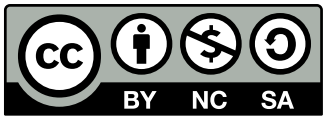
Aalto University
School of Science

Master's Programme in Mathematics and Operations Research

A Hybrid Metaheuristic for Efficient Elevator Zoning in Buildings with Sky Lobbies

Niko Miller

Master's Thesis
2025



Aalto University
School of Science

Author Niko Miller

Title A Hybrid Metaheuristic for Efficient Elevator Zoning in Buildings with Sky Lobbies

Degree programme Mathematics and Operations Research

Major Applied Mathematics

Supervisor Prof. Fabricio Oliveira

Advisor Mr. Mirko Ruokokoski (MSc)

Collaborative partner KONE Corporation

Date April 15, 2025 **Number of pages** 96 **Language** English

Abstract

This thesis studies the problem of optimal sky lobby placement in high-rise buildings – a central but previously unsystematized question in elevator system design. The problem is examined jointly with elevator zoning, and the thesis presents the first formal mathematical model that captures their joint structure. To solve the problem, a novel hybrid metaheuristic method is developed, combining simulated annealing for sky lobby placement with a hidden genes genetic algorithm for zoning the resulting building segments. The method is applied to an extensive computational study across a wide range of building parameters, enabling identification of general patterns in optimal solutions and challenging several prevailing design heuristics. The work establishes a systematic foundation for future research on elevator system design in high-rise buildings and opens numerous directions for extending the model to more complex real-world settings.

Keywords Sky lobbies, Elevator zoning, Elevator system design, High-rise buildings, Combinatorial optimization, Metaheuristics, Hybrid metaheuristics, Simulated annealing, Genetic algorithm, Hidden genes genetic algorithm

Aalto-yliopisto
Perustieteiden
korkeakoulu

Tekijä Niko Miller

Työn nimi A Hybrid Metaheuristic for Efficient Elevator Zoning in Buildings with Sky Lobbies

Koulutusohjelma Mathematics and Operations Research

Pääaine Applied Mathematics

Työn valvoja Prof. Fabricio Oliveira

Työn ohjaaja DI Mirko Ruokokoski

Yhteistyötaho KONE Corporation

Päivämäärä 15.4.2025 **Sivumäärä** 96 **Kieli** englanti

Tiivistelmä
Tämä diplomityö tarkastelee korkeiden rakennusten yläodotustilojen (sky lobby) optimaalisen sijoittelun ongelmaa, joka on keskeinen mutta aiemmin järjestelmällisesti tutkimaton kysymys hissijärjestelmien suunnittelussa. Ongelmaa tarkastellaan yhdessä hissien vyöhyköinnin (zoning) kanssa, ja työssä esitetään ensimmäinen matemaattinen malli, joka huomioi näiden kahden rakenteen yhteisriippuvuuden. Ongelman ratkaisemiseksi kehitettiin uusi hybridimetaheuristiikka, joka yhdistää simuloidun jäähdytyksen yläodotustilojen sijoitteluun ja hidden genes -geneettisen algoritmin rakennussegmenttien hissien vyöhyköintiin. Menetelmää sovellettiin laajaan laskennalliseen tutkimukseen vaihtelevilla rakennusparametreilla, minkä ansiosta tunnistettiin yleisiä piirteitä optimaalisissa ratkaisuissa ja haastettiin useita vallitsevia suunnitteluheuristiikkoja. Työ luo systemaattisen perustan korkeiden rakennusten hissijärjestelmien suunnittelun jatkotutkimukselle ja tarjoaa lukuisia mahdollisuuksia mallin laajentamiseen monimutkaisempien rakennusasetelmien tarkasteluun.

Avainsanat Yläodotustilat, Hissien vyöhyköinti, Hissijärjestelmien suunnittelu, Pilvenpiirtäjät, Kombinatorinen optimointi, Metaheuristiikat, Hybridi-metaheuristiikat, Simuloitu jäähdytys, Geneettinen algoritmi, Hidden genes -geneettinen algoritmi

Preface

I would like to express my sincere gratitude to my advisor, Mirko Ruokokoski, who introduced me to this thesis topic and trusted me with it from the outset. His expertise in optimization and deep knowledge of the elevator industry were invaluable throughout the project. I am particularly thankful for his generosity in allowing me to build upon parts of his existing codebase, which provided a strong technical foundation for my own work. I especially appreciate his flexibility and commitment during the final phases of the project, when deadlines grew tighter and the workload intensified.

I also wish to thank my supervisor, Professor Fabricio Oliveira, for his valuable guidance and constructive feedback throughout the thesis process. His expertise was instrumental in making the computational aspects of this work a distinctive strength. His comments on computational tractability were critical in shaping the development of the final model. Furthermore, his advice on hyperparameter tuning and effective use of high-performance computing resources enabled me to conduct extensive computational experiments that would not have been feasible otherwise. His academic insight helped me grow as a researcher and deliver a thesis I am genuinely proud to present.

I am grateful to KONE Corporation for the opportunity to carry out this thesis project. Contributing to applied research in high-rise elevator systems has been both intellectually stimulating and highly motivating. The thought that the methods developed in this work might one day support the design of the world's tallest buildings has been a constant source of inspiration. Working at KONE Building made the experience especially memorable – and the occasional moments spent on the 16th-floor terrace, gazing at the sunlight shimmering on the sea, often helped me step back and see the bigger picture.

I would also like to thank Aalto University for providing an outstanding academic environment. My interdisciplinary studies, the vibrant student culture, and my exchange semester in San Diego, California, are among the most memorable highlights of my time at Aalto. As the saying goes: *"A man can leave Otaniemi, but Otaniemi never leaves a man."* I could not agree more.

Finally, I want to thank my family and friends for their unwavering support and belief in me. Their curiosity, encouragement, and sincere interest in my work have been a continuous source of motivation. It has been deeply meaningful to feel that this journey has mattered not only to me but also to those around me.

Keilaniemi, April 15, 2025

Niko M. Miller

Contents

Symbols and Abbreviations

Symbols

L	Number of elevators in a group
C	Elevator car capacity (persons)
d	Floor height (meters)
v	Rated elevator speed (m/s)
m	Number of sky lobbies (or number of zones, depending on context)
N	Total number of populated floors (excluding ground floor)
N_P	Total number of passengers over all floors
N_k	Number of passengers on floor k
p_k	Fraction of passengers on floor k
P	Expected number of passengers in an elevator car
H	Expected reversal floor
S	Expected number of stops per trip
t_v	Transit time between adjacent floors
t_s	Additional time per stop (door and deceleration time)
t_p	Average passenger entering/exiting time
Z	Number of possible zoning arrangements
$S_m(N)$	Number of valid ways to place m sky lobbies among N floors
$f(x)$	Objective function
p_c	Crossover probability
p_m	Mutation probability

Abbreviations

RTT	Round-trip Time
INT	Interval
NTT	Nominal Travel Time
CLF	Car Load Factor
HC5	Handling Capacity (persons carried in 5 min)
%HC5	HC5 relative to the building population
DCS	Destination Control System
SA	Simulated Annealing
GA	Genetic Algorithm
HGGA	Hidden Genes Genetic Algorithm
COP	Combinatorial Optimization Problem
TSP	Traveling Salesman Problem
HPC	High-Performance Computing
CLT	Call Time
WT	Waiting Time
TT	Transit Time
TTD	Time to Destination
RD	Ride Time
JT	Journey Time

Definitions

Elevator Terminology

Main Lobby	The primary entrance floor of a building, from which passengers typically begin their elevator journey.
Sky Lobby	A lobby (transfer) floor located above the main lobby where passengers switch from shuttle elevators to local elevators (or vice versa).
Shuttle Elevator	A large-capacity elevator (single- or double-deck) that travels non-stop between the main lobby floor and one or more sky lobby floors.
Local Elevator	An elevator that provides service to a designated set of floors (a zone) within a building stack, typically starting from either the main lobby (if no sky lobby exists) or a sky lobby.
Stack	A building segment between the main lobby and a sky lobby, between two sky lobbies, or between the highest sky lobby and the top floor. Each stack is served by one or more local elevator groups.
Zone	A contiguous subset of floors within a stack, served by a particular elevator group. A single stack can be split into multiple zones.
Zoning	A design method that partitions the floors of a stack into one or more zones, each served by a dedicated elevator group, to reduce the number of stops and save core area.
Elevator Group	A set of elevators operating in a coordinated manner to serve a particular set of floors or one or more zones.
Double-deck Elevator	An elevator design featuring two stacked elevator cars that move together in a single shaft, enabling simultaneous boarding on two adjacent floors.
Shaft Area	The cross-sectional area required for each elevator shaft within the building's core.
Lobby Area	The total floor area directly allocated for passenger waiting and elevator entry/exit in front of a specific elevator excluding the area occupied by elevator shafts.
Core Area	The total area occupied by elevator shafts, lobbies, and supporting infrastructure (e.g., machine rooms, corridors) within the building's central core.

Service Quality Measures

Call Time	Time from when the landing call is given until the assigned elevator begins decelerating to the floor where the call was made.
Waiting Time	Time from when the passenger gives the call or joins the queue until the elevator begins opening its doors at the origin floor.
Transit Time	Time from when the elevator doors start opening at the origin floor until they start opening again at the destination floor.
Ride Time	Time from when the passenger enters the elevator until they exit at the destination floor.
Time to Destination	Time from when the passenger gives the call or joins the queue until the elevator begins opening its doors at the destination floor.
Journey Time	Time from when the passenger gives the call or joins the queue until they exit the elevator at the destination floor.

Traffic Terminology

Traffic Type	A classification of elevator traffic based on where passengers originate and where they exit (e.g., uppeak, downpeak, two-way, interfloor).
Traffic Pattern	A time-based variation in elevator traffic, as different traffic types can dominate during different periods (morning uppeak, midday two-way, evening downpeak).
Uppeak Traffic	A morning or arrival traffic pattern dominated by passengers traveling from the main lobby to upper floors.
Downpeak Traffic	An evening or departure traffic pattern dominated by passengers traveling from upper floors down to the main lobby.
Interfloor Traffic	Passenger flow that travels between floors within the building (excluding the main lobby), for example moving from one tenant floor to another tenant floor.
Round-Trip Time	The time (seconds) from when an elevator's doors open at the main lobby until they reopen at the main lobby after serving one or more upper floors.

Nominal Travel Time	The time (seconds) for an elevator to run express from the main lobby to its highest served floor with no intermediate stops.
Interval	The average time (seconds) between consecutive elevator cars departing from the main lobby in uppeak conditions.
Handling Capacity	The maximum number of passengers an elevator system can transport within a specified time frame (commonly a 5-minute period).

Metaheuristic Terminology

Metaheuristic	A high-level algorithmic framework designed to find near-optimal solutions to complex optimization problems by intelligently guiding lower-level heuristics.
Hybrid Metaheuristic	A metaheuristic approach that combines two or more algorithms (e.g., population-based and single-solution-based) to exploit their complementary strengths.
Simulated Annealing	A single-solution metaheuristic inspired by the metallurgical annealing process. It iteratively explores a neighborhood of solutions and employs a cooling schedule that decreases the likelihood of accepting worse solutions over time.
Genetic Algorithm	A population-based metaheuristic inspired by biological evolution. Solutions are encoded as chromosomes, and operators such as selection, crossover, and mutation iteratively improve the population.
Hidden Genes Genetic Algorithm	A variation of the genetic algorithm that handles variable-length or variable-cardinality solutions by allowing certain genes to be hidden (inactive), facilitating more flexible solution representations.
Neighborhood	In simulated annealing, the set of solutions obtained by making small or incremental changes to the current solution, from which the next candidate solution is chosen.
Cooling Schedule	In simulated annealing, the mechanism that controls how the "temperature" parameter decreases over time, affecting the probability of accepting worse solutions.

Mutation	In genetic algorithms, a genetic operator that randomly alters one or more genes in a chromosome-like structure, introducing new traits into a population.
Crossover	In genetic algorithms, a genetic operator that combines segments of two parent solutions to produce one or more offspring solutions, enabling the exchange of genetic material.
Selection	In genetic algorithms, a process for choosing which individuals (solutions) advance to the next generation, typically based on fitness or objective function quality.
Elitism	In genetic algorithms, a policy whereby the best solutions from one generation are automatically carried over to the next, ensuring the retention of top performers.

1 Introduction

In 2023, 186 buildings with a height of 200 m or more were completed worldwide, marking a record year for tall-building construction (Council on Tall Buildings and Urban Habitat, 2024). The number of tall buildings has grown exponentially and is now approximately ten times higher than at the turn of the millennium. The global total has surpassed 2,400 tall buildings, including 241 “super-tall" structures rising above 300 m, more than half of which have been completed since 2015. Furthermore, the average height of the world’s 100 tallest buildings has increased from 281.5 meters in 2020 to 409.6 meters in 2023. These trends demonstrate that the design of highly functional elevator systems for tall buildings is becoming increasingly important.

This proliferation of tall buildings would not have been possible without significant innovations in elevator system design. One such innovation was the introduction of sky lobbies. Since the 1970s, sky lobbies have become an integral component of elevator system design in high-rise buildings owing to their ability to significantly reduce the elevator system’s footprint and thereby increase the proportion of rentable area. A sky lobby is an intermediate transfer floor on which passengers switch from shuttle elevators to local elevators to reach their final destination. Without the use of sky lobbies, many of the world’s tallest buildings would be economically infeasible. A poorly designed elevator configuration not only increases waiting and travel times, but can also significantly inflate the elevator core, reducing the rentable area and overall profitability. Consequently, developers and building owners are strongly motivated to explore systematic methods for optimizing sky lobby configurations.

Because sky lobbies divide a building into distinct segments, known as stacks, the effectiveness of an elevator configuration involving sky lobbies cannot be evaluated independently of the elevator configurations within each stack. Elevator zoning is a method used to design optimal elevator configurations for stacks. This involves partitioning a stack into disjoint sets of contiguous floors, referred to as zones. Typically, each zone is served by a dedicated elevator group, and the zones do not share floors, except for the building entrance level. Exact methods have been developed to solve the zoning problem using single-objective formulations (Powell, 1971, 1975; Ruokokoski et al., 2018), whereas metaheuristic approaches have extended the problem to multi-objective settings (Viita-aho, 2019).

In contrast to elevator zoning, the optimal design of sky lobby configurations remains considerably more ambiguous in the literature. Schroeder (1984) argues that a 100-floor building is not economically feasible without a sky lobby and dedicated shuttle elevators between the ground floor and the sky lobby. While it has long been understood that there exists a threshold beyond which one or more sky lobbies become necessary, there is an ongoing debate regarding the precise location of that threshold.

Schroeder (1989) suggests that sky lobbies should be considered for buildings exceeding 40 – 50 floors. According to Fortune (1985), sky lobbies beyond approximately 60 floors are necessary. Barney and Al-Sharif (2015) similarly notes that conventional zoning can typically serve buildings up to 60 floors from a main terminal lobby, and

that the use of double-deck elevators can extend this limit to 80 floors. More recently, Siikonen (2024) claimed that, depending on factors such as tenant profile, population, floor height, and elevator technology, buildings with 70 – 90 floors may still be served without the need for sky lobbies.

Clearly, there is no consensus on when sky lobbies should be introduced. Nor is there agreement on how many sky lobbies should be used or on which floors they should be located. Furthermore, the potential core area savings are inconclusive. Although existing studies, such as Schroeder (1985), Schroeder (1989), and Siikonen (2024), offer valuable insights into the types of sky lobby configurations that may be advisable and the potential extent of core area savings, definitive conclusions remain unresolved. A key limitation of these studies is that the placement of sky lobby floors and the zoning of local elevator groups within stacks were largely arbitrary. As a result, the configurations proposed in the literature are almost certainly suboptimal.

A likely reason for the lack of consistent and conclusive results in the literature is the absence of a systematic optimization-based method for solving the sky lobby configuration problem. Precisely this gap is addressed in this thesis. Given information about a building's characteristics and population, this study seeks to answer the following research questions:

1. How many sky lobbies should be used?
2. On which floors should the sky lobbies be placed?
3. What is the potential reduction in elevator core area resulting from the use of optimally placed sky lobbies?

To address these questions, the problem of determining an optimal elevator configuration in a building with sky lobbies is formulated as an optimization problem with a single objective: minimizing the elevator core area. This minimization is subject to the conventional elevator design criteria to ensure adequate performance. A novel hybrid metaheuristic with nested structure is developed to solve this problem and applied to compute near-optimal solutions across a wide range of problem instances.

On the outer level of the proposed hybrid, simulated annealing (SA) – a single-solution based metaheuristic – is used to propose k sky lobby floors for a building. These sky lobbies partition the building into $k + 1$ distinct building segments (stacks). On the inner level, the hidden genes genetic algorithm (HGGA) – a population-based metaheuristic – is used to find the optimal zoning arrangement for the stacks. A shuttle elevator configuration is determined for stacks with a sky lobby. High performance computing resources are used to find optimal hyperparameters for the metaheuristic components, to cache partial solutions and lastly to compute the final results.

In contrast to earlier studies that relied on a limited number of arbitrarily selected configurations, the proposed method explores the entire search space of sky lobby configurations and converges on near-optimal solutions for any given set of parameters. As a result, this thesis presents a unified framework in which both sky lobby placement

and zoning configurations within stacks are optimized jointly, rather than being treated in isolation.

This thesis makes three main contributions. First, it provides a formal mathematical formulation of the sky lobby placement problem, which has not previously been established. Second, by presenting solutions across a wide range of building heights and population densities, this work offers new insights into when sky lobbies should be introduced, how many should be used, and the resulting elevator core area savings. These findings challenge prevailing industry assumptions, which are largely based on rules of thumb. Third, on the methodological side, this thesis provides further evidence of the effectiveness of the HGGA in variable-sized design space problems. Moreover, it is the first study to successfully employ HGGAs within a hybrid metaheuristic framework. Collectively, these contributions advance both the theoretical and practical understanding of sky lobby optimization and provide a solid foundation for future research on the topic.

