into our model 9S-15-282-4-28	Optimal	21924.00	60%
fed sol of SND into our model 13S-15-282-5-28	Optimal	64696.00	87%
fed sol of SND into our model 12S-15-282-6-28	Optimal	48021.00	80%
fed sol of SND into our model 11S-15-282-7-28	Optimal	40732.00	73%
fed sol of SND into our model 10S-15-282-8-28	Optimal	25301.00	67%
fed sol of SND into our model 11S-15-282-9-28	Optimal	40388.00	73%

Figure 5.7: Output of non-integrated model

The performance comparison between the two models was conducted in Excel, allowing for the calculation of the average profit increase and the average acceptance rate for each topology and demand class. This approach made it possible to analyze the impact of the integrated model across different scenarios, highlighting its efficiency in handling various demand structures. By aggregating results across

multiple instances, the analysis provides an extensive view of how the integrated approach improves overall system performance. Figure 5.8 compares the separate and integrated models for the Grid topology with K = 50, showing their impact on objective value, profit increase, and acceptance rate. Three key points emerge from the image:

- The average objective value is 64944 for the Separate model and 73839 for the integrated one.
- This means an average profit increase of 13.7%, proving that the integrated model performs better.
- The average acceptance rate is 69% indicating a good level of service and the model's ability to accommodate a significant portion of the demand.

GRID TOPOLOGY - K = 50	OBJECTIVE VALUE			
INSTANCE	SEPARATE	INTEGRATED	PROFIT INCREASE	ACCEPTANCE RATE
01-Grid50	54022	69402	28.5%	64%
02-Grid50	61728	71261	15.4%	68%
03-Grid50	63093	68383	8.4%	68%
04-Grid50	67819	86102	27.0%	72%
05-Grid50	61158	63255	3.4%	68%
06-Grid50	67733	81688	20.6%	70%
07-Grid50	62382	75004	20.2%	68%
08-Grid50	56482	63350	12.2%	68%
09-Grid50	57557	59413	3.2%	64%
10-Grid50	97466	100536	3.1%	80%
AVERAGE VALUES	**64944**	**73839**	**13.7%**	**69%**

Figure 5.8: Grid-50: Comparison

The profit increase is calculated by dividing the difference between the objective value in the integrated model and the objective value in the separate model by the objective value in the separate model. These results confirm that the integrated model is more efficient in increasing profits. The same results are shown in the image 5.9 in a bar chart illustrating the profit gap between the two models: separate or unintegrated (blue) and integrated (orange). The X-axis displays the ten instances executed for this specific class of demand, while the Y-axis represents profit values, ranging up to approximately 120000. Each instance is associated with two bars, enabling a direct comparison of profitability between the two models. The data clearly show that the integrated model consistently outperforms the separate model in all instances. Although in some cases the difference is minimal, in others, such as "04-Grid50" and "07-Grid50," the overall gap is enough larger as seen in the previous figure with average profit increase, demonstrating an advantage of the integrated approach. This bar chart provides an immediate visual representation of the differences between the two models. Observing the variations in bar heights, it is possible to quickly identify the superior performance of the integrated model in every instance. The complete dataset can be found in Appendix B, where all

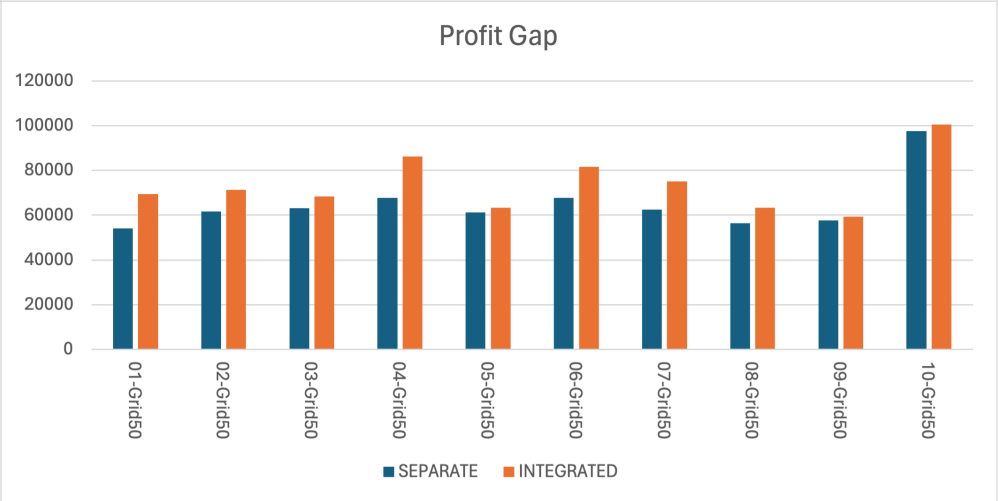


Figure 5.9: Grid-50: Bar Chart

the detailed values are reported. In this section, the discussion will focus on the general results obtained for each topology, as well as the average profit increase, which takes into account all 120 instances executed for this thesis work. Rather than analyzing individual cases, the goal is to provide a broader overview of the performance differences between the models.