The scope of this thesis is limited to office buildings with up to three sky lobbies and an evenly distributed population across floors. In practice, most designs feature one or two sky lobbies, and only the tallest or most complex buildings employ three or more. Analyzing up to three sky lobbies thus captures the majority of real-world use cases. The traffic type is assumed to be up-peak, meaning that down-peak and interfloor traffic are not considered. This choice was motivated by the availability of analytical formulas for up-peak conditions. Moreover, up-peak traffic is typically the most demanding scenario for elevator systems; hence, if a configuration performs well under up-peak conditions, it is likely to perform adequately under other traffic patterns as well.

The remainder of this thesis is organized into five sections. Section 2 provides background on elevator systems and reviews the literature on elevator zoning and sky lobby configurations. Section 3 introduces metaheuristics and presents the rationale behind the methodological choices in this thesis. Section 4 describes the methodology used in this study. Section 5 presents the results, including the optimal number and placement of sky lobbies, potential core area savings, and the resulting elevator core-to-office space ratios. Finally, Section 6 concludes the thesis by discussing the results, outlining the limitations of the study, and proposing directions for future research.

2 Elevator Systems

This section provides a foundation for understanding the research problem addressed in this thesis. It begins with an overview elevator traffic, including key performance metrics and traffic patterns observed in high-rise buildings. Next, it introduces zoning as a strategy to reduce the space occupied by elevators in a building and reviews approaches for determining optimal zoning arrangements. The section then discusses the concept of sky lobbies, their benefits, and design considerations. Finally, the motivation for this thesis is presented, identifying gaps in existing research and defining the scope of this study.

2.1 Elevator Traffic Fundamentals

In this subsection, the fundamental principles of elevator traffic are described. First, the vertical transportation problem is outlined. Next, common traffic types and patterns observed in buildings are presented. Finally, core equations and performance measures used in elevator research are introduced.

2.1.1 The Vertical Transportation Problem

Barney and Al-Sharif (2015) define the vertical transportation problem as follows: move a specific number of passengers from their origin floors to their respective destination floors, minimizing passenger waiting and traveling time while also minimizing the number of elevators, core space (i.e., the combined volume of elevator shafts and lobby areas), cost, and energy consumption.

In practice, the vertical transportation problem is an optimization problem with multiple objectives and constraints. On the one hand, increasing the number or speed of elevators allows passengers to reach their destination floors faster. However, additional elevators incur higher installation, maintenance, and modernization costs, and they occupy building areas that could otherwise generate rental income. Therefore, solving the vertical transportation problem always involves a compromise between budget constraints, available building space, and the desired level of passenger service (Siikonen, 1997b).

Addressing the vertical transportation problem requires balancing supply and demand, where supply is determined by the number, rated speed, and capacity of elevators, and demand refers to passenger arrivals.

2.1.2 Elevator System Events

In a conventional control system, a passenger arrives at the elevator lobby and places an elevator call by pressing either the “up” or “down” button, depending on the direction of their intended destination floor. This call is referred to as a landing call[1] (Barney & Al-Sharif, 2015). When the elevator arrives, the passenger enters and selects their

[1] Also the terms "hall call" and "corridor call" have been used.

destination floor by pressing the corresponding button inside the elevator car. In a destination control system (DCS), the passenger requests an elevator by specifying their destination floor directly on a device called a destination operation panel (DOP), which is located in the elevator lobby. The elevator system then informs the passenger on the DOP which elevator to board.

In a conventional control system, the call time (CLT) begins when the elevator call is placed and ends when the elevator starts to decelerate for the passenger's destination floor (Sorsa, 2002). Waiting time (WT) is defined as the interval between the passenger's arrival at the elevator lobby and the moment the elevator doors begin opening at the departure floor. Transit time (TT) refers to the interval from when the doors start opening at the departure floor until they start opening at the destination floor. The total time to destination (TTD) is defined as the interval from the passenger's arrival at the elevator lobby until the doors begin opening at their destination floor (Barney & Al-Sharif, 2015). Thus, the TTD is the sum of the waiting time (WT) and TT, expressed as:

$$TTD = WT + TT.$$

Ride time (RD) refers to the interval from when a passenger enters the elevator until they exit (Sorsa, 2002). Journey time (JT) is defined as the duration from the moment a passenger registers a landing call or a destination call until they exit the elevator system (Barney & Al-Sharif, 2015). Thus, JT can be expressed as the sum of WT and RD:

$$JT = WT + RD.$$

Figure 1 illustrates the sequence of passenger events in the elevator system and maps them to corresponding service quality measures defined in the previous paragraphs.

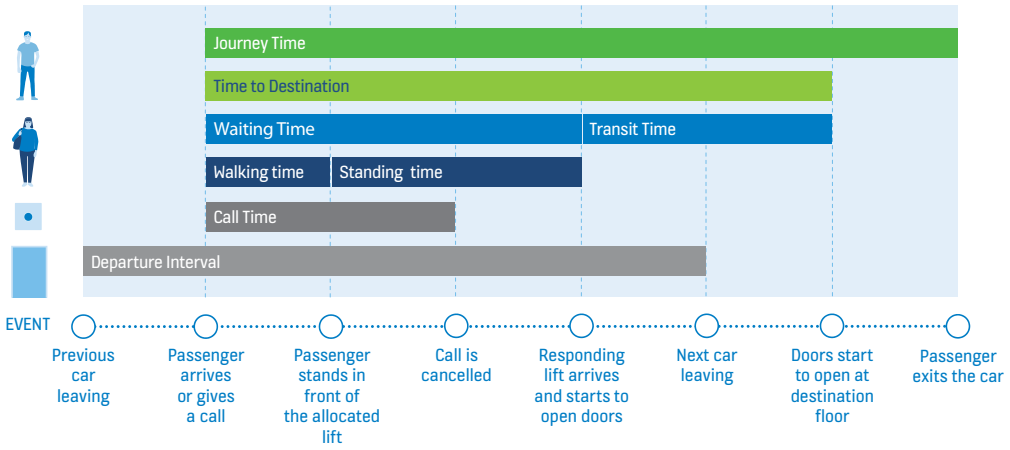


Figure 1: Sequence of passenger events in the elevator system and the corresponding service quality measures. Adapted from Siikonen (2021)

Like many other services, elevator demand can be highly irregular. Peaks in demand can place strain on the system, considerably increasing passenger journey times. The primary difficulty in planning an elevator system lies not in calculating its expected performance, but in accurately estimating the demand (Barney & Al-Sharif, 2015). Therefore, understanding how demand varies across different scenarios is crucial for designing efficient elevator systems. Common traffic types and patterns that characterize these variations are described in the next section.

2.1.3 Traffic Types and Patterns

According to Siikonen (1997a), elevator traffic in office buildings and similar multi-floor buildings can be categorized into three main traffic components: incoming, outgoing, and inter-floor. Each component may vary in intensity throughout the day. Incoming traffic refers to passengers arriving at the building, while outgoing traffic consists of passengers leaving the building. Inter-floor traffic involves passengers traveling between populated floors within the building.

The prevailing traffic condition is often a mixture of these three components. Siikonen (1997a) identified five distinct traffic types based on the relative intensity of each component: upward, downward, two-way, inter-floor, and mixed traffic. Table 1 shows the categorization of traffic types based on these components.

Table 1: Definitions of elevator traffic types based on the distribution of traffic components. The percentages in the Incoming, Outgoing, and Inter-floor columns represent the share of traffic in each respective component. The resulting traffic type classification is shown in the Traffic Type column. Adapted from Viita-aho (2019)

Incoming	Outgoing	Inter-floor	Traffic Type
100%	-	-	Upward
-	100%	-	Downward
50%	50%	-	Two-way
40%	40%	20%	Mixed
25%	25%	50%	Inter-floor

When traffic intensity is considered, distinct traffic patterns emerge. Barney and Al-Sharif (2015) identified four typical daily traffic patterns in office buildings: uppeak traffic, midday (lunchtime) traffic, random interfloor traffic, and downpeak traffic.

An uppeak traffic condition occurs when the dominant (or only) traffic flow is upward, with all or most passengers entering the elevator system at the building's main lobby. Uppeak traffic typically happens in the morning, as passengers arrive at work and seek to reach their workstations on upper floors (Barney & Al-Sharif, 2015).

A midday (lunchtime) traffic condition occurs around midday and features prominent traffic flow to and from specific floors, one of which may be the main lobby (Barney

& Al-Sharif, 2015). Midday traffic is sometimes called two-way traffic, as building occupants typically leave the building via the main lobby for lunch and subsequently return, creating consecutive outgoing and incoming traffic.

Random interfloor traffic occurs when no clear pattern of elevator calls can be identified (Barney & Al-Sharif, 2015). It is the most common pattern and persists for most of the working day in office buildings. Typically, it results from occupants visiting different floors for work-related purposes, such as meetings.

A downpeak traffic condition exists when the dominant or sole traffic flow is downward, with all or most passengers leaving the elevator system at the main lobby. Downpeak traffic occurs at the end of the workday, as occupants exit their offices. Evening downpeak is generally more intense than morning uppeak; however, handling capacity during downpeak is around 50% greater because elevators make fewer stops. Typically, elevators stop at selected floors to pick up departing passengers and then run express to the main lobby (Barney & Al-Sharif, 2015).

Strakosch (1967) presented the typical traffic patterns in office buildings. These patterns are illustrated in Figure 2.

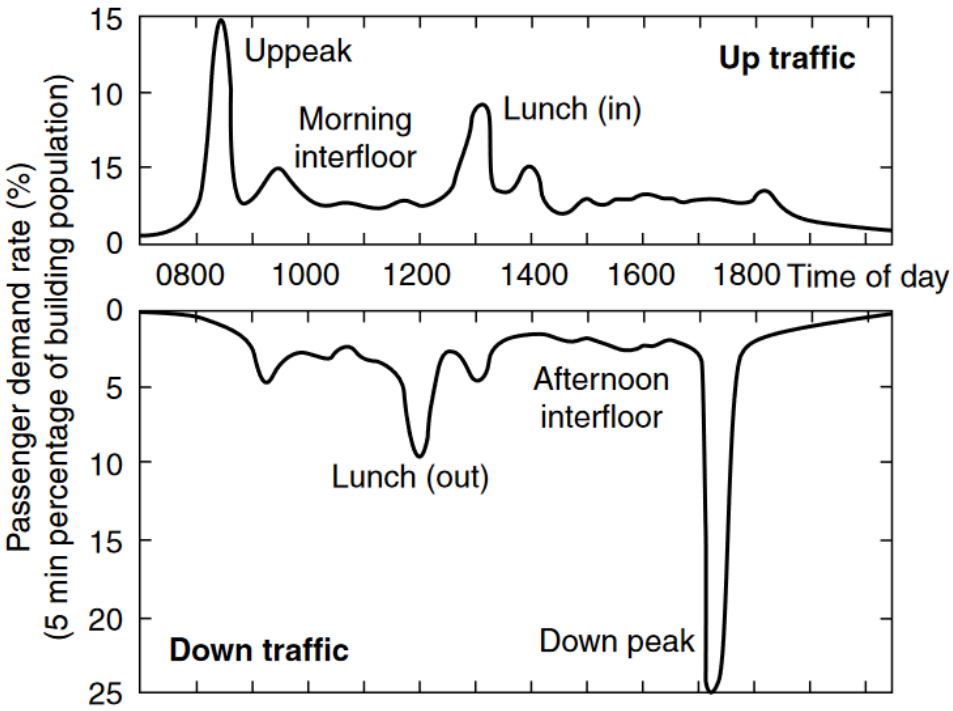


Figure 2: Typical elevator traffic patterns in an office building over the course of a working day. The figure is divided into upward traffic (top) and downward traffic (bottom). The x-axis shows time from 8 am to 8 pm, while the y-axis indicates traffic intensity, measured as the percentage of the building population requiring elevator service within a 5-minute interval. Adapted from Strakosch (1967)

Uppeak traffic occurs prominently in the morning, followed by midday two-way traffic (lunchtime), random interfloor traffic throughout the day, and finally, an intense downpeak at the end of the workday. While these traffic patterns are still present in modern office buildings, their location (in time), shape and magnitude may have shifted and changed slightly due to changes in building usage – such as the addition of restaurants and cafeterias on upper floors, the adoption of flexible working hours and remote work options, and restrictions like indoor smoking bans.

2.1.4 Equations for Uppeak Traffic

Research on elevator performance has primarily focused on uppeak traffic conditions, based on the common assumption that an elevator system capable of handling uppeak traffic can effectively handle all other traffic types as well.

Moreover, uppeak traffic is the only traffic type for which analytical equations exist, as it involves only a single active call per passenger. In contrast, other traffic patterns often involve multiple simultaneous calls, making system performance highly dependent on the specific group control logic and rendering accurate mathematical modeling significantly more difficult.

In this thesis, only uppeak traffic is considered. The corresponding performance equations are derived in the following paragraphs.

Perhaps the most fundamental measure of elevator performance is the round-trip time (RTT). Barney and Al-Sharif (2015) define RTT as the duration (in seconds) of a single elevator car's journey around a building, starting from when the car doors open at the main lobby and ending when the doors reopen at the main lobby after completing the trip. A widely accepted formula for RTT is:

$$RTT = 2Ht_v + (S+1)t_s + 2Pt_p, \tag{1}$$

where H is the expected reversal floor, t_v is the transit time between adjacent floors, S is the expected number of stops during the trip, t_s is the time spent at each stop, P is the expected number of passengers carried, and t_p is the average time required for a single passenger to enter or exit the elevator car.

Interval (INT) is the average time between successive elevator car arrivals at the main lobby floor

$$INT = \frac{RTT}{L}, \tag{2}$$

where L is the number of elevators in the group. (Barney & Al-Sharif, 2015)

The 5-minute handling capacity (HC5) of an elevator group is the number of passengers it can service during uppeak traffic conditions with a car load factor (CLF) of 80%. It is given by

$$HC5 = \frac{300 \cdot CLF \cdot C \cdot L}{RTT}, \tag{3}$$

where C is the elevator car size in persons and L is the number of elevators in the group.

HC5 may also be given relative to the total population in the building (POP) in which case we denote it by %HC5 (Ruokokoski & Siikonen, 2017). The formula reads

$$\%HC5 = \frac{HC5}{POP} \cdot 100\%. \tag{4}$$

Nominal travel time (NTT) is the time period for an elevator to travel from the ground floor to the highest served floor without any stops (Ruokokoski et al., 2018). It is given by

$$NTT = \frac{d \cdot N}{v}, \tag{5}$$

where d is the floor height, N is the floor count, and v is the rated speed of the elevator.

Traditional elevator system design criteria are based on handling capacity, interval, and nominal travel time during uppeak traffic conditions (Ruokokoski & Siikonen, 2017). In residential buildings, handling capacity should exceed 5–7.5% per 5 minutes, while in commercial buildings, it should be 11–13% per 5 minutes (Strakosch, 1984). The average interval should range between 40 and 100 seconds in residential buildings and between 20 and 30 seconds in commercial buildings (Barney & Dos Santos, 1985). Nominal travel time should be less than 50 seconds in residential buildings and less than 32 seconds in commercial buildings (Siikonen, 1997b).

2.1.5 Revised RTT Formulation

There is a shortcoming in the RTT calculation as given by (1) – it does not account for the dependence of acceleration and deceleration times on travel distance, elevator speed, acceleration, and jerk. Moreover, it does not allow for varying passenger numbers between floors. Roschier and Kaakinen (1978) addressed these shortcomings and derived a more accurate formula for calculating RTT. A brief overview of their proposal follows. For a full derivation, please refer to the original publication.

Consider a general building with the main entrance on floor zero. Subsequent floors are $1, 2, \cdots, N$. Let N_k denote the count of passengers on floor $k, 1 \leq k \leq N$. Then the total number of passengers is

$$N_P = \sum_{k=1}^{N} N_k. \tag{6}$$

Let p_k denote the fraction of passengers on floor k so that

$$p_k = \frac{N_k}{N_P}. \tag{7}$$

Clearly,

$$\sum_{k=1}^{N} p_k = 1. \tag{8}$$

Let r denote the travel distance (in floors) from floor i to floor j so that $r = j - i$. The number of initial passengers in the car is P.

When $1 \le r \le N$, the expected number of r-floor runs in upward direction (U_r) and downward direction (D_r) are given by

$$\begin{aligned} U_r = & \sum_{i=1}^{N-(r-1)} \left[\left(1 - \sum_{k=i}^{i+(r-2)} p_k\right)^P - \left(1 - \sum_{k=i}^{i+(r-1)} p_k\right)^P \right] \\ & - \sum_{i=1}^{N-r} \left[\left(1 - \sum_{k=i}^{i+(r-1)} p_k\right)^P - \left(1 - \sum_{k=i}^{i+r} p_k\right)^P \right], \end{aligned} \tag{9}$$

$$D_r = \left(\sum_{k=1}^{r} p_k\right)^P - \left(\sum_{k=1}^{r-1} p_k\right)^P. \tag{10}$$

The expected number of stops in upward direction (S_u) and downward direction (S_d) are formulated as

$$S_u = \sum_{r=1}^{N} U_r, \tag{11}$$

$$S_d = \sum_{r=1}^{N} D_r = 1. \tag{12}$$

Finally, the formula for RTT is

$$RTT = \sum_{r=1}^{N} (U_r + D_r)(t_r + t_d) + 2Pt_p, \tag{13}$$

where U_r and D_r are the expected number of runs with r-floor distance in upward and downward direction, respectively; t_r is the flight time for an r-floor distance; t_d is the additional delay during a single stop, namely the door operating time; P is the expected number of passengers carried; and t_p is the the average time for a single passenger to enter or leave a car.

2.2 Elevator Zoning

2.2.1 The Concept of Zoning

For smaller buildings, elevators are typically assigned to a single group serving each populated floor. As the floor count increases, so does the expected number of stops. As a consequence, the performance of a single group of elevators deteriorates (Barney & Al-Sharif, 2015). One solution is to keep increasing the elevator count until performance targets are met. However, this significantly increases the core area occupied by elevators as each shaft must be built from the ground up.

A more efficient solution is to limit the number of floors each elevator must serve – a strategy known as zoning. In zoning, the building is divided into disjoint sets of contiguous floors, called zones, with each zone typically served by its own dedicated group of elevators. Except for the entrance floor, these zones generally do not overlap (Ruokokoski et al., 2018).

In its simplest form, zoning divides a large lift group into two distinct zones: a low-rise (LR) zone and a high-rise (HR) zone. Figure 3 illustrates this arrangement.

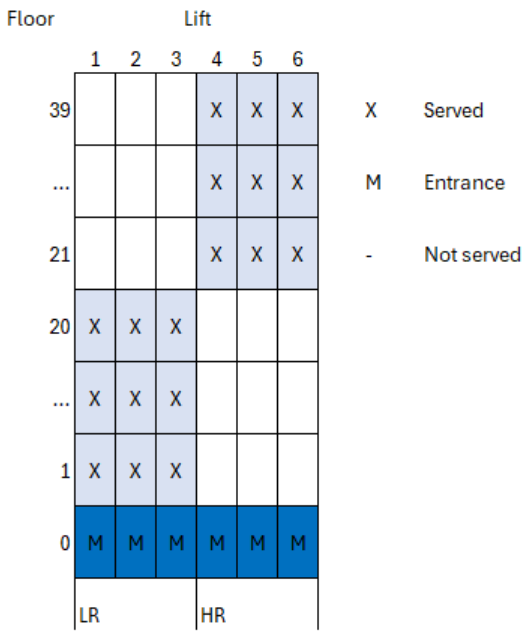


Figure 3: Typical zoning arrangement for a building with 40 floors. The main lobby is located on floor 0, and the highest populated floor is 39. The low-rise (LR) zone covers floors 1 – 20, and the high-rise (HR) zone covers floors 21 – 39. Both zones are served by a dedicated group of three elevators

The LR elevators serve floors immediately above the entrance while the HR elevators performs an express ride past the LR floors and serve the top part of the building. Since shafts of the LR elevators occupy only the lowest part of the building, a significant

portion of the core area is saved. Furthermore, the LR elevators can have a lower rated speed than the HR elevators since the total distance traveled is shorter. This allows smaller machinery, which are less expensive and consume less energy (Ruokokoski et al., 2018).

In a similar manner, a building can be divided into as many zones as needed. Furthermore, the exact floor split of the zones may be tailored. Finding a suitable zoning arrangement is a difficult problem, since the number of zoning arrangements grows exponentially with floor count. For a building with N floors, the number of zoning arrangements is

$$Z = 2^{N-1}. \tag{14}$$

However, there is often an upper limit to the number of zones due to practical reasons. Ruokokoski et al. (2018) showed that if the number of zones is restricted to m, the number of zoning arrangements is

$$Z = \sum_{k=1}^{m} \binom{N-1}{k-1}. \tag{15}$$

Figure 4 plots values of Z from Equation (15) for various choices of m as a function of the floor count N on a logarithmic scale.

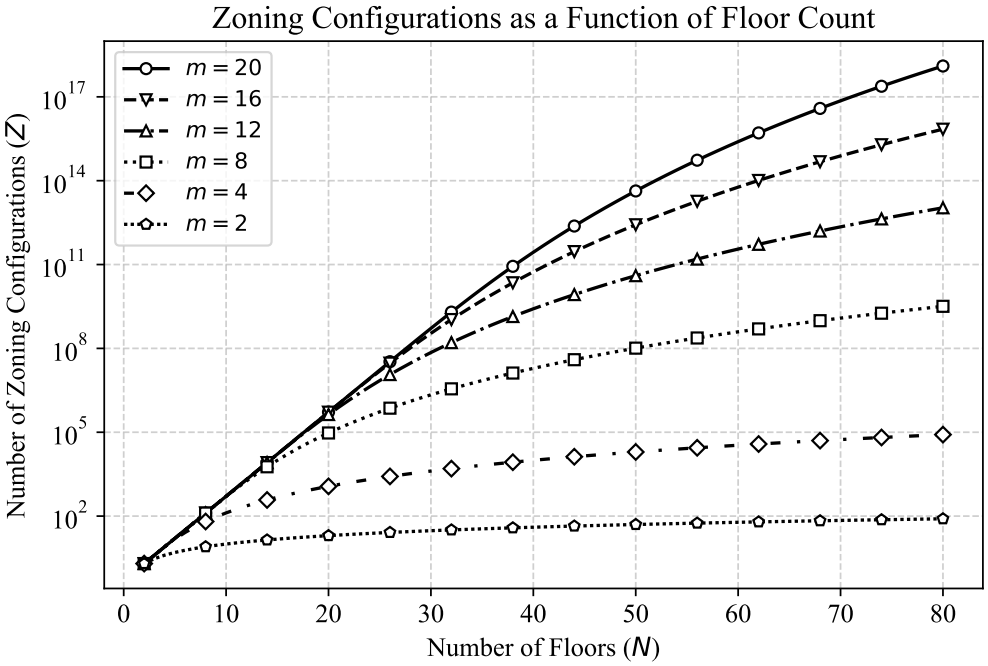


Figure 4: Number of zoning configurations $Z = \sum_{k=1}^{m} \binom{N-1}{k-1}$ for various maximum zone counts m as a function of the floor count N. The plot uses a logarithmic scale on the y-axis for clarity

For example, the number of zoning arrangements is roughly one billion when the floor count is 60 and the maximum zone count is 8.

2.2.2 Review of Zoning Methods

In 1971, Bruce Powell introduced the first optimization method for the static zoning problem using a dynamic programming procedure. Powell's method minimizes a single objective function such as the maximum filling time of a building or the difference in filling times between zones.

Powell's algorithm builds the overall solution recursively by reusing partial solutions from the previous step. In the first step, the algorithm computes all possible one-zone arrangements that serve floors 1 through k, for each $k = 1, \ldots, N$, where N is the total number of floors.

Next, for the two-zone case, the algorithm determines the optimal splitting point x so that the first zone serves floors 1 to $x - 1$ (using the already computed one-zone solutions) and the second zone serves floors x to k (for each $k = 2, \ldots, N$). The objective function value for the two-zone arrangement is then evaluated by combining the precomputed value for the first zone with the new computation for the second zone.

For three zones, the procedure is similar: the algorithm selects an optimal splitting point x such that the third (uppermost) zone serves floors x to k, while the optimal two-zone arrangement (previously computed) takes care of floors 1 to $x - 1$. This process continues – incrementing the number of zones – until the desired maximum number of zones, say m, is reached.

Powell's initial method requires as input the number of elevators for each zone and their maximum velocities. Four years later, Powell extended his earlier work by presenting a method that determines the least amount of elevators needed in each zone to satisfy given performance criteria along with their velocities (Powell, 1975). He also added a maximum waiting time constraint based on interval and introduced four practical constraints:

(C1) All zones must contain an even number of cars.

(C2) Car speeds in any zone must be no slower than the speed of the cars in the next lower zone.

(C3) Car speeds are subject to a minimum speed restriction that is a function of the lowest floor in the zone.

(C4) Passenger handling capacity must meet or exceed a predetermined value (e.g., 12 percent of the zone's population in a 5-minute period).

Even though Powell's method produces an optimal zoning within seconds, it has not been widely adopted in the industry – more than 50 years after Powell's work, Ruokokoski and colleagues noted that the current practice in the lift industry is still

more or less based on rules of thumb, duty table calculations, and the designer's expertise (Ruokokoski et al., 2018).

Ruokokoski et al. (2018) made the following modifications to Powell's method: i) the nominal velocity of elevators is selected based on the highest floor of the zone instead of the lowest floor; ii) the car load factor is a decision variable instead of being a constant fixed to 100% since using fixed car load factors may lead to over- or under-sizing; iii) the number of elevators in a zone should be at minimum and it can differ from values of other zones only by 2 but do not need to be even; and iv) The RTT formula by Roschier and Kaakinen (1978) is used which takes into account the exact running times of each flight during the round trip, instead of using flight time approximations.

Furthermore, Ruokokoski et al. (2018) chose the rated velocity of elevators based on a nominal travel time constraint. That is, the time period for an elevator to travel from the ground floor to the highest floor in the zone without any stops must be shorter than a predetermined value. In addition to maximum filling time of a building, the authors considered two new objective functions: the core area occupied by elevators and the total number number of elevators in the building.

In their work, the optimal zoning $M_f(N)$ with respect to objective function f for a building having N upper floors is obtained by the following dynamic programming recursion:

$$M_f(N) = \min_{2\le n\le 10}\left\{\min_{1\le Z_n\le Z_{\max}}\left[\min_{Z_n\le x\le N} F\left(M_f^{Z_n-1}(x-1), f(x, N, v^*, P^*, L^*)\right)\right]\right\}, \tag{16}$$

where n is the number of elevators, which can vary between n and $n+2$. Z_n is the number of zoning alternatives, and $Z_{\max}$ is the predefined maximum number of zones. v^* is the nominal velocity of the elevators in the zone, P^* is the average number of passengers per elevator, and L^* is the number of elevators in the zone.

The superscript $*$ indicates that the variables are chosen in such a way that they satisfy the given constraints. In the case of maximum filling time, the function $F(\cdot)$ corresponds to the maximum of $M_f^{Z_n-1}(\cdot)$ and $f(\cdot)$, whereas for the objective functions related to core area and the total number of elevators, $F(\cdot)$ represents the sum of $M_f^{Z_n-1}(\cdot)$ and $f(\cdot)$.

Ruokokoski et al. (2018) found that the maximum filling time objective contradicts with both the core area and the total number of elevators objective. The former objective prioritizes solutions where zones have similar filling times and handling capacities while the latter objectives prioritize solutions with minimal elevator shafts. The authors concluded that zoning should thus be considered as a multi-objective optimization problem.

Based on the findings of Ruokokoski et al. (2018), Viita-aho (2019) posed the zoning problem as a multi-objective optimization problem. Viita-aho (2019) considered four

different objective functions but his main focus was on minimizing the price of the elevator configuration and the core area occupied by the elevators.

Instead of using dynamic programming like Ruokokoski et al. (2018) and Powell (1971, 1975), Viita-aho (2019) applied the reference-point-based elitist non-dominated sorting genetic algorithm (NSGA-III) to identify Pareto-optimal zoning solutions. As genetic operators, NSGA-III uses real-valued polynomial mutation and simulated binary crossover.

Viita-aho (2019) approached genetic encoding as follows. To limit the size of the search space of the genetic algortihm, the number of zones, say i, is decided before running the algorithm. Then, during each iteration, the decision variables are:

The upper floors of the zones, given by $\mathbf{K} = [k_1, k_2, \ldots, k_i]$, the elevator speeds $\mathbf{V} = [v_1, v_2, \ldots, v_i]$, the elevator car capacities $\mathbf{C} = [C_1, C_2, \ldots, C_i]$, and the number of elevators $\mathbf{L} = [L_1, L_2, \ldots, L_i]$. The decision variable vector or chromosome $\mathbf{x}$ is then defined as:

$$\mathbf{x} = [\mathbf{K}\ \mathbf{V}\ \mathbf{C}\ \mathbf{L}], \tag{17}$$

where the chromosome length is $n = 4 \cdot i$.

Viita-aho (2019) systematically solved the zoning problem as described in Algorithm 1. In the algorithm, Z denotes the zone count and Z_{max} is the upper limit of zones; L denotes the elevator count and L_{max} is the upper limit of elevators; r represents the run index, and $R_{\max}$ is the total number of runs; t denotes the generation, and $G_{\max}$ is the total number of generations.

Algorithm 1 Pseudocode of the iterative zoning procedure in Viita-aho (2019)

```
procedure ZONING
    for each Z = 1, ..., Z_max do
        for each L = 3, ..., L_max do
            for each r = 1, ..., R_max do
                for each t = 1, ..., G_max do
                    NSGA-III(Z, L, r, t)
                end for
            end for
        end for
    end for
end procedure
```

Viita-aho (2019) found that genetic algorithms are suitable for solving the zoning problem in a multi-objective setting. However, he mentions a few drawbacks. First, the search space is very large, which means that the genetic algorithm might get stuck in a local minimum. Second, if the zone count is a variable, the population size in the genetic algorithm needs to be increased significantly to obtain good results. Lastly, with large floor counts, runtime of the algorithm grows clearly because the algorithm converges slower as the number of possible zoning arrangements increases.

Results also showed that the two main objectives (price and core area) are generally not in conflict. In other words, an elevator configuration with smaller core area was generally also less expensive. The opposite was true when price and maximum filling time were optimized: an arrangement that fills the building fast was more costly. Furthermore, Viita-aho (2019) found that zones are typically smaller in the lower portion of a building and that car sizes and elevator speeds tend to be greater in zones that are located higher in the building.

In addition to optimization methods, some rule-based approaches has been suggested. For instance, Al-Sharif et al. (2016, 2017) proposed the following trigger rule: if the number of elevators exceeds 8 for conventional group control (or 12 for destination control), and if the car capacity exceeds 26 persons (or 2000 kg), then the building should be zoned or the number of zones should be increased if zoning is already in place.

Another rule, addressing excessive transit time, specifies that if the average transit time exceeds 90 seconds, the building should be zoned or have its number of zones increased. A third rule provides recommendations on how to divide the building population when zoning. For two zones, 57% of the population should be assigned to the lower zone and 43% to the upper zone. For three zones, the lower zone should accommodate 43% of the population, the middle zone 30%, and the upper zone 27%. In the case of four zones, the first should contain 29%, the second 27%, and both the third and fourth zones should have 22% each. This distribution is designed to equalize the number of elevators serving each zone and maintain symmetry.

Current industry practices often rely on simple heuristic rules, presumably because they are easier and faster to apply than the presented optimization methods. Despite Powell's optimization approach being available for over fifty years, it has seen limited practical use. Furthermore, existing zoning methods have a significant limitation—they do not explicitly account for elevator lobby areas. In tall buildings with multiple elevator zones, lobby spaces can occupy a considerable area. Consequently, existing methods may suggest zoning arrangements that include too many elevators. Thus, there remains considerable room for improving elevator zoning by developing methods that balance accuracy with simplicity and practicality.

In addition to zoning, an effective way to reduce the footprint of elevators in a building is to use sky lobbies – intermediate transfer points in tall buildings. The next section discusses sky lobbies and their benefits in elevator zoning.

2.3 Sky Lobbies

As the number of floors in a high-rise building increases, an increasing share of the building must be reserved for elevator shafts. At some point, adding more elevators becomes impractical – elevator shafts and machine rooms takes up too much space, making the building inefficient and unprofitable for its owners (Sorsa, 2002). Before the 70s, this problem placed a cap on building size. Around 1970, a new type of solution to the core space problem was introduced: sky lobbies. A sky lobby divides

the building into two smaller building segments called stacks, where the upper stack is served by shuttle elevators transporting passengers to the sky lobby, from where they continue by local elevator groups to their final destination (Al-Sharif, 2017; Schroeder, 1989).

Shuttle elevators typically serve only the main lobby floor and one sky lobby floor, ensuring a fast express ride for the first leg of a passenger's journey. With multiple sky lobbies, shuttle elevators may stop at more than one sky lobby floor. Shuttle elevators can be either single or double deck. In the case of double-deck shuttles, a corresponding double-deck sky lobby is required to accommodate simultaneous passenger unloading from both cars.

There are five types of sky lobby configurations (Barney & Al-Sharif, 2015; Schroeder, 1989):

1. Single-deck shuttles with single-deck local elevators (e.g., the original World Trade Center).
2. Double-deck shuttles with single-deck local elevators (e.g., Sears Tower).
3. Double-deck shuttles with double-deck local elevators (e.g., PETRONAS Towers).
4. Single-deck shuttles with single-deck local elevators operating in a top-down fashion (no known examples).
5. Double-deck shuttles with single-deck local elevators operating in a top-down fashion (e.g., UOB Plaza).

When determining the size of shuttle elevator groups, downpeak conditions play a critical role. Schroeder (1984) explains that elevator groups above the sky lobby are capable of handling 40–50% more traffic during downpeak than during uppeak, as fully loaded cars travel non-stop to the sky lobby. Therefore, to avoid congestion at the sky lobby during downpeak traffic conditions, the capacity of the shuttle elevator group must also be sized 40–50% larger than what an uppeak analysis would indicate. Schroeder (1984) claims that without this additional downpeak capacity requirement, shuttle elevators and sky lobbies would be the most economical configuration for any high-rise building.

The number of sky lobby combinations grows rapidly with floor count, especially when there are multiple sky lobbies. What follows is a derivation of the number of sky lobby configurations as a function of the floor count and the sky lobby count. k sky lobbies effectively split the building into $k + 1$ stacks. Let N denote the total number of floors in the building. A sky lobby may be located on any floor from 1 to $N - 1$ (with floor 0 as the entrance and floor N as the top floor). If there are multiple sky lobbies, they must not be placed on consecutive floors. We now derive the number of valid combinations for sky lobby placements.