Below, bar charts are presented for each topology, visually illustrating the comparison between different models. In addition to the histograms, tables are also provided, summarizing the average profit increase for each demand class within every topology. These tables allow for a clearer understanding of how the integrated approach impacts profitability across various scenarios, offering a more structured interpretation of the results.

Figure 5.10 illustrates the average profit increase calculated across all cases related to the Grid topology, meaning that the values take into account all 30 instances (10 for each demand class) executed for this specific topology. This visual representation provides an immediate and intuitive understanding of how the integrated approach performs in comparison to the separate approach across the entire dataset for this topology. In contrast, table 5.5 presents a more detailed breakdown, showing the average profit increase corresponding to each demand class within the Grid topology. This allows for a clearer analysis of how different demand patterns influence the overall profitability improvements, offering a more structured interpretation of the results. While in the case of the table 5.3 showing the relationship between the number of requests and the objective value, a directly proportional trend was observed between the two variables, the situation is different when analyzing the percentage profit increase between the two models. In this case,

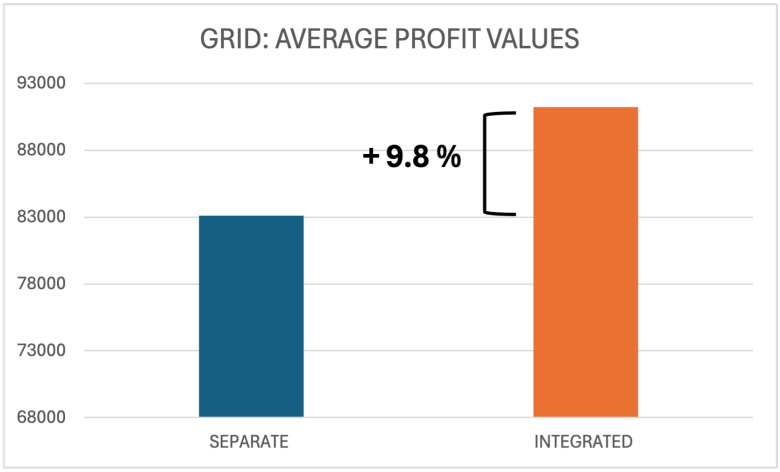


Figure 5.10: Grid: Average Delta Profit

in table 5.5 the relationship between the number of requests and the percentage profit increase is inversely proportional. This means that as the number of requests increases, the relative profit improvement achieved by the integrated model compared to the separate model tends to decrease. This inverse trend suggests that the advantages of integration are more significant when the number of requests is lower, while the gap between the two models becomes smaller as the request volume grows. Below, table 5.6 is presented that summarizes these findings for

Demand (K)	Average Profits Increase
50	13.7%
75	10.0%
100	7.1%

Table 5.5: Grid Topology: Average Profits Increase for Demand Class

the other two topologies as well. This table provides a full overview, allowing for a direct comparison of how the relationship between demand and profit increase behaves across different network configurations. In both of the additional cases, a similar inversely proportional relationship between demand and profit difference is observed. This confirms the trend seen in the Grid topology, reinforcing the idea that the relative advantage of the integrated model decreases as the number of requests increases. This pattern suggests that while the integrated approach consistently outperforms the separate model, its impact is more pronounced when the request volume is lower, gradually diminishing as the demand grows.

Finally, the histograms below show the average increase, considering all the instances of the two remaining topologies, without categorizing them based on the

Topology	Demand (K)	Average Profits Increase
Grid	50	13.7%
	75	10.0%
	100	7.1%
Bipartite	50	9.9%
	75	7.9%
	100	7.8%
Lanza	15	10.6%
	20	6.1%
	25	4.3%

Table 5.6: Average profits increase for different topologies

demand class. Even in the case of the last two topologies, the results continue to demonstrate the advantages of the integrated model, confirming its effectiveness and superiority in comparison to alternative approaches.

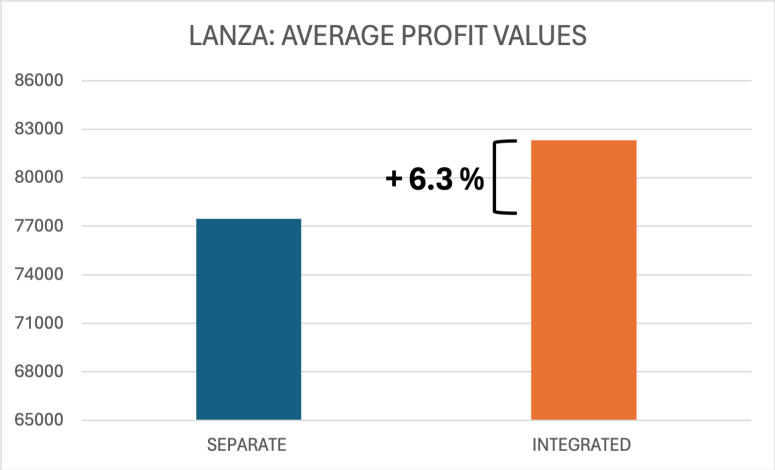


Figure 5.11: Lanza: Average Delta Profit

The chart in the figure 5.13 is an effective way to visualize what has just been observed in the various tables. It illustrates how the percentage change in average profits evolves across three different topologies, as the value of K increases. Overall, the trend shows a steady decline in profit improvements for all topologies as K grows, indicating that the relative benefit of these configurations diminishes with larger values. Finally, the average is also calculated across all 120 values, considering all instances of the three topologies and including all demand classes. The results reveal an overall profit increase of 7.8%, which is gained by adopting the integrated model. This outcome further highlights the advantages of the integrated approach in

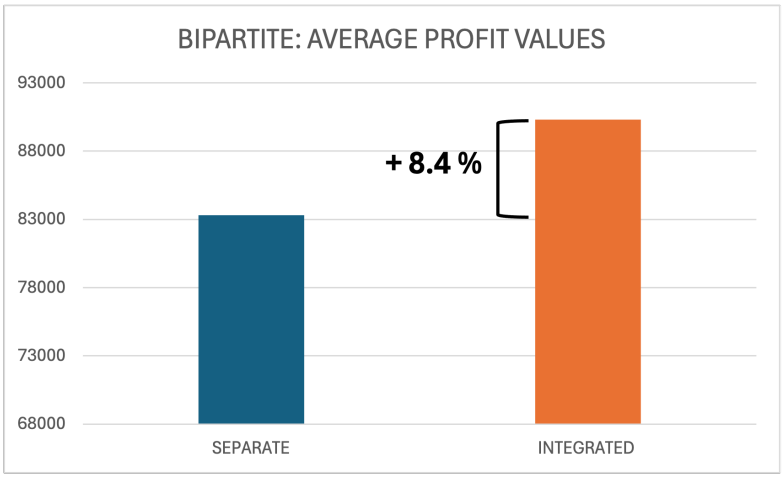


Figure 5.12: Bipartite: Average Delta Profit

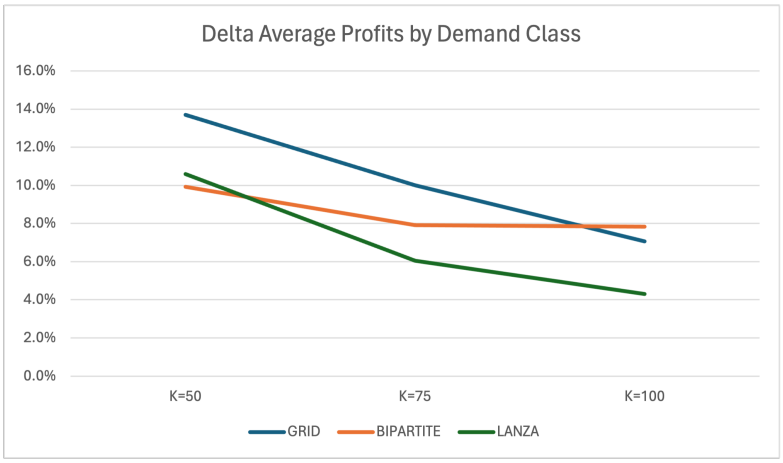


Figure 5.13: Delta Average Profits by Demand Class and Topology

improving overall profitability across different configurations and demand scenarios. Results are shown in figure 5.14. The results of this analysis clearly demonstrate that the integrated model consistently performs better than the separate model in terms of both profitability and efficiency. The advantages of integration are evident across different levels of analysis: first, when comparing each demand class within every topology, second, when considering each topology as a whole, and finally, when looking at individual instances regardless of topology. In all these cases, the integrated model results in higher profits and a well-structured acceptance rate, proving its superiority in optimizing decision-making.

However, a key trend that emerges is that the benefits of the integrated model tend to decrease as demand increases. This suggests that integration is particularly

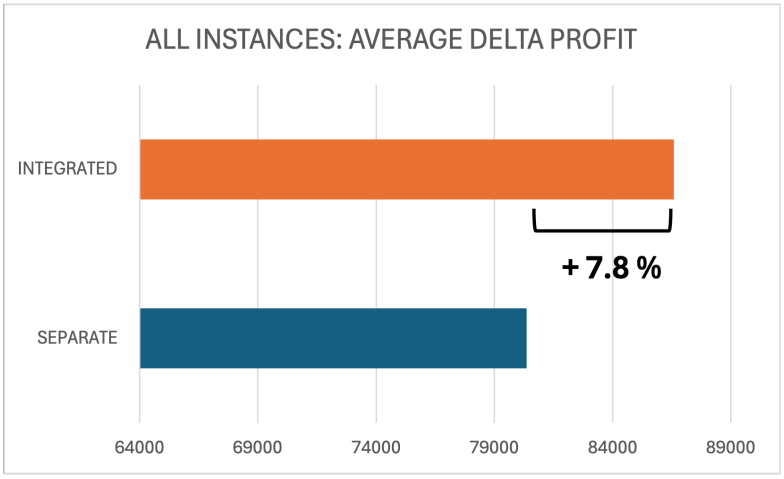


Figure 5.14: All Instances: overall benefits from integrated model

effective when the system operates under moderate or lower demand conditions, where optimization has a greater impact on resource allocation and profitability. Despite this gradual reduction in benefits, the improvements remain evident across all scenarios, confirming that the integrated model represents a more efficient and structured approach to managing logistics and optimizing operations. The results underline that, even if the magnitude of the benefits may vary, the integrated model consistently provides a more profitable and strategically sound solution.