Suppose we have a building with N floors numbered $0, 1, 2, \ldots, N$, and sky lobbies may be placed on floors 1 through $N - 1$. If we wish to place m sky lobbies such that

no two are on consecutive floors, let the positions be

$$a_1, a_2, \ldots, a_m \quad \text{with} \quad 1 \le a_1 < a_2 < \cdots < a_m \le N-1,$$

subject to

$$a_{i+1} - a_i \ge 2 \quad \text{for } i = 1, 2, \ldots, m-1.$$

To eliminate the non-adjacency constraint, we introduce shifted variables:

$$b_i = a_i - (i-1), \quad i = 1, 2, \ldots, m.$$

Since $a_{i+1} \ge a_i + 2$, it follows that

$$b_{i+1} = a_{i+1} - i \ge a_i + 2 - i = b_i + 1 > b_i,$$

so the b_i form an increasing sequence without any gap restrictions. Note that:

$$b_1 = a_1 \ge 1, \quad \text{and} \quad b_m = a_m - (m-1) \le (N-1) - (m-1) = N - m.$$

Thus, the b_i are chosen from the set $\{1, 2, \ldots, N-m\}$ with the only condition that

$$b_1 < b_2 < \cdots < b_m.$$

The number of ways to choose m distinct numbers from $\{1, 2, \ldots, N-m\}$ is given by the binomial coefficient

$$\binom{N-m}{m}.$$

Hence, the general formula for the number of valid sky lobby placements is

$$S_m(N) = \binom{N-m}{m}, \tag{18}$$

where N is the floor count and m is the number of sky lobbies.

Figure 5 illustrates the values of $S_m(N)$ for $m = 1, 2, 3, 4$ on a logarithmic scale. For instance, when $N = 100$ and $m = 3$, the number of possible sky lobby configurations is approximately 100,000.

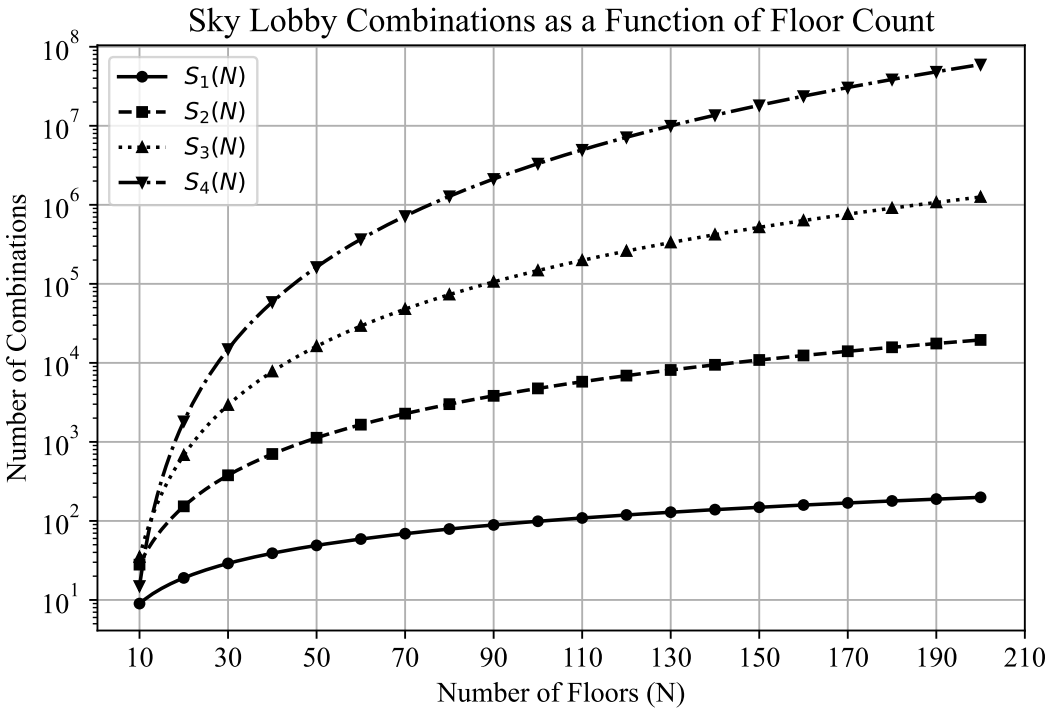


Figure 5: Number of valid sky lobby placements as a function of the total number of floors N. The curves represent: $S_1(N) = N - 1$ for one sky lobby, $S_2(N) = \binom{N-2}{2}$ for two sky lobbies, $S_3(N) = \binom{N-3}{3}$ for three sky lobbies, and $S_4(N) = \binom{N-4}{4}$ for four sky lobbies. A logarithmic y-axis highlights the rapid combinatorial growth as N increases

In the next subsection, a real world elevator system with multiple sky lobbies is presented.

2.3.1 Case Study: World Trade Center

One of the earliest buildings with a sky lobby configuration – the first World Trade Center – was built in 1971. It was 415 m high with 110 floors and 2 sky lobbies on the 44th and 78th floors, dividing the building into three stacks. To further facilitate traffic at the sky lobbies, two-way escalators provided service between the floors immediately above and below the sky lobby floors (Otis Elevator Company, 1967). Figure 6 illustrates the elevator system design, highlighting the three stacks and their corresponding transfer points at the sky lobbies.

The first stack comprised local elevators serving floors below the 44th-floor sky lobby. It included four groups of six single-deck lifts, each serving different floor zones: floors 9 – 16 at 4 m/s, floors 17 – 24 at 5 m/s, floors 25 – 32 at 6 m/s, and floors 33 – 40 at 7 m/s. Passengers traveling beyond the 40th floor used express shuttle elevators operating at 8 m/s to reach the first sky lobby (Barney & Al-Sharif, 2015).

The second stack extended from the 44th-floor sky lobby to the 78th floor. This stack was served by eight express shuttle elevators traveling at 8 m/s. From the sky lobby,

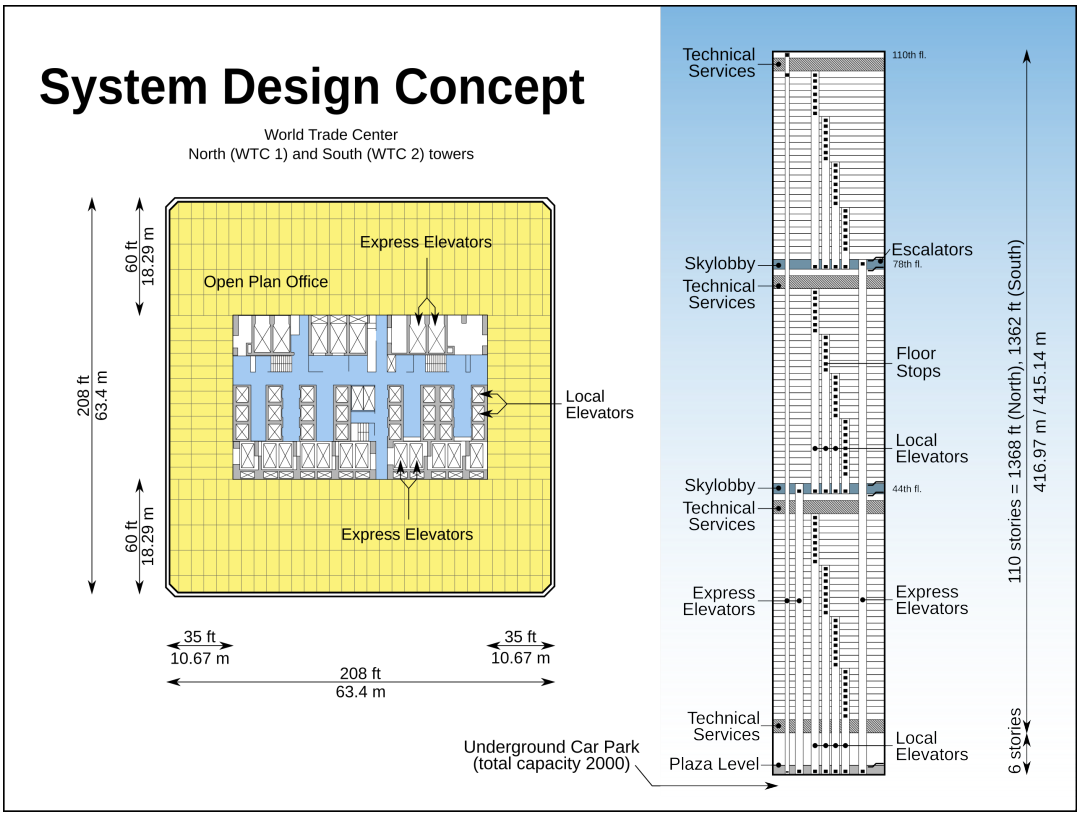


Figure 6: Elevator system layout of the original World Trade Center (1972–2001), illustrating the sky lobbies at the 44th and 78th floors and the resulting three stacks. Express (shuttle) elevators start from the main lobby and bypass intermediate floors to reach the sky lobbies, while local elevators and short escalator connections serve individual zones

four groups of six single-deck local elevators provided access to mid-rise floors: floors 46 – 54 at 2.5 m/s, floors 55 – 61 at 4 m/s, floors 62 – 67 at 4 m/s, and floors 68 – 74 at 5 m/s (Barney & Al-Sharif, 2015).

The third stack served the upper portion of the building, beginning at the 78th-floor sky lobby. Eight express shuttle elevators transported passengers from the ground level to this sky lobby at 8 m/s, where they transferred to local elevators. This stack included four groups of six single-deck lifts serving floors 80 – 86 at 2.5 m/s, floors 87 – 93 at 4 m/s, floors 94 – 99 at 4 m/s, and floors 100 – 107 at 5 m/s (Barney & Al-Sharif, 2015).

According to Otis – the elevator manufacturer – one of the most important factors that made the project economically feasible was incorporating sky lobbies. (Otis Elevator Company, 1967).

2.3.2 Advantages and Disadvantages

The main advantage of introducing sky lobbies is reduced core space occupied by elevators. Due to the effect of splitting the building into smaller stacks, a sky lobby permits the local elevator zones to be placed on top of one another with the effect of adding more than one elevator to each slot in the elevator shaft. With double-deck elevators, a total of four cars can operate in the same shaft. Sharing a single shaft between multiple elevators significantly reduces the core area occupied by elevators (Fortune, 1995).

Another advantage of introducing sky lobbies is reduced cost. Even though the use of a sky lobby and shuttle elevators increases the total number of elevators, the installation cost can be lower, because local elevators in the upper stacks no longer need to take an extensive express ride over the lower stacks and may thus have lower speeds (Schroeder, 1984). The only elevators that are required to have high speeds are the shuttle elevators transporting passengers between main lobby and sky lobby floors.

The main drawback of introducing sky lobbies is that passengers traveling to floors in the upper stacks must take two separate elevator trips: an express ride to the sky lobby using a shuttle elevator, and then to the destination floor using a local elevator.

Another drawback of introducing sky lobbies is the creation of dead floors – floors just below the sky lobby that cannot be served by either the low-rise or high-rise elevator groups. This issue arises because the machine room for the high-rise elevators is typically located directly beneath the pit of the low-rise elevators, leading to two or three floors that lack direct elevator access (Schroeder, 1989).[2] However, these floors are not entirely wasted; their high location makes them well-suited for essential building infrastructure, such as heating, ventilation, and electrical systems. Alternatively, if connected to the sky lobby via separate elevators or escalators, they can function as premium office spaces, benefiting from direct access to shuttle elevators and a quieter environment compared to other tenant floors. The sky lobby itself also presents opportunities for commercial use, such as restaurants, shops, or lounges (Schroeder, 1989).

2.3.3 Review of Existing Studies

Schroeder (1985) analyzed the benefits of using one versus two sky lobbies in a study of 14 buildings, each with up to 214 floors and 100 occupants per floor. Surprisingly, the results indicated that incorporating two sky lobbies increases space requirements, suggesting that a single sky lobby may be the more efficient solution. Instead of increasing the sky lobby count, Schroeder (1985) envisioned quadruple-deck elevators, though he acknowledged their technical infeasibility at the time. Notably, even today, quadruple-deck elevators have not been implemented in practice, whereas buildings with up to five sky lobbies, such as the Shanghai Tower, have been constructed.

[2] This issue can be avoided with machine room-less (MRL) elevator systems, first introduced by KONE in 1996, which eliminate the need for a dedicated machine room.

Schroeder (1989) showed that introducing sky lobbies becomes advantageous when the floor count reaches 40 or more. Schroeder (1989) analyzed how various common sky lobby configurations compare to a zoned configuration without sky lobbies in a building with 40 populated floors and 100 passengers per floor. The study assessed the core area and cost implications of four main sky lobby configurations: (A) single-deck shuttles with single-deck local elevators, (B) double-deck shuttles with single-deck local elevators, (C) single-deck shuttles with single-deck local elevators utilizing a "top-down" elevatoring strategy, and (D) double-deck shuttles with double-deck local elevators.

Results showed that solution A with 1-deck elevators reduced the core area by 28% and came at a 23% lower real cost after adjusting for savings in operational expenses, including energy savings, maintenance savings, and space savings as rental income [3]. Solution B with 2-deck shuttles reduced the core area by 33% and came at a 39% lower real cost. Solution D with both shuttles and local being 2-deck reduced core area 51% and comes at a 139% lower real cost. Solution C with top/down elevatoring reduced the core area only by 14% and came at a 62% higher real cost and was thus deemed unattractive. Even though solutions B and D seem more attractive based on the numbers, Schroeder (1989) notes that these configurations are infeasible in practice for the considered building. The reason is that if a shutdown were to occur, handling capacity and interval would deteriorate too drastically due to smaller elevator groups in those configurations. However, for larger buildings, such configurations become both feasible and attractive.

In a recent study, Siikonen (2024) demonstrated that a sky lobby can significantly reduce the core space of elevators in a building. In the paper, Siikonen (2024) used simulation to assess the core space demand of various elevator configurations in a 312-meter-high building with 78 floors and an estimated population of 10,920 people.

The six elevator configurations considered in the study fall into two main categories: zoned building arrangements, where all elevator groups start from the ground, and sky-lobby arrangements, where a shuttle group serves as an intermediary. The zoned building arrangement consists of seven elevator groups starting from the ground, with four different configurations: (1) single elevators with a conventional control system using up and down call buttons, (2) single elevators with a destination control system, (3) double-deck elevators with a conventional control system using up and down call buttons, and (4) double-deck elevators with a destination control system.

In contrast, the sky-lobby arrangement, which includes a shuttle group and six local elevator groups, features two configurations: (5) double-deck elevators with a destination control system for all elevator groups, and (6) double-deck elevators with a destination control system for the local elevator groups while implementing a multi-car solution for the shuttle group, utilizing three circulating systems with five cabins each.

[3] Operational expenses were capitalized at 10%

In the zoned arrangement, the double-deck destination control system (4) required only 60% of the core area compared to the conventional single-deck system (1). However, the sky-lobby configurations (5 and 6) demonstrated even greater space efficiency. Results showed that the sky-lobby arrangement utilizing double-deck elevators with a destination control system (5) reduced the core area to 46% of the conventional zoned system. Notably, the sky-lobby solution incorporating a multi-car shuttle system (6) achieved the most significant reduction, requiring only 37% of the core area of the conventional single-deck arrangement without a sky lobby (1).

Overall, the existing results are inconclusive regarding when sky lobbies should be used, on which floors they should be located, and what their core area savings potential is. Prior studies lack uniform modeling assumptions and typically cover only a few arbitrarily chosen instances, making it difficult to identify patterns or generalize findings. One possible reason for this is that the question of optimal sky lobby placement is mathematically complex and lacks a formal derivation. This combinatorial optimization problem involves an extremely large search space, as it constitutes a nested version of Equations (18) and (15). Problems of this scale are generally unsolvable using exact methods. Instead, approximate methods are employed to determine near-optimal solutions within practical runtimes. Metaheuristics represent an important class of these approximate methods and will be the focus of the next section.

3 Metaheuristics for Combinatorial Optimization

The sky lobby placement problem exemplifies a large-scale *combinatorial optimization* problem, where the search space is so extensive that exact methods become intractable. In such scenarios, *metaheuristics* provide a powerful framework for generating high-quality solutions within reasonable computation times. These algorithms have proven effective across numerous complex domains – from scheduling and routing to facility layout – by balancing exploration of a vast search space with exploitation of promising solution regions.

In the sections that follow, we first define combinatorial optimization more formally, then provide an overview of the principal metaheuristic paradigms, and finally highlight how hybrid methods can further enhance performance. This foundation sets the stage for developing an algorithmic approach capable of tackling the optimal zoning and sky lobby placement problem posed in the preceding section.

3.1 Combinatorial Optimization Problems

Combinatorial optimization problems seek to find an optimal solution from a finite (or countably infinite) set of discrete possibilities. Such problems often involve binary or integer variables, but the requirement is not strictly that all variables must be integer-valued – rather, that the set of feasible solutions itself is discrete.

Perhaps the most well known COP is the Travelling Salesman Problem (TSP). Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? Both the zoning problem of finding the optimal zoning arrangement of elevators and the sky lobby problem of finding the optimal count of sky lobbies and their floors are COPs.

In COPs, we are looking for an object in a finite – or potentially a countably infinite – set. This object is usually in the form of an integer number, a permutation, a subset or a graph (Blum & Roli, 2003).

According to Blum and Roli (2003), a combinatorial optimization problem $P = (S, f)$ is defined by:

- a set of variables $X = \{x_1, \ldots, x_n\}$;
- variable domains $D_1, \ldots, D_n$;
- constraints among variables;
- an objective function f to be minimized, where

$$f : D_1 \times \cdots \times D_n \to \mathbb{R}^+.$$

The set of all possible feasible assignments is given by:

$$S = \{s = \{(x_1, v_1), \ldots, (x_n, v_n)\} \mid v_i \in D_i, s \text{ satisfies all the constraints}\}.$$

Here, (x_i, v_i) indicates that variable x_i is assigned the value v_i. Blum and Roli (2003) use set-of-pairs notation to keep track of which value belongs to which variable. An equivalent way would be to write solutions as vectors $(v_1, \ldots, v_n)$.

S is typically called the solution (or search) space, since each solution $s \in S$ is a candidate solution to the problem. To solve the problem, we find a solution $s^* \in S$ with minimum objective function value, i.e., $f(s^*) \leq f(s)\ \forall s \in S$. The solution s^* is called the globally optimal solution of P, and the set of globally optimal solutions is the subset $S^* \subseteq S$.