5.5 Discussion and Conclusion

To conclude this work, it is important to summarize the key steps taken throughout the thesis. The thesis work began with the creation of instances in Excel in order to develop distinct scenarios that represented varied operational settings. These instances were designed to simulate different network topologies and demand classes, ensuring that the models could be tested under diverse conditions. Having a structured dataset that could be used to assess how well various optimisation techniques performed was the aim. To make sure the Excel instances were properly constructed and that the data was consistent, it was crucial to check them after they were generated. VBA scripts were created for this purpose and used to verify that the instances that were generated were proper. By assisting in the detection and correction of potential data mistakes, these scripts made sure that the input given to the models was trustworthy and representative of actual situations. Once the instances were validated, the next step was to estimate costs, which were used as a proxy to determine profits. The profit of each request was computed by multiplying a reference parameter (parProfit) by a random factor uniformly distributed between

0.8 and 1.4, and then dividing by the total number of requests. This approach ensured that profits varied across different requests while maintaining a consistent overall distribution.

With the input data ready, the integrated model was developed and tested. This model combines the SND problem with the BPP, meaning that the allocation of requests and the optimization of space utilization were handled simultaneously. The integrated model aimed to maximize profits while efficiently assigning and transporting demand, taking into account capacity constraints and operational costs. The results showed that the integrated model led to higher profits and better resource utilization, making it a more efficient solution compared to a separate approach. A non-integrated model, in which the two optimisation issues (SND and BPP) were solved independently, was also used to evaluate the true impact of integration. This made it possible to compare the integrated and non-integrated methods directly. The outcomes demonstrated that the integrated model produced notable efficiency gains and consistently beat the non-integrated model. The final part of the analysis involved a quantitative comparison between the two models, conducted in Excel. The average profit increase and the acceptance rate were calculated for each demand class within every topology, for each topology as a whole, and for all instances without distinguishing by topology. The results confirmed that, regardless of the scenario, the integrated model consistently delivered better performance. The results show an inverse relationship between demand levels and the benefits of the integrated model. As demand increases, the advantage of integration tends to decrease, likely due to the system reaching a saturation point where optimization has less impact. However, even at higher demand levels, the integrated model still provides better efficiency and profitability compared to the non-integrated approach. In conclusion, this thesis demonstrated the advantages of integrating service network design with bin packing. The integrated model resulted in higher profitability and better resource allocation, making it a more effective and strategically advantageous approach. While the impact of integration becomes less pronounced as demand increases, its benefits remain evident across all cases, proving that this method can be applied successfully to optimize logistics and transportation operations.

Appendix A

Full Scripts

A.1 Scripts for Windows

A.1.1 Check Duplicated-NonCyclic

```
1     Sub CheckDuplicated_NonCyclic()
2     Dim ws As Worksheet
3     Dim lastRow As Long
4     Dim i As Long
5     Dim currentKey As String
6     Dim dict As clsDictionary
7     Dim duplicates As String
8     Dim count As Long
9     Set ws = ThisWorkbook.ActiveSheet
10
11    ' To find the last not empty row
12    lastRow = ws.Cells(ws.Rows.count, "B").End(xlUp).Row
13
14    ' To Initialize the dictionary as an object of the new class
      created
15    Set dict = New clsDictionary
16
17    ' output variables
18    duplicates = "Duplicated rows: " & vbCrLf
19    count = 0
20
21    ' Cycle on all lines and generate a key by concatenating the
      right values
22    For i = 2 To lastRow
23        currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-"
      & _
```

```
                    UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-" &
                    UCase(Trim(CStr(ws.Cells(i, "H").Value))) & "-" &
                    UCase(Trim(CStr(ws.Cells(i, "I").Value)))

        ' Existence check
        If dict.Exists(currentKey) Then
            duplicates = duplicates & "Row " & i & " (duplicate of row " & dict.Item(currentKey) & ")" & vbCrLf
            count = count + 1
        Else
            dict.Add currentKey, i
        End If
    Next i

  ' To show results
    If count > 0 Then
        MsgBox duplicates & vbCrLf & "Total number of duplicate rows: " & count, vbInformation
    Else
        MsgBox "There are no duplicate rows.", vbInformation
    End If
End Sub
```

A.1.2 Find and Update Duplicated-NonCyclic

```

Sub FindAndUpdateDuplicates_NonCyclic()
    Dim ws As Worksheet
    Dim lastRow As Long
    Dim i As Long
    Dim currentKey As String
    Dim dict As clsDictionary
    Dim updateDone As Boolean

    Set ws = ThisWorkbook.ActiveSheet
    lastRow = ws.Cells(ws.Rows.count, "B").End(xlUp).Row
    ' Dictionary is created thanks our new class defined
    Set dict = New clsDictionary
    ' Initialize the flag
    updateDone = False

    ' Cycle over all rows
    For i = 2 To lastRow
        ' Key Generation
```

```
20          currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-"
    & _
21                       UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
    & _
22                       UCase(Trim(CStr(ws.Cells(i, "H").Value))) & "-"
    & _
23                       UCase(Trim(CStr(ws.Cells(i, "I").Value)))
24      ' Key existence check
25          If dict.Exists(currentKey) Then
26              ' If it already exists, updates it
27              ws.Cells(i, "H").Value = WorksheetFunction.Round(Rnd() *
    7 + 1, 0)
28              ws.Cells(i, "I").Value = WorksheetFunction.Round(2 + Rnd
    () * 4, 0)
29              'Update key in dictionary
30
31              currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "
    -" & _
32                       UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
    & _
33                       UCase(Trim(CStr(ws.Cells(i, "H").Value))) & "-"
    & _
34                       UCase(Trim(CStr(ws.Cells(i, "I").Value)))
35
36                  'Since here we cannot directly use dictCurrentKey we
    have to do this
37                      If dict.Exists(currentKey) Then
38                          dict.Remove currentKey
39                      End If
40                          dict.Add currentKey, True '
41
42              ' Flag
43              updateDone = True
44          Else
45              dict.Add currentKey, i
46          End If
47      Next i
48      ' Final message
49      If updateDone Then
50          MsgBox "Duplicates have successfully been updated!",
    vbInformation
51      Else
52          MsgBox "No duplicates found.", vbInformation
53      End If
54  End Sub
```

A.1.3 Check Duplicated-Cyclic

```
1
2  'As NonCyclic one but with different key-cells
3  Sub CheckDuplicated_Cyclic()
4      Dim ws As Worksheet
5      Dim lastRow As Long
6      Dim i As Long
7      Dim currentKey As String
8      Dim dict As clsDictionary
9      Dim duplicates As String
10     Dim count As Long
11     Set ws = ThisWorkbook.ActiveSheet
12
13     lastRow = ws.Cells(ws.Rows.count, "B").End(xlUp).Row
14
15     Set dict = New clsDictionary
16
17
18     duplicates = "Duplicated rows: " & vbCrLf
19     count = 0
20
21     For i = 2 To lastRow
22         currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-"
       & _
23                        UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
       & _
24                        UCase(Trim(CStr(ws.Cells(i, "L").Value))) & "-"
       & _
25                        UCase(Trim(CStr(ws.Cells(i, "M").Value)))
26
27         If dict.Exists(currentKey) Then
28            duplicates = duplicates & "Row " & i & " (duplicate of row
       " & dict.Item(currentKey) & ")" & vbCrLf
29            count = count + 1
30     Else
31             dict.Add currentKey, i
32         End If
33     Next i
34
35     If count > 0 Then
36         MsgBox duplicates & vbCrLf & "Total number of duplicate rows:
       " & count, vbInformation
37     Else
38         MsgBox "There are no duplicate rows.", vbInformation
39     End If
40 End Sub
```

A.1.4 Find and Update Duplicated-Cyclic

```
As NonCyclic but with different reference cells and different Random FUnction
Sub FindAndUpdateDuplicates_Cyclic()
    Dim ws As Worksheet
    Dim lastRow As Long
    Dim i As Long
    Dim currentKey As String
   Dim dict As clsDictionary

    Dim updateDone As Boolean

    Set ws = ThisWorkbook.ActiveSheet
    lastRow = ws.Cells(ws.Rows.count, "B").End(xlUp).Row
    Set dict = New clsDictionary

    updateDone = False

    For i = 2 To lastRow
            currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-" & _
                        UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-" & _
                        UCase(Trim(CStr(ws.Cells(i, "L").Value))) & "-" & _
                        UCase(Trim(CStr(ws.Cells(i, "M").Value)))

        If dict.Exists(currentKey) Then
            ws.Cells(i, "L").Value = WorksheetFunction.Round(Rnd() * 7 + 1, 0)
            ws.Cells(i, "M").Value = WorksheetFunction.Round(1 + Rnd() * 9, 0)

             currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-" & _
                        UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-" & _
                        UCase(Trim(CStr(ws.Cells(i, "L").Value))) & "-" & _
                        UCase(Trim(CStr(ws.Cells(i, "M").Value)))

              If dict.Exists(currentKey) Then
                    dict.Remove currentKey
                End If
                    dict.Add currentKey, True
```