Since COPs often arise in practice, they have been studied extensively. Algorithms for solving COPs can be categorized into complete and approximate methods (Blum & Roli, 2003). Complete methods find an optimal solution within bounded time for every finite-size instance. In contrast, approximate methods sacrifice the guarantee of finding optimal solutions in order to obtain sufficiently good solutions within a significantly shorter time frame.

Many COPs have been shown to be $\mathcal{NP}$-hard. Even without $\mathcal{NP}$-hardness, combinatorial explosion makes complete methods infeasible in practice for many COPs. Siikonen (1997b) provides an example with the elevator dispatching problem: "In a building with an eight-car group there can be nearly 60 hall calls when all the up and down calls are taken into account. The total number of possible route combinations to be calculated are more than $6.2 \cdot 10^{57}$". Therefore, a complete method would only be feasible for low buildings with 2-3 elevators in the group.

Due to the practical limitations of complete methods, approximate methods for solving COPs have received much attention from researchers. Since the 1980s, metaheuristics have emerged as a powerful and versatile class of methods, providing general-purpose frameworks for solving complex optimization problems. These methods will be covered in more detail in the next section.

3.2 Definition and Characteristics of Metaheuristics

The term metaheuristics was first introduced by Glover (1986). The term derives from the composition of two Greek words. Heuristic derives from the verb heuriskein, which means “to find”, while the suffix meta means “beyond, in an upper level” (Blum & Roli, 2003). The name reflects the nature of metaheuristics as a higher-level procedure for guiding the search process of an optimal solution to some problem.

Lodi et al. (1999) define metaheuristics as follows: "A metaheuristic is an iterative master process that guides and modifies the operations of subordinate heuristics to efficiently produce high-quality solutions. It may manipulate a complete (or incomplete) single solution or a collection of solutions at each iteration. The subordinate heuristics may be high (or low) level procedures, or a simple local search, or just a construction method.".

Of fundamental importance when using metaheuristics is to dynamically balance between diversification (exploration) and intensification (exploitation) (Blum & Roli,

2003). The aim is to quickly identify regions in the search space with high quality solutions and to avoid spending time in regions which are either already explored or which do not contain high quality solutions. In exploration, unexplored regions are visited to ensure that all regions of the search space are more evenly explored and that the search is not confined to only few regions. During exploitation, promising regions are explored more thoroughly in the hope to find better solutions (Talbi, 2009).

Talbi (2009) suggests classifying metaheuristics to single-solution based and population-based metaheuristics.[4] Population-based metaheuristics place greater emphasis on exploration through the concurrent management and evolution of multiple candidate solutions, which promotes broader search space coverage. Genetic algorithms exemplify population-based methods. Conversely, single-solution-based methods place more emphasis on exploitation by iteratively improving a single solution with an intensive local search. Simulated annealing is a representative example of single-solution methods.

Figure 7 shows a classification diagram for common algorithmic strategies used in combinatorial optimization.

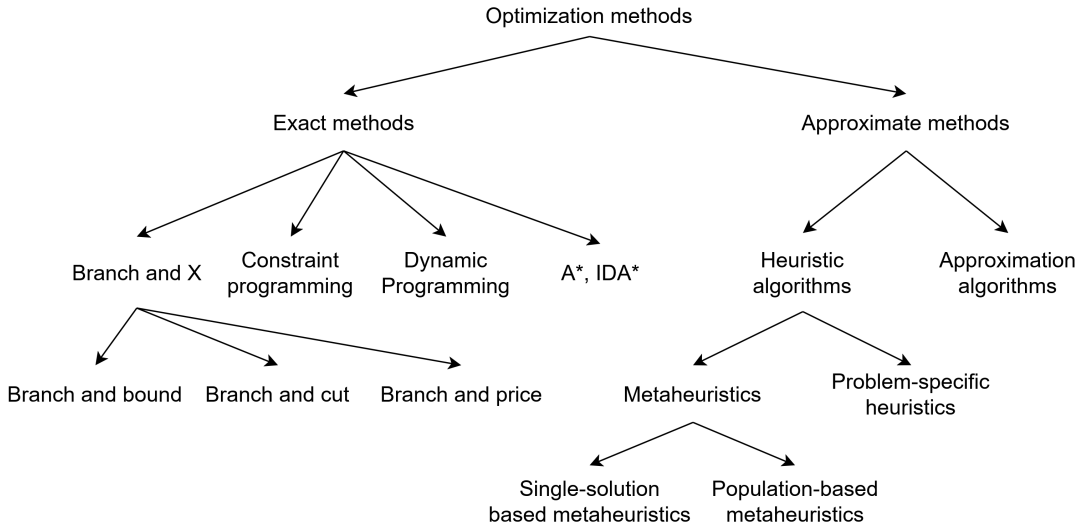


Figure 7: Classification of combinatorial optimization methods. This diagram focuses on algorithmic strategies primarily used for solving discrete optimization problems, such as exact methods (e.g., branch and bound) and approximate methods including heuristics and metaheuristics. A* is a graph traversal and path searching algorithm and IDA* is one of its variants (Iterative Deepening A*). Adapted from Talbi (2009)

At the top level, these methods are divided into exact methods, which guarantee optimal solutions (e.g., branch and bound), and approximate methods, which aim for near-optimal solutions with reduced computational effort. Among the approximate

[4]Other classification methods include inspiration (nature-inspired vs. non-nature-inspired), memory usage (memory vs. memoryless), determinism (deterministic vs. stochastic), and search strategy (iterative vs. greedy).

methods, we distinguish between heuristic algorithms and approximation algorithms. A key subset of heuristics is metaheuristics, which can be further classified into single-solution based and population-based approaches, as discussed earlier.

3.3 Single-solution based Metaheuristics

As the name suggests, single-solution based metaheuristics (S-metaheuristics) focus on iteratively improving a single solution. Talbi (2009) describes these methods as "walks" through the search space, where each step moves from the current solution to a neighboring one.

Each move in an S-metaheuristic consists of two phases: the generation phase and the replacement phase (Talbi, 2009). In the generation phase, a set of neighbors $N(s)$ is generated from the current solution s, typically by applying a move operator m that performs a small perturbation to the solution s. A solution $s' \in N(s)$ is called a neighbor of s. In the replacement phase, a new solution $s' \in N(s)$ is selected to replace s. This process is repeated until a termination criterion is met.

Algorithm 2 presents a high-level template for S-metaheuristics. The functions `StoppingCriteriaMet()`, `GenerateNeighbors()`, and `SelectNeighbor()` correspond to checking the stopping condition, generating the set of neighbors, and selecting a neighbor from this set, respectively.

Algorithm 2 High-level template for S-metaheuristics

1: **Input:** Initial solution s_0
2: $t \leftarrow 0$
3: **while not** StoppingCriteriaMet() **do**
4: $\quad N(s_t) \leftarrow \text{GenerateNeighbors}(s_t)$
5: $\quad s_{t+1} \leftarrow \text{SelectNeighbor}(N(s_t))$
6: $\quad t \leftarrow t + 1$
7: **end while**
8: **Output:** Best solution found

There are two fundamental design choices shared by all S-metaheuristics: the definition of the neighborhood function and the choice of initial solution. Without a well-defined neighborhood structure, S-metaheuristics are unlikely to succeed (Talbi, 2009).

The neighborhood definition depends strongly on the chosen representation for the problem at hand. With a binary string representation, the neighborhood of a solution may consist of strings, where one bit of the solution is flipped. For permutation problems like the TSP, the neighborhood of a solution could consist of permutations that result from swapping the positions of two cities in the current tour. This is known as a swap neighborhood, and it contains all solutions that can be reached by exchanging the order of any two cities in the current permutation.

There typically is a tradeoff between the size (or diameter) and the quality of the neighborhood to use and the computational complexity to explore it (Talbi, 2009).

Large neighborhoods may improve the quality of the obtained solutions since more neighbors are considered at each iteration. However, this comes at a computational cost of generating and evaluating a large neighborhood.

The main property that is needed from a neighborhood is locality. Locality refers to the effect in the solution when performing a move in the neighborhood. When small moves are made, the solution should not change much. This is referred to as strong locality. Weak locality means that small moves result in large changes in the solution in which case the search converges to a random search (Talbi, 2009).

In addition to defining a suitable neighborhood, it is important to choose an appropriate initial solution from which the trajectory int hte search space starts. According to Talbi (2009), the two main strategies for choosing an initial solution are a random approach and a greedy approach. A third strategy, not explicitly considered by Talbi, is to leverage domain knowledge to construct an initial solution that is likely close to the optimum.

To illustrate these strategies, consider the traveling salesman problem (TSP). In a random approach, an initial TSP solution is generated by constructing a random tour through all cities, without regard to total distance. A greedy approach might use the nearest-neighbor heuristic, which begins at a randomly chosen city and iteratively selects the closest unvisited city until the tour is complete. A domain knowledge-based approach could involve consulting an experienced traveler who has insight into efficient city sequences based on practical experience.

Regarding the strategy of choosing the initial solution, there is a tradeoff between quality of solutions and computational time (Talbi, 2009). The best strategy will depend on the efficiency of the two approaches and the properties of the S-metaheuristic. For example, a random approach may be justified for a larger neighborhood since it is less sensitive to the initial solution. However, even though producing a random solution may be faster than using a greedy approach, the metaheuristic may require significantly more iterations to converge if the random solution is from a poor region of the search space.

Most well known S-metaheuristics include local search, simulated annealing, and tabu search. Among the various S-metaheuristics, simulated annealing stands out due to its ability to probabilistically accept worsening solutions, making it particularly effective for escaping local optima.

3.3.1 Simulated Annealing

Metropolis et al. (1953) developed the Metropolis algorithm to simulate the behavior of a physical system in thermal equilibrium at a fixed temperature. The algorithm is based on generating a sequence of states of the solid with Monte Carlo methods.

The Metropolis algorithm proceeds as follows. Starting from an initial state i with energy E_i, a new state j with energy E_j is generated by modifying the position of one particle. In each step, the energy difference $\Delta E = E_j - E_i$ is computed. If ΔE is

negative, the new state j has lower energy and it becomes the current state i. If ΔE is positive, the probability that state j becomes the current state i is given by

$$\mathbb{P}(i \leftarrow j) = \exp\left(-\frac{\Delta E}{k_b \cdot T}\right), \tag{19}$$

where T represents the fixed temperature of the solid and $k_b = 1.38 \cdot 10^{-23} J/K$ is the Boltzmann constant. The acceptance criterion for the new state j in Equation (19) is called the Metropolis criterion.

The Metropolis algorithm simulates thermal equilibrium at a given temperature by generating a large number of transitions. Over time, the distribution of states converges to the Boltzmann distribution, which gives the probability that the system occupies state i of energy E_i at temperature T (Gendreau & Potvin, 2019).

Algorithm 3 shows a pseudo code of the Metropolis algorithm, where U denotes the uniform distribution.

Algorithm 3 The Metropolis Algorithm

Input: Initial state i, energy function $E(i)$, temperature T, number of iterations N
Compute initial energy $E_i = E(i)$
for $t = 1$ to N **do**
    Generate a new candidate state j by perturbing state i
    Compute energy $E_j = E(j)$
    Compute energy difference $\Delta E = E_j - E_i$
    if $\Delta E < 0$ **then**
        Accept new state: $i \leftarrow j$
    else
        Compute acceptance probability $P = \exp(-\Delta E/(k_B T))$
        Draw a random number $r \sim U(0, 1)$
        if $r < P$ **then**
            Accept new state: $i \leftarrow j$
        end if
    end if
end for
Output: Final state i

In the 1980s, a new powerful method to solve combinatorial optimization problems emerged (Černý, 1985; Kirkpatrick et al., 1983). This method was called Simulated Annealing (SA) and it is rooted on the Metropolis Algorithm. In SA, the Metropolis acceptance criterion is applied repeatedly to generate a search trajectory through the solution space of the optimization problem. The fundamental idea in SA is to allow moves resulting in solutions of worse quality than the current solution (uphill moves) in order to escape from local minima (Blum & Roli, 2003)

SA draws its power from a close analogy with the physical annealing process in metallurgy. In the physical process, a material is heated and then gradually cooled

to remove defects and reach a stable low-energy crystalline structure. In the context of optimization, the goal is to find a solution with the lowest objective value. This analogy is formalized in Table 2, where each concept from physics maps naturally to its counterpart in optimization: the state of the material corresponds to a candidate solution, atomic positions to decision variables, and energy to the objective function value. The ground state, which minimizes energy, aligns with the global optimum, while metastable states represent local optima. The temperature controls the randomness of state transitions in both cases – high temperatures allow more exploration, while low temperatures focus the search around good solutions.

Table 2: Analogy Between the Physical System and the Optimization Problem in the context of SA. Adapted from Talbi (2009)

Physical System	**Optimization Problem**
System state	Solution
Molecular positions	Decision variables
Energy	Objective function
Ground state	Global optimal solution
Metastable state	Local optimum
Rapid quenching	Local search
Temperature	Control parameter T

In the Metropolis algorithm, a system evolves by transitioning between states with different energy levels, accepting new states probabilistically based on the Metropolis criterion. SA generalizes this concept by interpreting energy as an objective function and allowing controlled exploration of suboptimal solutions through a control parameter T for the temperature.

The algorithm starts with an initial solution and iteratively explores random neighboring solutions. At each iteration, a candidate solution s' is generated, and the change in the objective function is computed as

$$\Delta E = f(s') - f(s).$$

The probability of accepting s' is given by

$$\mathbb{P}\{\text{accept } s'\} = \begin{cases} 1, & \text{if } \Delta E \leq 0, \\ \exp\left(-\frac{\Delta E}{T}\right), & \text{if } \Delta E > 0. \end{cases}$$

This acceptance probability, derived from the Boltzmann distribution, allows the algorithm to escape local minima by permitting occasional transitions to worse

solutions. The temperature T controls this randomness; at high T, the search is more exploratory, while at low T, the algorithm behaves more greedily.

Over time, the temperature decreases according to a cooling schedule $g(T)$, progressively reducing the likelihood of accepting worse solutions and guiding the search toward convergence. Each temperature level follows an equilibration phase[5], where multiple solution transitions occur before cooling further (Talbi, 2009). This structure ensures a gradual transition from exploration to exploitation.

Having introduced the Metropolis acceptance criterion and the concept of gradually lowering the temperature T to guide the search from exploration to exploitation, we now present the complete SA procedure in Algorithm 4.

The procedure initializes a solution s and an initial temperature $T_{\max}$. For each temperature level, multiple iterations of the Metropolis acceptance criterion are performed to approximate equilibrium, with the corresponding iteration count denoted by `EquilibriumIterations()`. The temperature is then updated according to the cooling schedule $g(T)$, gradually reducing the probability of accepting non-improving moves. This process repeats until the stopping condition, checked by `StoppingCriteriaMet()`, is satisfied. Finally, the best solution s^* is returned.

Algorithm 4 The simulated annealing algorithm

Input: Initial solution s_0, initial temperature $T_{\max}$, cooling schedule $g(T)$
$s \leftarrow s_0$
$T \leftarrow T_{\max}$
while not StoppingCriteriaMet() **do**
    for $i = 1$ to EquilibriumIterations(T) **do**
        Generate a random neighbor s'
        $\Delta E \leftarrow f(s') - f(s)$
        if $\Delta E \leq 0$ **then**
            $s \leftarrow s'$ // Better solution
        else
            $P \leftarrow \exp(-\Delta E/T)$
            Draw $r \sim U(0, 1)$
            if $r < P$ **then**
                $s \leftarrow s'$ // Accept worse solution with probability P
            end if
        end if
    end for
    $T \leftarrow g(T)$
end while
Output: Best solution found s^*

Along with the neighborhood definition, the most critical for SA performance is the

[5]The number of transitions per temperature level, often called the equilibration phase, is typically set empirically or based on heuristic tuning.

cooling schedule, which refers to the controlled scheme to decay the temperature. What follows is a derivation showing that with certain choices, SA converges to the global optimum. Moreover, most common cooling schedules are presented.

The dynamic process of moving from one candidate solution to the next during SA is a Markov chain, since the successor solution depends only on the current solution. Aarts et al. (1997) showed that under certain conditions on the cooling schedule, the algorithm converges in probability to the global optimum. More precisely, there exists a constant $\Gamma \in \mathbb{R}$ such that the probability of finding the global minimum approaches 1 as the number of iterations k tends to infinity, i.e.,

$$\lim_{k \to \infty} P(\text{global minimum found after } k \text{ steps}) = 1,$$

if and only if the sum of the exponential terms related to the cooling schedule diverges, that is,

$$\sum_{k=1}^{\infty} \exp\left(\frac{\Gamma}{T_k}\right) = \infty,$$

where T_k represents the temperature at the k-th iteration.

However, cooling schedules which guarantee convergence to a global optimum are often too slow for practical applications. Therefore, faster cooling schedules are commonly used (Blum & Roli, 2003). Perhaps the most prevalent is the geometric (or exponential) cooling schedule, defined by:

$$T_{k+1} = \alpha T_k,$$

where $\alpha \in (0, 1)$ is a constant, resulting in:

$$T_k = T_0 \alpha^k,$$

where T_0 is the initial temperature. Other common cooling schedules include linear cooling, $T_k = T_0 - \beta k$ for some constant $\beta > 0$, polynomial cooling, $T_k = \frac{T_0}{(1+k)^\gamma}$ for some constant $\gamma > 0$, and logarithmic cooling, $T_k = \frac{T_0}{\log(1+k)}$.

The initial temperature and cooling schedule must be adapted for the problem instance at hand since the cost of escaping a local minima depends on the search landscape (Blum & Roli, 2003). A simple method to inform the decision of an appropriate initial temperature is to sample the search space with a random walk and record statistics (e.g., mean and standard deviation) of the objective function values. Overall, due to the sheer number of combinations, it is impossible to find the optimal hyperparametrization for each problem instance in SA.