A.2 Scripts for MacOS

A.2.1 Check Duplicated-NonCyclic

```
1
2  Sub CheckDuplicated_NonCyclic()
3      Dim ws As Worksheet
4      Dim lastRow As Long
5      Dim i As Long
6      Dim currentKey As String
7      Dim dict As clsDictionary
8      Dim duplicates As String
9      Dim count As Long
10     Set ws = ThisWorkbook.ActiveSheet
11
12     ' To find the last not empty row
13     lastRow = ws.Cells(ws.Rows.count, "B").End(xlUp).Row
14
15     ' To Initialize the dictionary as an object of the new class
       created
16     Set dict = New clsDictionary
17
18     ' output variables
19     duplicates = "Duplicated rows: " & vbCrLf
20     count = 0
21
22     ' Cycle on all lines and generate a key by concatenating the
       right values
23     For i = 2 To lastRow
24         currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-"
       & _
25                      UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
       & _
26                      UCase(Trim(CStr(ws.Cells(i, "H").Value))) & "-"
       & _
27                      UCase(Trim(CStr(ws.Cells(i, "I").Value)))
28
29         ' Existence check
30          If dict.Exists(currentKey) Then
31            duplicates = duplicates & "Row " & i & " (duplicate of row
        " & dict.Item(currentKey) & ")" & vbCrLf
32            count = count + 1
33     Else
34             dict.Add currentKey, i
35         End If
36     Next i
37
```

```
    ' To show results
    If count > 0 Then
        MsgBox duplicates & vbCrLf & "Total number of duplicate rows: " & count, vbInformation
    Else
        MsgBox "There are no duplicate rows.", vbInformation
    End If
End Sub
```

A.2.2 Find and Update Duplicated-NonCyclic

```
Sub FindAndUpdateDuplicated_NonCyclic()
    Dim ws As Worksheet
    Dim lastRow As Long
    Dim i As Long
    Dim currentKey As String
    Dim dict As clsDictionary
    Dim updateDone As Boolean

    Set ws = ThisWorkbook.ActiveSheet
    lastRow = ws.Cells(ws.Rows.count, "B").End(xlUp).Row
    ' Dictionary is created thanks our new class defined
    Set dict = New clsDictionary
    ' Initialize the flag
    updateDone = False

    ' Cycle over all rows
    For i = 2 To lastRow
        ' Key Generation
        currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-" & _
                     UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-" & _
                     UCase(Trim(CStr(ws.Cells(i, "H").Value))) & "-" & _
                     UCase(Trim(CStr(ws.Cells(i, "I").Value)))
      ' Key existence check
        If dict.Exists(currentKey) Then
            ' If it already exists, updates it
            ws.Cells(i, "H").Value = WorksheetFunction.Round(Rnd() * 7 + 1, 0)
            ws.Cells(i, "I").Value = WorksheetFunction.Round(2 + Rnd() * 4, 0)
            'Update key in dictionary

```

```
30              currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "
    -" & _
31                           UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
    & _
32                           UCase(Trim(CStr(ws.Cells(i, "H").Value))) & "-"
    & _
33                           UCase(Trim(CStr(ws.Cells(i, "I").Value)))
34
35                  'Since here we cannot directly use dictCurrentKey we
    have to do this
36                           If dict.Exists(currentKey) Then
37                               dict.Remove currentKey
38                          End If
39                              dict.Add currentKey, True '
40
41                  ' Flag
42                  updateDone = True
43              Else
44                  dict.Add currentKey, i
45              End If
46      Next i
47      ' Final message
48      If updateDone Then
49          MsgBox "Duplicates have successfully been updated!",
    vbInformation
50      Else
51          MsgBox "No duplicates found.", vbInformation
52      End If
53  End Sub
```

A.2.3 Check Duplicated-Cyclic

```
1
2   'As NonCyclic one but with different key-cells
3   Sub CheckDuplicated_Cyclic()
4       Dim ws As Worksheet
5       Dim lastRow As Long
6       Dim i As Long
7       Dim currentKey As String
8       Dim dict As clsDictionary
9       Dim duplicates As String
10      Dim count As Long
11      Set ws = ThisWorkbook.ActiveSheet
12
13      lastRow = ws.Cells(ws.Rows.count, "B").End(xlUp).Row
14
```

```
15      Set dict = New clsDictionary
16
17
18      duplicates = "Duplicated rows: " & vbCrLf
19      count = 0
20
21      For i = 2 To lastRow
22          currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) & "-"
    & _
23                       UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
    & _
24                       UCase(Trim(CStr(ws.Cells(i, "L").Value))) & "-"
    & _
25                       UCase(Trim(CStr(ws.Cells(i, "M").Value)))
26
27          If dict.Exists(currentKey) Then
28              duplicates = duplicates & "Row " & i & " (duplicate of row
     " & dict.Item(currentKey) & ")" & vbCrLf
29              count = count + 1
30      Else
31              dict.Add currentKey, i
32          End If
33      Next i
34
35      If count > 0 Then
36          MsgBox duplicates & vbCrLf & "Total number of duplicate rows:
     " & count, vbInformation
37      Else
38          MsgBox "There are no duplicate rows.", vbInformation
39      End If
40  End Sub
```