3.3.2 Applications of Simulated Annealing

SA has been successfully applied in various $\mathcal{NP}$-hard combinatorial optimization problems. For example, Connolly (1990) used SA to solve the Quadratic Assignment Problem (QAP). The QAP involves assigning items to locations so that the total

cost from their pairwise interactions and distances is minimized. The authors found improved solutions for several of the largest problems available in the literature at the time of publication using only modest CPU time. Furthermore, the proposed scheme did not require retuning for different problem instances. In sum, Connolly (1990) were able to develop one of the most effective heuristics for solving the QAP using SA.

Another successful application was in Laarhoven et al. (1992). The authors used SA to solve the Job-shop Scheduling Problem (JSP) – one of the most difficult CO problems there is. JSP concerns scheduling a set of jobs, each with a specific sequence of tasks on different machines, to minimize the total completion time while ensuring no machine handles more than one task at a time. Their SA scheme was able to find better solutions for the largest instances than well known heuristics tailored specifically for the problem, namely Matsuo et al. (1989) and Lasserre (1992). In cases where SA outperformed the tailored heuristics, it came at a computational cost. However, as noted in Laarhoven et al. (1992), this cost is easily compensated by the the ease of implementation and the high quality of solutions.

In both studies, great care was taken to ensure that the neighborhood structure and the selection of neighbors were proper and that the cooling schedule was optimized. Connolly (1990) ran a large number of test runs on two different methods to choose neighbors from the neighborhood: a random and sequential approach to conclude that the latter performed better for this class of problems. Laarhoven et al. (1992) reported that a slow cooling schedule (slow decrement of the control parameter) significantly improved the quality of the average best solution returned by the algorithm. To conclude, these studies demonstrate that with a carefully designed neighborhood structure and an optimized cooling schedule, SA is an excellent method to solve very difficult CO problems.

3.4 Population-based Metaheuristics

Whereas S-metaheuristics focus on improving a single solution, population-based metaheuristics (P-metaheuristics) iteratively improves a population of solutions. P-metaheuristics start from an initial population of solutions. Until a stopping criteria is met, an iterative process of generating a new population of solutions and replacing the current population of solutions follows. In the replacement phase, a selection is carried out from the current and the new population. Examples of P-metaheuristics are evolutionary algorithms (EAs), particle swarm optimization (PSO), and artificial immune systems (AISs). Most P-metaheuristics are inspired by nature (Talbi, 2009)

Algorithm 5 shows the general framework for P-metaheuristics. The functions `StoppingCriteriaMet()`, `GenerateNewPopulation()`, and `SelectNewPopulation()` correspond to checking the stopping condition, generating the new population, and selecting a new population, respectively.

Accroding to Talbi (2009), one way of categorizing P-metaheuristics is the way in

Algorithm 5 High-level template for P-metaheuristics

1: $P \leftarrow P_0$ // Initialize population
2: $t \leftarrow 0$
3: **while not** StoppingCriteriaMet() **do**
4: $\quad P'_t \leftarrow \text{GenerateNewPopulation}(P_t)$
5: $\quad P_{t+1} \leftarrow \text{SelectNewPopulation}(P_t \cup P'_t)$
6: $\quad t \leftarrow t + 1$
7: **end while**
8: **Output:** Best solution(s) found

which a new population of solutions is generated. In evolution based P-metaheuristics, solutions of the population are selected and reproduced through variation operations such as mutation and recombination. Variation operations act directly on the representations of the solutions. An example of this type of P-metaheuristic is EAs. In blackboard-based P-metaheuristics, solutions of the population participate in the construction of a shared memory, which will guide the generation of new solutions. Compared to the first class, recombination between solutions is indirect. Bee colony is an example of the latter class of P-metaheuristics. In this thesis, the focus will be on the first class of P-metaheuristics, namely a specific type of EAs called genetic algorithms.

3.4.1 Genetic Algorithms

Holland (1992) developed the Genetic Algorithm (GA) through an effort to computationally model the biological evolution of species. Most organisms evolve by two main processes: natural selection and sexual reproduction. Natural selection determines which members of a population survive to pass their genetic material to successive generations. Sexual reproduction ensures mixing and recombination among the genes of their offspring, accelerating the rate of evolution. Occasionally, random mutations occur, introducing new genetic variations that may influence the evolutionary trajectory of a population.

In the work of Holland (1992), the search for a good solution to a problem translates into a search over specific bit strings. The problem is encoded by identifying a set of binary characteristics relevant to the task at hand. Each candidate solution is then represented as a fixed-length bit string, where each bit indicates the presence (1) or absence (0) of a corresponding characteristic. For instance, when evaluating apartments to rent, the characteristics could include "has a balcony," "is near public transport," "includes heating," and "allows pets." A candidate apartment represented by the string 1101 would have a balcony, be near public transport, allow pets, but not include heating. This binary encoding forms the foundation for constructing a population of potential solutions.

The algorithm begins with a randomly generated set of such binary strings, each evaluated using a problem-specific fitness function that quantifies its quality. In

the apartment example, the fitness function could score each apartment based on how well it matches a tenant's preferences, assigning higher values to configurations that fulfill more desirable traits. Over successive generations, higher-performing strings are selected and recombined to form new candidate solutions. Occasional mutations prevents the population from becoming uniform and thereby incapable of further evolution. Through this evolutionary process, the population gradually converges toward increasingly good solutions by continually mixing and propagating advantageous characteristics.

As discussed in Section 3.1, the search spaces associated with complex combinatorial optimization problems can become extremely large. GAs address this challenge by effectively casting a wide net over the search space, allowing simultaneous sampling from multiple regions. Regions with higher average fitness tend to be sampled more frequently, guiding the search toward most promising regions over time.

Algorithm 6 outlines the high-level procedure of a simple genetic algorithm.

Algorithm 6 Simple Genetic Algorithm (adapted from Mitchell (1998)). The algorithm evolves a population of fixed-length binary string chromosomes using fitness-proportionate selection, and single-point crossover and bit-wise mutation as the variation operators

Input: Population size n, chromosome length l, fitness function f, crossover probability p_c, mutation probability p_m
Initialize population P_0 with n random binary strings of length l
$t \leftarrow 0$
while not StoppingCriteriaMet() **do**
    Evaluate fitness $f(x)$ for each $x \in P_t$
    Initialize new population $P' \leftarrow \emptyset$
    while $|P'| < n$ **do**
        Select parents $x_1, x_2 \sim P_t$ with fitness proportionate selection
        Perform single-point crossover on x_1, x_2 to obtain offspring y_1, y_2 with probability p_c
        Otherwise, set $y_1 \leftarrow x_1$, $y_2 \leftarrow x_2$
        Mutate each bit in y_1 and y_2 independently with probability p_m
        Add y_1 and y_2 to P'
    end while
    if $|P'| > n$ **then** // Odd population size
        Remove one individual at random from P'
    end if
    $P_{t+1} \leftarrow P'$
    $t \leftarrow t + 1$
end while
Output: Best individual(s) in P_t according to fitness f

In each generation, parent chromosomes are selected using fitness-proportionate selection (Algorithm 7). Offspring are then produced using single-point crossover

(Algorithm 8) with probability p_c, and otherwise copied directly. After crossover, each bit of the offspring is independently mutated with probability p_m, a process known as bit-wise mutation (Algorithm 9). This evolutionary cycle is repeated until a predefined stopping criterion is met, checked with the function `StoppingCriteriaMet()`.

Algorithm 7 Fitness-proportionate selection (roulette wheel selection)

Input: Population P with fitness values $f(x)$ for each $x \in P$
Compute total fitness $F = \sum_{x \in P} f(x)$
For each $x \in P$, compute selection probability $p(x) = f(x)/F$
Sample two individuals $x_1, x_2 \in P$ independently according to distribution $p(x)$
Output: Selected parents x_1, x_2

Algorithm 8 Single-point crossover

Input: Parent chromosomes x_1, x_2 of length l
Sample crossover point $k \sim \{1, 2, \ldots, l-1\}$
$y_1 \leftarrow$ concatenate $x_1[1:k]$ with $x_2[k+1:l]$
$y_2 \leftarrow$ concatenate $x_2[1:k]$ with $x_1[k+1:l]$
Output: Offspring chromosomes y_1, y_2

Algorithm 9 Bit-wise mutation (flip-bit mutation)

Input: Chromosome y of length l, mutation probability p_m
for $j = 1$ to l **do**
    Sample $r \sim U(0, 1)$
    if $r < p_m$ **then**
        Flip bit $y[j]$ (i.e., $y[j] \leftarrow 1 - y[j]$)
    end if
end for
Output: Mutated chromosome y

Fitness-proportionate selection (also known as roulette wheel selection) selects individuals for reproduction with a probability proportional to their fitness. This means that highly fit individuals have a greater chance of being selected, but even lower-fitness individuals retain a non-zero probability. While intuitive and historically common, fitness-proportionate selection may lead to premature convergence if a few individuals dominate early Talbi (2009).

In modern GAs, fitness-proportionate selection has been largely replaced by rank-based selection methods (Goldberg & Deb, 1991). The most common variant is tournament selection, described in Algorithm 10. In this method, a small is subset randomly sampled from the population, and the fittest among them is selected. The most typical tournament size is 2, but a larger tournament size may also be used (Goldberg & Deb, 1991). Tournament selection is simple to implement, easily parallelizable, and allows precise control over selection pressure through the tournament size parameter.

Algorithm 10 Tournament selection

1: **Input:** Population P with fitness values $f(x)$ for each $x \in P$, tournament size k
2: Randomly sample k individuals from P to form tournament set T_1
3: $x_1 \leftarrow \arg\max_{x \in T_1} f(x)$
4: Randomly sample k individuals from P to form tournament set T_2
5: $x_2 \leftarrow \arg\max_{x \in T_2} f(x)$
6: **Output:** Selected parents x_1, x_2

Figure 8 illustrates the principle of single-point crossover applied to two binary-valued chromosomes. In this example, the crossover point is located at position $k = 4$, meaning the genetic material is exchanged after the fourth gene. The resulting offspring inherit the first k genes from one parent and the remaining genes from the other.

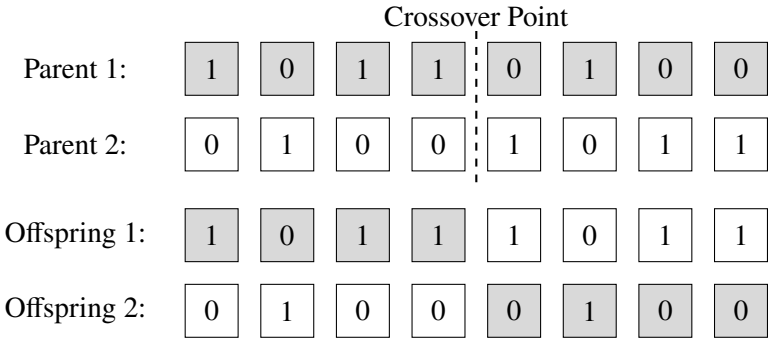


Figure 8: Single-point crossover in a genetic algorithm. Two parent chromosomes exchange genetic material at position $k = 4$, resulting in two new offspring. Genes inherited from the first parent are highlighted in gray

Figure 9 illustrates the bit-flip mutation operator applied to a single binary-valued chromosome. In this example, the gene at position 5 (using 1-based indexing) is selected for mutation and flipped from 0 to 1. The number of genes that get mutated can range between 0 and L, where L is the length of the chromosome. However, often the mutation probability is low, so it is common that none or only a few genes gets mutated.

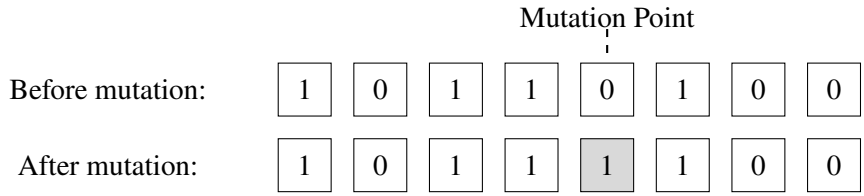


Figure 9: Bit-flip mutation in a genetic algorithm. The bit at position 5 is flipped from 0 to 1, highlighted in gray

In the classic simple GA by Holland (1992), outlined in Algorithm 6, the offspring replace their parents in the subsequent generation. A direct consequence is that the best

solution might not survive from one generation to the next. Elitism is a mechanism whereby the best solution in the current generation is directly passed to the subsequent generation. It was shown by Rudolph (1994) that the canonical GA converges to the global optimum when elitism is applied.

Typically, elitism is parametrized such that a given proportion of the current population (e.g., chromosomes with top 5% fitness values) is passed directly to the subsequent population. A side effect of elitism is that it increases selective pressure by limiting the number of "slots" for newly created chromosomes in the subsequent generation (Gendreau & Potvin, 2019).

GAs have several strengths over traditional optimization methods. Notably, they are well-suited for complex objective functions and are inherently parallelizable (Yang, 2010). GAs can effectively handle problems with objective functions that are linear or nonlinear, stationary or non-stationary, continuous or discontinuous, smooth or non-smooth, and even stochastic.

The most straightforward form of parallelization is the master-subordinate model, in which the evaluation of individuals' fitness is distributed across multiple processors. Since fitness evaluation is often the most computationally expensive component of a GA, this approach can yield substantial performance improvements (Cantú-Paz, 1998). As a result, GAs are sometimes described as "embarrassingly parallel". Other parallelization approaches include fine-grained and coarse-grained (or island-based) models (Cantú-Paz, 1998). These methods are outside the scope of this thesis, but it is worth noting that they alter the behavior of the algorithm itself. Unlike the master-subordinate model, which retains global selection and mating, fine-grained and island models restrict these operations to local neighborhoods or subpopulations, introducing additional parameters and affecting convergence.

The main weakness of GAs is its sensitivity to the formulation of the fitness function and hyperparameters such as population size, mutation and crossover probabilities, and the selection method (Yang, 2010). Any inappropriate choice can make it difficult for the algorithm to converge or in the worst case produce meaningless results. Moreover, due to the inbuilt randomness in GA, successive runs may produce different results.

3.4.2 Hidden Genes Genetic Algorithm (HGGA)

In order to recombine different chromosomes in a GA, their lengths must be equal. Hence, the number of design (or decision) variables must be the same in all solutions. However, in many applications, the number of decision variables is a question in itself. Abdelkhalik (2013) introduced the Hidden Genes Genetic Algorithm (HGGA) to allow the standard GA (simply GA hereafter) to be applied for problems with variable-size design space (VSDS). Since then, Abdelkhalik and collaborators have shown that HGGA effectively solves complex VSDS optimization problems (Abdelkhalik & Darani, 2016, 2018)

In HGGAs, some genes are hidden (inactive), so that their value does not affect the evaluation of chromosomes (i.e., computing the fitness function). The genes that are hidden in a chromosome represent variables that do not appear in a specific solution. This concept allows GA to handle VSDS optimization problems. Let DV_{max} be the maximum number of design variables. In HGGAs, all the solutions (chromosomes) have length DV_{max}. Thus, they can be treated similarly and standard GA operators such as crossover can be applied to them.

To determine whether a gene is hidden or not, each gene is associated with a tag. If the value of a tag is 1, then the corresponding gene is hidden (inactive), and if it is 0, the gene is not hidden (active) (Abdelkhalik & Darani, 2016). The tags are evolved through crossover and/or mutation operations, similarly to the genes corresponding to the design variables. Figure 10 illustrates the tags concept in the HGGA framework. In this example, genes g_1 and g_4 are hidden (highlighted in gray), and the tag row is also shown in gray to indicate its passive nature.

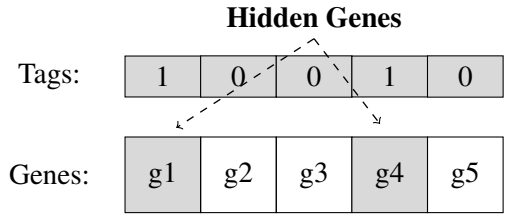


Figure 10: The concept of hidden genes and corresponding tags in a HGGA. Genes g_1 and g_4 have tags = 1, making them "hidden". Adapted from Abdelkhalik and Darani (2016)

Abdelkhalik and his collaborators drew the idea of hidden genes from the workings of the deoxyribonucleic acid (DNA). DNA is organized into a long structure called chromosome. Genes, which are instructions for making a protein, are contained in the DNA and are coded with a specific language with only 4 letters (A, G, T, and C) and 64 words. Differences between these words cause genes to produce different proteins, making cells of distinct organs function differently. An additional layer of coding determines which genes should be transcribed by the cell, depending on the cell's specific role. For instance, genes critical for heart function are turned off in kidney cells, and vice versa. The genes that are turned off are called hidden genes. There are several ways to hide genes from the cell. In one way, chemical groups get attached to the DNA and cover up parts of the gene. In another way, a cell can produce a protein that marks the genes to be read (Starr, 2013).

Algorithm 11 presents the Hidden Genes Genetic Algorithm (HGGA). The procedure begins by determining the maximum number of design variables, generating an initial population of chromosomes, and evaluating the fitness of each chromosome while excluding hidden genes. The population is then refined through iterative rounds of selection, crossover, and mutation until a stopping condition, checked by `StoppingCriteriaMet()`, is satisfied. The final output is the best chromosome(s) according to the specified fitness function.

Algorithm 11 The Hidden Genes Genetic Algorithm (HGGA). Adapted from Abdelkhalik (2013)

Input: Maximum number of design variables $DV_{\max}$, population size n, fitness function f, selection method, crossover operator, mutation operator
$C_L \leftarrow DV_{\max}$ // Chromosome length
Create initial population P_0 of size n with random chromosomes of length C_L
$t \leftarrow 0$
for all chromosome $x \in P_0$ **do**
    Determine hidden genes for x
    Compute $f(x)$ by excluding hidden genes
end for
while not StoppingCriteriaMet() **do**
    Selection: Perform competitive selection on P_t
    Variation: Apply crossover and mutation to generate offspring set O
    for all offspring $y \in O$ **do**
        Determine hidden genes for y
        Compute $f(y)$ by excluding hidden genes
    end for
    Form new population P_{t+1} from selected parents and O
    $t \leftarrow t + 1$
end while
Output: Best chromosome(s) in P_t according to f

An important design feature of the HGGA is the evolution of tags. In the first paper, Abdelkhalik and collaborators used a simple approach called the feasibility mechanism. The feasibility mechanism rule assumes initially no hidden genes in a chromosome; if the obtained chromosome is feasible then there is no hidden genes. If the solution is not feasible, then starting from one end of the chromosome the algorithm hides genes—one by one—until the chromosome becomes feasible (Abdelkhalik, 2013).