A.2.4 Find and Update Duplicated-Cyclic

```
1
2   As NonCyclic but with different reference cells and different Random
        FUnction
3   Sub FindAndUpdateDuplicated_Cyclic()
4       Dim ws As Worksheet
5       Dim lastRow As Long
6       Dim i As Long
7       Dim currentKey As String
8      Dim dict As clsDictionary
9
10      Dim updateDone As Boolean
11
```

```
12      Set ws = ThisWorkbook.ActiveSheet
13      lastRow = ws.Cells(ws.Rows.count, "B").End(xlUp).Row
14      Set dict = New clsDictionary
15
16      updateDone = False
17
18      For i = 2 To lastRow
19              currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) &
    "-" & _
20                      UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
    & _
21                      UCase(Trim(CStr(ws.Cells(i, "L").Value))) & "-"
    & _
22                      UCase(Trim(CStr(ws.Cells(i, "M").Value)))
23
24          If dict.Exists(currentKey) Then
25              ws.Cells(i, "L").Value = WorksheetFunction.Round(Rnd() *
    7 + 1, 0)
26              ws.Cells(i, "M").Value = WorksheetFunction.Round(1 + Rnd
    () * 9, 0)
27
28                currentKey = UCase(Trim(CStr(ws.Cells(i, "B").Value))) &
     "-" & _
29                      UCase(Trim(CStr(ws.Cells(i, "F").Value))) & "-"
    & _
30                      UCase(Trim(CStr(ws.Cells(i, "L").Value))) & "-"
    & _
31                      UCase(Trim(CStr(ws.Cells(i, "M").Value)))
32
33                If dict.Exists(currentKey) Then
34                        dict.Remove currentKey
35                    End If
36                        dict.Add currentKey, True
```

A.2.5 Dictionary Class

```
1  Private dictKeys As Collection
2  Private dictItems As Collection
3  Private Sub Class_Initialize()
4      Set dictKeys = New Collection
5      Set dictItems = New Collection
6  End Sub
7  ' to add an element
8  Public Sub Add(key As String, Item As Variant)
9      Dim index As Long
10     On Error Resume Next
```

```
11      ' to find thekey
12      index = dictKeys(key)
13      If Err.Number = 0 Then
14          Err.Raise 457, , "Chiave già esistente"
15      End If
16      On Error GoTo 0
17     'if  there is not error, add the key and the element in the
       collection
18      dictKeys.Add key, key
19      dictItems.Add Item, key
20  End Sub
21  ' to verify if key exists
22  Public Function Exists(key As String) As Boolean
23      Dim k As Variant
24      On Error Resume Next
25      For Each k In dictKeys
26          If k = key Then
27              Exists = True
28              On Error GoTo 0
29              Exit Function
30          End If
31      Next k
32      Exists = False
33      On Error GoTo 0
34  End Function
35
36  ' to get a value with key
37  Public Function Item(key As String) As Variant
38      Item = dictItems(key)
39  End Function
40
41  ' to remove a key and an element
42  Public Sub Remove(key As String)
43      dictKeys.Remove key
44      dictItems.Remove key
45  End Sub
```

Appendix B

Tables

B.1 Bipartite: Full Results

Instance	Separate	Integrated
01-Bip50	60716	63918
02-Bip50	46324	54767
03-Bip50	64890	69063
04-Bip50	64386	66750
05-Bip50	79546	89062
06-Bip50	55120	55595
07-Bip50	56866	69583
08-Bip50	62588	67329
09-Bip50	98933	106639
10-Bip50	63673	75202
01-Bip75	82975	96216
02-Bip75	64221	74901
03-Bip75	62705	69124
04-Bip75	74130	73918
05-Bip75	96644	105959
06-Bip75	82287	85209
07-Bip75	83159	94002
08-Bip75	88905	83449
09-Bip75	111390	121140
10-Bip75	77540	85194
01-Bip100	91244	97518
02-Bip100	103298	110216
03-Bip100	94699	103292
04-Bip100	84465	100340

Instance	Separate	Integrated
05-Bip100	102063	113569
06-Bip100	94962	106286
07-Bip100	101108	108682
08-Bip100	120221	124734
09-Bip100	136382	135833
10-Bip100	93789	101797

B.2 Grid: Full Results

Instance	Separate	Integrated
01-Grid50	54022	69402
02-Grid50	61728	71261
03-Grid50	63093	68383
04-Grid50	67819	86102
05-Grid50	61158	63255
06-Grid50	67733	81688
07-Grid50	62382	75004
08-Grid50	56482	63350
09-Grid50	57557	59413
10-Grid50	97466	100536
01-Grid75	87286	97474
02-Grid75	74033	79019
03-Grid75	82575	81650
04-Grid75	77659	88520
05-Grid75	75976	79216
06-Grid75	85363	102889
07-Grid75	86297	100501
08-Grid75	76174	88978
09-Grid75	75260	85077
10-Grid75	107090	107147
01-Grid100	118545	128355
02-Grid100	85320	87749
03-Grid100	82701	90243
04-Grid100	98360	104008
05-Grid100	83652	90799
06-Grid100	113007	126151
07-Grid100	103966	112025
08-Grid100	119266	125543