After having researched HGGAs for a few years, Abdelkhalik and Darani (2016, 2018) had developed further mechanisms for evolving tags. Abdelkhalik and Darani (2018) presented two main approaches: logical evolution and stochastic evolution. In *logical evolution*, tags for the offspring are derived from the tags of their parents using fixed logical rules such as OR or AND. In *stochastic evolution*, tags are evolved through genetic operators such as mutation and crossover, similarly to the main genes in the chromosome.

3.4.3 Applications of Genetic Algorithms

Genetic Algorithms (GA) have been used effectively in the elevator industry to solve complex combinatorial optimization problems. An important application, sometimes called the Elevator Car Routing Problem (ECRP), has been tackled for both single and double deck elevators. Tyni and Ylinen (1999) employed a GA to address the

allocation of passenger landing calls to single deck elevators. Their approach is a two-level hybrid: at the upper level, the GA allocates landing calls to elevators using selection, crossover, and mutation; at the lower level, each elevator's service sequence is determined by a simple heuristic known as the "collective control principle". This aims to minimize the average landing call time. A gene bank was used to store previously computed fitness values, which decreased CPU time by 65% (Tyni & Ylinen, 1999). This permitted real-time execution of the call-allocation algorithm in KONE's elevators. Their method was one of the first real-time applications of genetic algorithms in control systems (Tyni & Ylinen, 1999, 2001).

In this approach, the chromosome representation includes call genes, which specify which landing calls to serve, and direction genes for any elevator that is not currently moving, ensuring an idle elevator always has an assigned direction (up or down) (Tyni & Ylinen, 2001).

Sorsa et al. (2003) adapted the algorithm of Tyni and Ylinen (1999) to solve the ECRP for double deck elevators in real time. Building on the framework above, Sorsa et al. (2003) extended the chromosome to include deck genes, which determine which calls each deck serves. By combining the newly introduced deck genes with the existing call and direction genes, the algorithm can decide whether an idle elevator should start moving up or down and how each deck should respond to incoming calls. Analysis of CPU time indicated that a real time extension for quadruple-deck elevators could be feasible, even though the number of solutions would grow to 10^{120} when considering 8 quadruple-deck elevators serving 40 floors,[6] highlighting GA's applicability for very large problem instances.

Another application of GAs in the elevator industry has been the zoning problem of elevators. As described earlier, Viita-aho (2019) studied the static zoning problem and developed a GA based method to find Pareto-optimal zoning arrangements when optimizing for both core area occupied by elevators and price of the elevator system. Liu et al. (2010) studied dynamic zoning and used GA to find zoning arrangements that minimize energy usage. Their algorithm was able to reduce the energy consumption by more than 10% and the average waiting time by more than 40% when comparing to a traditional zoning method.

The HGGA has been successfully applied to standard benchmark mathematical functions and interplanetary trajectory optimization problems by Abdelkhalik and his collaborators (Abdelkhalik, 2013; Abdelkhalik & Darani, 2016, 2018). In these studies, the HGGA has consistently outperformed the standard Genetic Algorithm (GA) and demonstrated its capability to solve problems with very large search spaces.

An interplanetary trajectory optimization problem can be defined as follows. Given a range of possible departure dates from Earth, a range of possible arrival dates to a target planet, and the dry mass of the spacecraft, the objective is to determine the mission architecture and corresponding trajectory variables that optimize a specified objective function – typically minimizing the required fuel mass (Abdelkhalik &

[6]For reference, the number of solutions is 10^{36} for 8 single-deck elevators serving 20 floors

Darani, 2018). The mission architecture includes decisions on the number of planetary flybys, the number of Deep Space Manoeuvres (DSMs)[7] in each leg of the journey, as well as the specific flyby planets, the dates and times of flybys and DSMs, the magnitude and direction of each DSM, and the exact launch and arrival dates.

Such problems are inherently complex and, in many cases, intractable using exact optimization methods. In Abdelkhalik and Darani (2018), the HGGA was applied to solve the optimal trajectory problem from Earth to Jupiter.

To manage the high computational cost associated with interplanetary trajectory optimization, Abdelkhalik and Darani (2018) employed a two-phase solution strategy: first solving a zero-DSM problem to determine the sequence of flybys, and then optimizing the remaining variables in a multigravity-assist trajectory with deep space maneuvers. Using this approach, the HGGA with Mechanism A (mutation probability of 5%) produced a trajectory consisting of two flybys – Venus and Earth – resulting in a total mission sequence of Earth–Venus–Earth–Jupiter with a final mission cost of 10.1266 km/s. This solution improves upon previous results in the literature, such as that of Olympio et al. (2007), which assumed the same planet sequence and fixed flyby and departure dates, yielding a cost of 10.267 km/s.

Unlike earlier methods, the HGGA does not require the mission architecture to be fixed in advance and is capable of autonomously determining it as part of the optimization process. Among the tested tag evolving methods, Mechanism A and Logic A both achieved strong performance, each reaching a success rate of approximately 75% in repeated experiments. The success rate measures the proportion of runs in which the algorithm converges to the best-found solution.

The high success rate of the HGGA coupled with mechanism A in solving the trajectory of the Earth-Jupyter mission indicates that the HGGA is a robust method to solve highly complex combinatorial optimization problems with variable design space. This hints that it should be well suitable for the zoning problem, where the number of zoning arrangements is enormous and where the optimal zone count is not known in advance.

3.5 Hybrid Metaheuristics

Research on metaheuristics for combinatorial optimization has increasingly shifted from an algorithm-centric to a problem-oriented perspective. As Blum et al. (2011) notes, “nowadays the focus is on solving the problem at hand in the best way possible, rather than promoting a certain metaheuristic”. This shift has been accompanied by growing interest in hybrid metaheuristics – combinations of different metaheuristic strategies – motivated by their potential to outperform standalone algorithms.

Talbi (2002) emphasizes that “the best results found for many practical or academic optimization problems are obtained by hybrid algorithms”, citing successful combi-

[7] DSMs are propulsive impulses used to change the spacecraft’s velocity instantaneously.

nations of descent local search, simulated annealing, tabu search, and evolutionary algorithms.

The core motivation for hybridization lies in the complementary strengths of different metaheuristics. As Blum et al. (2011) further explains, "the main motivation behind the hybridization of different algorithms is to exploit the complementary character of different optimization strategies." In other words, combining metaheuristics can create synergies that enhance search performance.

Talbi (2002) suggests classifying hybrid metaheuristics based on either design or implementation considerations. Design considerations refer the hybrid algorithm itself, including its architecture and functionality. Implementation, on the other hand, concerns the hardware, programming model and environment on which the algorithm is run. Figure 11 shows the classification based on design considerations.

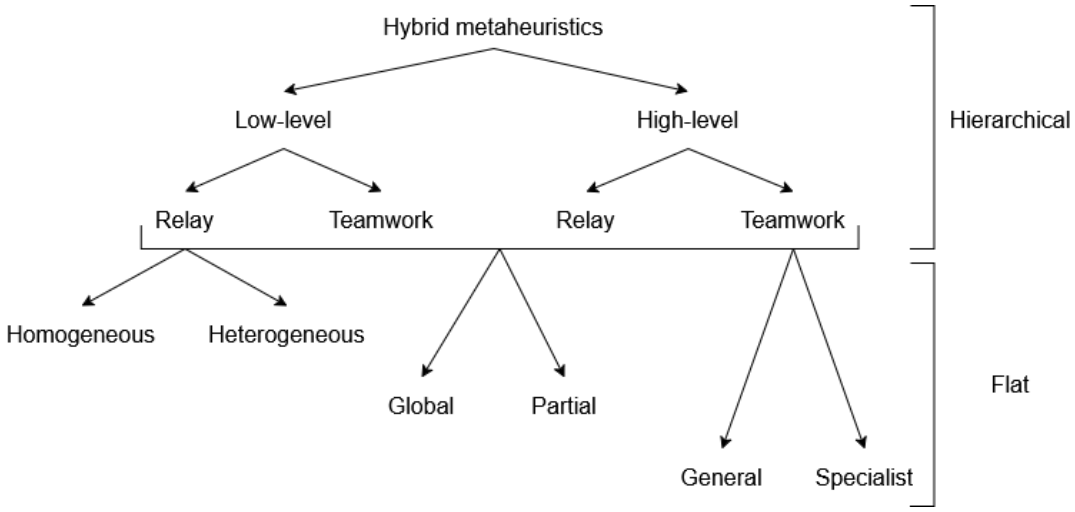


Figure 11: Classification of hybrid metaheuristics based on design considerations. Adapted from Talbi (2002)

On the first level, hybrid metaheuristics are classified to low-level and high-level hybrids. In the low-level class, a given function of a metaheuristic is replaced by another metaheuristic where as in the high-level class, different metaheuristics are self-contained. In this thesis, I will focus on high-level hybrids.

Within high-level hybrids, the second level classifies between relay and teamwork hybrids. In relay hybrids, different metaheuristics are applied in a sequential manner, using the output of the previous metaheuristic as an input for the subsequent metaheuristic. Relay hybrids can be viewed as a pipeline of metaheuristics. Teamwork hybrids, on the other hand, concerns cooperating metaheuristics that are ran in a parallel fashion. In this thesis, I focus on relay hybrids.

In high-level relay hybrids (HRH) self contained metaheuristics are executed in a sequence. Population-based metaheuristics are well suited for finding promising regions in the search space. On the other hand, it is well known that they are not

particularly good at fine-tuning solutions that are close to the optimal solutions. Therefore, a common approach is to combine evolutionary algorithms like GAs with single-solution based metaheuristics such as tabu search, which take over the phase of exploiting the promising regions in the search space (Talbi, 2002). Depending on the problem structure, sometimes also single-solution based metaheuristics may feed into population-based metaheuristics. For example, Lin et al. (1991) proposed initial solutions with SA and used GAs to enrich the solutions found.

There can be two kinds of HRHs: heterogeneous and homogeneous. In homogeneous HRHs, the same metaheuristic is used in all of the algorithms whereas in heterogeneous HRHs, different metaheuristics are used. For example, nested GAs would be considered a homogeneous high-level relay hybrid, but the application of Lin et al. (1991) above can be classified as a heterogeneous HRH.

Finally, all hybrids are either global or partial based on how the individual metaheuristics cooperate. In global hybrids, all algorithms use the full search space whereas in partial hybrids, the problem is decomposed into sub problems, each one having its own search space. An example of a partial homogeneous HRH application can be found in Husbands et al. (1991). Husbands and coauthors decomposed the job-shop scheduling problem into individual jobs, where the optimal process plan for each job was solved by a GA. A communication medium collected fittest individuals from each GA, and evaluated the resulting schedule as a whole, rewarding the best process plans.

In this thesis, a heterogeneous HRH will be used to study the problem of zoning with sky lobbies. Initially, SA will be employed to propose sky lobby floors. The sky lobby floors split the building into stacks. Next, The optimal zoning arrangement for these stacks will be solved with HGGAs. This second level method can be viewed as another hybrid, which can be classified as a partial homogeneous HRH without mutual constraints between the HGGAs. Optimal zoning arrangements for each individual stack will be combined in the SA. The SA will also factor in the shuttle elevator configuration and evaluate the objective function of the full solution. A more detailed description of the full scheme will provided in Section 4.

4 Methodology

4.1 Introduction and Overview

The objective of this thesis is to determine the optimal configuration of sky lobbies in a high-rise building such that the total core area occupied by the elevator system is minimized. The core area includes both the elevator shaft and the lobby space required for passenger transfer. Specifically, the problem is defined as follows: Given arbitrary building parameters (floor count and population per floor), what is the optimal number of sky lobbies and their respective locations such that the total elevator core area is minimized while satisfying performance constraints?

From a computational perspective, the research question results in a nested combinatorial optimization problem (COP). At the outer level, the task is to determine the optimal number and locations of sky lobbies. At the inner level, the goal is to find the optimal zoning configuration for each sub-building (or stack) defined by the selected sky lobbies, such that each elevator group serving a zone satisfies the performance criteria while contributing as little as possible to the total core area.

As discussed in Section 2.2, finding an optimal zoning arrangement is a challenging task, as the number of possible configurations grows exponentially with the floor count (see Equation (15)). Furthermore, Section 2.3 showed that the number of possible sky lobby configurations also becomes large in high-rise buildings, particularly when multiple sky lobbies are considered (see Equation (18)).

When these two problems are nested, the search space of the combinatorial optimization problem becomes prohibitive, and no directly applicable complete methods are available to solve it. As demonstrated in Section 3, metaheuristics are effective tools for addressing combinatorial optimization problems with complex structure and very large search spaces. Furthermore, the literature on hybrid metaheuristics (Section 3.5) highlights that hybrid methods are often essential for solving difficult combinatorial problems effectively. In particular, the relative strengths of different metaheuristics can be leveraged to align with the problem structure, enabling the construction of tailored and high-performing approaches. For this reason, the main research question in this thesis is addressed using a novel hybrid metaheuristic framework.

The rest of the methodology section is structured as follows. Section 4.2 presents the optimization framework, including the decision variables, the objective function, the performance constraints, and the full mathematical formulation. Section 4.3 describes the computational model, which is based on a hybrid metaheuristic that combines Simulated Annealing (SA) and a Hidden Genes Genetic Algorithm (HGGA). The implementation of SA and HGGA is explained in Sections 4.3.1 and 4.3.2, respectively, along with the methods used to tune their parameters. In Section 4.3.3, the HGGA is benchmarked against an exact method to validate its effectiveness in finding the optimal zoning arrangement for stacks.

4.2 Optimization Framework

4.2.1 Decision Variables

Let the building consist of $N \in \mathbb{Z}_+$ floors, numbered from 0 (ground floor) to N (top floor), and denote the set of floors by

$$\mathcal{F} = \{0, 1, 2, \ldots, N\}.$$

The outer-level decision variable is a vector of k ordered integers representing the floors designated as sky lobbies:

$$\mathbf{s} = (s_1, s_2, \ldots, s_k), \quad \text{with } 0 < s_1 < s_2 < \cdots < s_k < N.$$

The k sky lobbies in $\mathbf{s}$ partition the building into $k + 1$ contiguous vertical segments (stacks). The first stack (stack 0) comprises floors from 0 up to $s_1 - 1$ (with floor s_1 acting as the sky lobby). The second stack spans from s_1 to $s_2 - 1$, and each intermediate stack is defined by the floors between consecutive sky lobbies. Finally, the last stack (stack k) covers floors from s_k to N.

For each stack $j = 0, 1, \ldots, k$, the inner-level decision variable, is a vector of positive integers representing the upper floors of the elevator zones within that stack. Denote the zoning configuration for stack j by

$$\mathbf{z}^{(j)} = \left(u_1^{(j)}, u_2^{(j)}, \ldots, u_{n_j}^{(j)}\right) \subseteq \mathcal{F}^{(j)},$$

where each $u_i^{(j)}$ represents the upper floor of zone i in stack j, and $\mathcal{F}^{(j)}$ is the set of floors in stack j.

In each stack, the first zone serves floors starting from the base of the stack (i.e., floor 0 for stack 0 or $s_j + 1$ for stacks $j > 0$) up to $u_1^{(j)}$. Each subsequent zone i serves the floors immediately following the previous zone, that is, from $u_{i-1}^{(j)} + 1$ to $u_i^{(j)}$, with the final zone extending to the top floor of the stack.

For example, consider a 20-story stack (with floors numbered 0 through 20) divided into two zones. If the first zone's upper floor is 10, then the first zone operates from floor 0 to 10, and the second zone serves floors 11 to 20.

Thus, the overall configuration of the elevator system is represented by the pair

$$\left(\mathbf{s}, \{\mathbf{z}^{(j)}\}_{j=0}^{k}\right),$$

where the vector $\mathbf{s}$ determines the sky lobby placements and the collection $\{\mathbf{z}^{(j)}\}_{j=0}^{k}$ specifies the zoning configurations within each stack.

4.2.2 Objective Function

In this thesis, the focus is on minimizing a single objective: the core area occupied by elevators in a building with sky lobbies.

In Ruokokoski et al. (2018), the core area occupied by elevators in a single zone was determined as

$$CA(u, C, L) = (u + 1) \cdot L \cdot A(C), \tag{20}$$

where u is the upper floor of the zone, L is the number of elevators, and $A(C)$ is the cross-sectional area of the shaft, with C (the elevator capacity, measured in persons) chosen from a discrete set.

Equation (20) does not take into account the lobby area of elevators. Omitting the lobby area from the elevator core area calculation creates a bias in favor of adding more elevators, as their space requirements appear underestimated. Therefore, what follows is a new formulation for core area which takes lobby area into account. In the proposed formulation, elevator capacities are chosen from a discrete set, with each capacity uniquely determining the corresponding shaft dimensions and car depth. Let us denote the shaft width, shaft depth, and car depth as s_w, s_d, and c_d, respectively, all of which are functions of C.

The shaft area of an elevator car can now be expressed as

$$A_s(C) = s_w(C) \cdot s_d(C), \tag{21}$$

and the lobby area of an elevator car as

$$A_l(C) = s_w(C) \cdot c_d(C). \tag{22}$$

Thus, the core area of an elevator car is defined as

$$\begin{aligned} CA_{car}(C) &= A_s(C) + A_l(C) \\ &= s_w(C) \cdot s_d(C) + s_w(C) \cdot c_d(C) \\ &= s_w(C) \left[s_d(C) + c_d(C) \right]. \end{aligned} \tag{23}$$

Now, the core area of elevators in a single zone is expressed as

$$CA_{zone}(u, C, L) = (u + 1) \cdot L \cdot CA_{car}(C), \tag{24}$$

where the notation follows Equation (20).