Instance	Separate	Integrated
09-Grid100	94373	104295
10-Grid100	117468	119233

B.3 Lanza: Full Results

Instance	Separate	Integrated
01-Lanza15	29752	31872
02-Lanza15	34475	39272
03-Lanza15	39950	53141
04-Lanza15	21924	28881
05-Lanza15	64696	59979
06-Lanza15	48021	53637
07-Lanza15	40732	40436
08-Lanza15	25301	29176
09-Lanza15	40388	48640
10-Lanza15	58679	72060
11-Lanza15	47257	45740
12-Lanza15	56768	68658
13-Lanza15	75419	77266
14-Lanza15	71639	70668
15-Lanza15	62834	63483
16-Lanza15	64380	68635
17-Lanza15	59551	75292
18-Lanza15	57848	65307
19-Lanza15	35324	45477
20-Lanza15	64728	67893
01-Lanza20	62955	77130
02-Lanza20	72004	86271
03-Lanza20	71803	72792
04-Lanza20	70323	77140
05-Lanza20	76677	79779
06-Lanza20	63595	65461
07-Lanza20	54834	68981
08-Lanza20	79331	76605
09-Lanza20	63688	86400
10-Lanza20	95910	94630
11-Lanza20	105501	104654
12-Lanza20	111295	124664

Instance	Separate	Integrated
13-Lanza20	98683	103398
14-Lanza20	85560	94955
15-Lanza20	98820	96898
16-Lanza20	96119	106585
17-Lanza20	113357	108048
18-Lanza20	95835	84346
19-Lanza20	100473	105898
20-Lanza20	72527	78097
01-Lanza25	66611	65132
02-Lanza25	82437	90966
03Lanza25	61370	60461
04-Lanza25	102467	108993
05-Lanza25	105134	123293
06-Lanza25	66718	85854
07-Lanza25	106024	111970
08-Lanza25	69792	75691
09-Lanza25	85249	88163
10-Lanza25	62479	74470
11-Lanza25	124543	134110
12-Lanza25	88252	95117
13-Lanza25	120412	120852
14-Lanza25	98351	95899
15-Lanza25	131969	133853
16-Lanza25	112747	124110
17-Lanza25	119737	107493
18-Lanza25	128364	128976
19-Lanza25	112099	95167
20-Lanza25	108663	116823

B.4 Comparison

Demand Class	Bipartite	Grid	Lanza
K=50	13.7%	9.9%	10.6%
K=75	10.0%	7.9%	6.1%
K=100	7.1%	7.8%	4.3%

Bibliography

[1] Teodor Gabriel Crainic. «Service Network Design for consolidation-based transportation-the fundamentals». In: B (Sept. 2024) (cit. on pp. 1, 9).

[2] Teodor Gabriel Crainic. «Service Network Design for consolidation-based transportation- Advanced Topics». In: (Sept. 2024) (cit. on pp. 6–8).

[3] Paolo Toth Silvano Martello. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990 (cit. on p. 11).

[4] Deniz Tursel Eliiyi Ugur Eliiyi. «Applications of bin packing models through the supply chain». In: *International journal of business and management* 1.1 (2009) (cit. on pp. 11, 12).

[5] Daniele Vigo Silvano Martello David Pisinger. «The Three-Dimensional Bin Packing Problem». In: *Operations Research* (1997) (cit. on p. 12).

[6] M.G.Speranza J.F Cotè G.Guastaroba. «The value of integrating loading and routing». In: *European Journal of Operational Research* 257.1 (2017), pp. 89–105 (cit. on pp. 13, 14).

[7] Mohamed Haouari Tülay Flamand Manuel Iori. «The transportation problem with packing constraints». In: *Computers and Operations Research* (2023) (cit. on p. 14).

[8] Fabien Lehuédé Mike Hewitt. «New formulations for the Scheduled Service Network Design Problem». In: *Transportation Research Part B* (Apr. 2023), pp. 117–133 (cit. on p. 14).

[9] Guido Perboli Maria Elena Bruni Teodor Crainic. «Bin Packing Methodologies for Capacity Planning in Freight Transportation and Logistics». In: https://doi.org/10.1007/9: Springer, Cham, 2024, pp. 115–147 (cit. on p. 18).

[10] Guido Perboli, Sara Khodaparasti, Walter Rei, Maria Elena Bruni, and Teodor Gabriel Crainic. «Scheduled Service Network Design with Packing Considerations». In: *Proc. The ninth international workshop on freight transportation and logistics*. 2024 (cit. on pp. 18, 58).

[11] Marcel Roelofs Johannes Bisschop, ed. *Aimms - User's Guide*. Paragon Decision Technology, 2006 (cit. on pp. 35, 37, 42).

[12] Giacomo Lanza, Teodor Gabriel Crainic, Walter Rei, and Nicoletta Ricciardi. «Scheduled service network design with quality targets and stochastic travel times». In: *European Journal of Operational Research* 288 (2021), pp. 30–46 (cit. on p. 49).

[13] Gita Taherkhania, Ioana C. Bileganb, Teodor Gabriel Crainic, and Michel Gendreaud, Walter Rei. «Tactical capacity planning in an integrated multi-stakeholder freight transportation system». In: *European Journal of Operational Research* (2022) (cit. on p. 50).