For a given stack j with a multi-zone configuration $\mathbf{z}^{(j)} = \left(u_1^{(j)}, u_2^{(j)}, \ldots, u_{n_j}^{(j)}\right)$, its total core area is

$$CA_{stack}^{(j)} = \sum_{i=1}^{n_j} CA_{zone}(u_i^{(j)}, C_i^{(j)}, L_i^{(j)}). \tag{25}$$

When sky lobbies are used, all but the first stack must have a shuttle elevator group to transfer passengers from the ground floor to the sky lobby. Therefore, an additional

core area corresponding to the shuttle elevator group must be included. In this case, the shuttle core area is denoted as

$$CA_{shuttle}(u_s, C_s, L_s), \tag{26}$$

where u_s, C_s, and L_s refer to the upper floor, car size, and elevator count of the shuttle lift group, respectively. The computation is performed in the same way as for local elevator groups in Equation (25), with the exception that the lobby area is calculated only for the sky lobby floor and the lower terminal floor – that is, the floors where the shuttle elevators stop.[8] The total core area of a stack then becomes

$$CA^{(j)}_{stack} = \sum_{i=1}^{n_j} CA_{zone}(u_i^{(j)}, C_i^{(j)}, L_i^{(j)}) + \mathbf{1}_{\{j>0\}} CA^{(j)}_{shuttle}(u_s^{(j)}, C_s^{(j)}, L_s^{(j)}), \tag{27}$$

where $\mathbf{1}_{\{j>0\}}$ is an indicator function taking the value 1 for all but the first stack (i.e., when $j > 0$) and 0 otherwise.

Finally, for a building partitioned into $k+1$ stacks (with k sky lobbies), the full building core area is

$$CA_{building} = \sum_{j=0}^{k} CA^{(j)}_{stack}. \tag{28}$$

Equation (28) represents the objective value in this thesis. Therefore, let us define

$$f\left(\mathbf{s}, \{\mathbf{z}^{(j)}\}_{j=0}^{k}\right) = CA_{building}. \tag{29}$$

4.2.3 Inequality Constraints

For each local elevator group (zone) in stack $j \in \{0, 1, \ldots, k\}$ and for each zone $i \in \{1, \ldots, n_j\}$, we impose the following performance constraints: the minimum relative handling capacity (%HC) of elevator groups is set 12% / 5 minutes (or 300 seconds, which is used in subsequent formulas), the maximum interval (INT) is set to 30 seconds, and the maximum nominal travel time (NTT) is 25 seconds. These thresholds reflect typical performance criteria for office buildings and are expressed formally below.

Let $C_i^{(j)}$, $L_i^{(j)}$, and $RTT_i^{(j)}$ denote, respectively, the capacity, the number of elevators, and the round-trip time for the ith zone in stack j, and let

$$POP\left(\ell_i^{(j)}, u_i^{(j)}\right)$$

denote the population served by the zone, where the lower floor $\ell_i^{(j)}$ is defined as

$$\ell_i^{(j)} = \begin{cases} 1, & \text{if } j = 0 \text{ and } i = 1, \\ s_j + 1, & \text{if } j > 0 \text{ and } i = 1, \\ u_{i-1}^{(j)} + 1, & \text{if } i > 1. \end{cases}$$

[8] The area adjacent to the shuttle elevator shafts on intermediate floors is typically used for auxiliary functions, such as restrooms.

Furthermore, let h denote the floor height and $v_i^{(j)}$ the rated speed of the elevator group in zone i of stack j. Then, the local elevator group constraints for %HC, INT, and NTT are respectively given by

$$g_{1,j,i}(\mathbf{s}, \mathbf{z}^{(j)}) = \frac{300 \cdot CLF \cdot C_i^{(j)} \cdot L_i^{(j)}}{RTT_i^{(j)} \cdot POP\left(\ell_i^{(j)}, u_i^{(j)}\right)} \times 100\% \geq 12\%, \tag{30}$$

$$g_{2,j,i}(\mathbf{s}, \mathbf{z}^{(j)}) = \frac{RTT_i^{(j)}}{L_i^{(j)}} \leq 30\,\text{s}, \tag{31}$$

$$g_{3,j,i}(\mathbf{s}, \mathbf{z}^{(j)}) = \frac{h}{v_i^{(j)}} \leq 25\,\text{s}. \tag{32}$$

For shuttle elevator groups, which are present only in stacks $j \in \{1, \ldots, k\}$, let the decision variables be given by $(u_s^{(j)}, C_s^{(j)}, L_s^{(j)})$, and denote by $RTT_s^{(j)}$ and $v_s^{(j)}$ the corresponding round-trip time and operating speed. In this configuration, the shuttle elevator group in stack j is responsible for handling passengers from the sky lobby on floor s_j to the top floor of the stack (the floor preceding the next sky lobby), or alternatively to the top floor N if $j = k$). Accordingly, we define

$$POP_s(j) = \begin{cases} POP(s_j, s_{j+1} - 1), & \text{for } j \in \{1, \ldots, k-1\}, \\ POP(s_k, N), & \text{for } j = k. \end{cases}$$

Thus, the constraints for the shuttle elevator groups constraints for %HC, INT, and NTT are respectively formulated as

$$g_{1,j}^{(\text{shuttle})}(\mathbf{s}) = \frac{300 \cdot CLF \cdot C_s^{(j)} \cdot L_s^{(j)}}{RTT_s^{(j)} \cdot POP_s(j)} \times 100\% \geq 12\%, \tag{33}$$

$$g_{2,j}^{(\text{shuttle})}(\mathbf{s}) = \frac{RTT_s^{(j)}}{L_s^{(j)}} \leq 30\,\text{s}, \tag{34}$$

$$g_{3,j}^{(\text{shuttle})}(\mathbf{s}) = \frac{h}{v_s^{(j)}} \leq 25\,\text{s}. \tag{35}$$

In summary, the complete set of inequality constraints that must be satisfied by the

model is given by

$$
\begin{aligned}
&\forall\, j \in \{0, \ldots, k\},\ \forall\, i \in \{1, \ldots, n_j\} : \\
&\quad g_{1,j,i}(\mathbf{s}, \mathbf{z}^{(j)}) \geq 12\%, && (36)\\
&\quad g_{2,j,i}(\mathbf{s}, \mathbf{z}^{(j)}) \leq 30\,\mathrm{s}, && (37)\\
&\quad g_{3,j,i}(\mathbf{s}, \mathbf{z}^{(j)}) \leq 25\,\mathrm{s}, && (38)\\
&\forall\, j \in \{1, \ldots, k\} : \\
&\quad g_{1,j}^{(\mathrm{shuttle})}(\mathbf{s}) \geq 12\%, && (39)\\
&\quad g_{2,j}^{(\mathrm{shuttle})}(\mathbf{s}) \leq 30\,\mathrm{s}, && (40)\\
&\quad g_{3,j}^{(\mathrm{shuttle})}(\mathbf{s}) \leq 25\,\mathrm{s}. && (41)
\end{aligned}
$$

4.2.4 Problem Formulation

Having defined the objective function and inequality constraints of the problem in the previous subsections, the complete formulation for the optimization problem is

$$
\begin{aligned}
\min_{\mathbf{s},\, \{\mathbf{z}^{(j)}\}_{j=0}^{k}} \quad & f\left(\mathbf{s}, \{\mathbf{z}^{(j)}\}_{j=0}^{k}\right) = CA_{building} \\
\text{s.t.} \quad & g_{1,j,i}(\mathbf{s}, \mathbf{z}^{(j)}) \geq 12\%, \quad \forall\, j \in \{0, \ldots, k\},\ \forall\, i \in \{1, \ldots, n_j\}, \\
& g_{2,j,i}(\mathbf{s}, \mathbf{z}^{(j)}) \leq 30\,\mathrm{s}, \quad \forall\, j \in \{0, \ldots, k\},\ \forall\, i \in \{1, \ldots, n_j\}, \\
& g_{3,j,i}(\mathbf{s}, \mathbf{z}^{(j)}) \leq 25\,\mathrm{s}, \quad \forall\, j \in \{0, \ldots, k\},\ \forall\, i \in \{1, \ldots, n_j\}, \\
& g_{1,j}^{(\mathrm{shuttle})}(\mathbf{s}) \geq 12\%, \quad \forall\, j \in \{1, \ldots, k\}, \\
& g_{2,j}^{(\mathrm{shuttle})}(\mathbf{s}) \leq 30\,\mathrm{s}, \quad \forall\, j \in \{1, \ldots, k\}, \\
& g_{3,j}^{(\mathrm{shuttle})}(\mathbf{s}) \leq 25\,\mathrm{s}, \quad \forall\, j \in \{1, \ldots, k\}, \\
& \mathbf{s} \in \left\{(s_1, \ldots, s_k) \subseteq \mathcal{F} : 0 < s_1 < s_2 < \cdots < s_k < N\right\}, \\
& \mathbf{z}^{(j)} \subseteq \mathcal{F}^{(j)}, \quad \forall\, j = 0, \ldots, k.
\end{aligned} \tag{42}
$$

4.3 Computational Model

A novel hybrid metaheuristic is developed to solve the optimization problem in (42). The hybridization is a heterogeneous high-level relay hybrid (HRH), which applies SA and HGGA sequentially. Initially, SA will be employed to propose sky lobby floors. The sky lobby floors split the building into stacks. Next, the optimal zoning arrangement for these stacks will be solved with HGGAs. This second level method can be viewed as another hybridization, classified as a partial homogeneous HRH without mutual constraints between the individual metaheuristics components (that is, the HGGAs). The optimal zoning arrangements for each individual stack in the inner level will be combined on the outer level, operated by the SA. The SA will also factor in the shuttle elevator configuration and evaluate the objective function of the

full solution. A high-level pseudo-code of the hybridization scheme is provided in Algorithm 12.

Algorithm 12 High-level procedure to determine optimal sky lobby configuration using a heterogeneous HRH (SA + HGGAs)

```
Input: Floor count, population per floor, sky lobby count
Generate initial sky lobby floor split using SA
while termination criteria not met do
    Split the building into stacks based on sky lobby floors
    for each stack j = 0, ..., k do
        Solve optimal zoning arrangement using HGGA
        if j > 0 then
            Determine shuttle elevator configuration
        end if
    end for
    Evaluate total core area (local elevator groups + shuttle elevator groups)
    Perturb sky lobby configuration using the SA move operator
end while
Output: Best solution found during the outer SA loop
```

A more detailed description of the implementations of the individual metaheuristics – SA and HGGA – will be provided in the next two subsections, 4.3.1 and 4.3.2, respectively.

4.3.1 Simulated Annealing

As explained earlier, SA is used on the outer level of the hybridization to find optimal floors for the sky lobbies. In this subsection, I explain how SA was adapted for this problem. In particular, how the solutions were represented, what initial solutions were used, how the neighborhood function was defined, and how the cooling schedule among other hyperparameters were determined.

A solution in the SA is represented as an array of k ordered integers denoting the floors where sky lobbies are placed:

$$\mathbf{s} = (s_1, s_2, \ldots, s_k), \quad \text{with } 0 < s_1 < s_2 < \cdots < s_k < N.$$

The initial solution for the SA is chosen such that sky lobbies divide the building into evenly spaced stacks. For example, if the building has 100 floors and one sky lobby, it is placed on the 50th floor. If there are two sky lobbies, they are placed on the 33rd and 66th floors, and so on. This method was selected for its simplicity and based on the observation that, in practice, sky lobbies are often placed in this manner. Furthermore, it is expected that the optimal solution is not far from this initial configuration, as the core area of a stack generally grows exponentially with floor count. In other words, it is seen unlikely that the optimal solution would involve significantly uneven stack sizes.

In this work, I introduce two parameters that govern the neighborhood function within the simulated annealing algorithm. The first parameter, referred to as the *perturbed floors* parameter, determines which sky lobbies will be subject to perturbation. This parameter may assume one of three values. When set to "all", every sky lobby in the solution is selected for perturbation. When set to "random", a random number of sky lobbies, uniformly chosen between one and the total number of sky lobbies, is selected. When set to 1, exactly one sky lobby is randomly chosen for perturbation.

The second parameter, the *neighborhood* size parameter, is a positive integer that specifies the range within which a sky lobby floor may be perturbed. Specifically, if the current floor is f and the neighborhood size is n, a new floor is chosen uniformly at random from the interval $[f - n, f + n]$. After constructing the set of sky lobbies to be perturbed and generating their new floors, the resulting solution s' is validated against the stack size constraints using the function `ValidateSolution()`. Should the constraints be violated, the proposed move is discarded.

Algorithm 13 presents the pseudo-code for the neighborhood function in the SA.

Algorithm 13 Definition of the neighborhood function in the SA component

Input: current solution s, perturbed floors parameter p, neighborhood size parameter n, total number of sky lobbies N
$s' \leftarrow s$
// Determine the set I of sky lobby indices to be perturbed
if p = "all" **then**
    $I \leftarrow \{0, 1, \ldots, N-1\}$
else if p = "random" **then**
    Draw an integer k uniformly at random from $\{1, \ldots, N\}$
    Randomly select k distinct indices from $\{0, 1, \ldots, N-1\}$ and assign them to I
else if $p = 1$ **then**
    Let L be a predetermined sky lobby index
    $I \leftarrow \{L\}$
end if
for each index $i \in I$ **do**
    Let $f \leftarrow s[i]$ // current floor of sky lobby i
    Draw a new floor f' uniformly at random from $[f - n,\ f + n]$
    Update $s'[i] \leftarrow f'$
end for
if ValidateSolution(s') = false **then**
    Output: current solution s // Stack size constraint violated
else
    Output: proposed solution s' // The solution is feasible
end if

The optimal parameters of the neighborhood function, along with those defining the cooling schedule, were determined through an extensive grid search. First, data were

collected on the range of possible objective function values to identify a suitable range for the initial and minimum temperatures. Next, multiple candidate values for the initial and minimum temperatures were tested in combination with linear, quadratic, logarithmic, and exponential cooling schedules across various problem instances. The parameters of the neighborhood function – *Perturbed floors* and *Neighborhood size* – were also included in the hyperparameter grid. Table 3 presents the range of values explored for each hyperparameter.

Hyperparameter	**Values**
Initial temperature	{500, 1000, 5000, 10000, 30000}
Minimum temperature	{1, 10, 100, 300}
Steps	{1000, 3000, 5000, 10000}
Cooling schedule	{`linear`, `quadratic`, `logarithmic`, `exponential`}
Neighborhood size	{1, 2, 3, 5, 10}
Perturbed floors	{1, `random`, `all`}

Table 3: Ranges of values used for each hyperparameter in the grid search. The parameters include settings for the simulated annealing cooling schedule and neighborhood structure

The hyperparameter grid shown in Table 3 was tested on a representative subset of instances, defined by the key parameters of the problem: the building's floor count, population per floor, and the number of sky lobbies. The tested floor count values were $40, 60, \ldots, 200$, and the tested population per floor values were $20, 40, \ldots, 200$. The number of sky lobbies tested was 1, 2, and 3. Each combination of hyperparameters and problem instances was run using 10 different random number generator seeds. For each case, the mean and standard deviation of the objective function were computed across the resulting sample.

Based on the results, the optimal hyperparameter configuration for each instance was identified by sorting the configurations in increasing order according to the following priority: (1) mean objective value and (2) standard deviation of the objective value. A mapping structure was then developed to generalize the optimal hyperparameter selection to unseen instances, including those not present in the original grid.

The mapping rule is as follows: round the floor count up to the nearest value in the grid (e.g., 19 is rounded up to 20, and 21 to 25). Population count rounding follows the same logic: 24 is rounded up to 25, while 26 is rounded up to 30. Thus, for the input parameter tuples (19, 24) and (21, 26), the corresponding hyperparameters from instances (20, 25) and (25, 30), respectively, would be used. The rationale behind this approach is that selecting hyperparameters optimized for a problem with a larger search space is generally safer than choosing those from a smaller one.

Table 4 shows optimal hyperparametrizations for a subset of instances with floor count 100. In the table, the problem parameters – floor count (FC), population per floor

(Pop), and sky lobby count (SL) – are provided in the first three columns and are visually separated by a vertical line from the simulated annealing hyperparameters: initial temperature (IT), minimum temperature (MT), steps (St), cooling schedule (CS), neighborhood size (NS), and perturbed floors (PF). The instances are grouped into separate sections by sky lobby count using horizontal lines for clarity.

FC	Pop	SL	IT	MT	St	CS	NS	PF
100	40	1	500	1	1000	`exponential`	1	random
100	100	1	500	1	1000	`exponential`	1	random
100	160	1	500	1	1000	`logarithmic`	2	1
100	200	1	500	1	1000	`logarithmic`	3	1
100	40	2	500	1	1000	`exponential`	1	1
100	100	2	500	1	1000	`exponential`	1	1
100	160	2	500	1	1000	`logarithmic`	2	1
100	200	2	500	1	1000	`logarithmic`	3	1
100	40	3	500	1	1000	`exponential`	1	random
100	100	3	500	1	1000	`exponential`	1	random
100	160	3	500	1	1000	`logarithmic`	2	1
100	200	3	500	1	1000	`logarithmic`	3	1

Table 4: Optimal SA hyperparametrizations for selected representative instances. The table is organized by floor count (FC), population per floor (Pop) and various sky lobby counts (SL). The vertical line separates the problem parameters (FC, Pop, SL) from the SA hyperparameters: IT = initial temperature, MT = minimum temperature, St = steps, CS = cooling schedule, NS = neighborhood size, and PF = perturbed floors. Optimal hyperparameters were chosen by ranking the hyperparameter combinations on aggregated results over 10 random number generator seeds with the following priority (1) mean objective value (2) standard deviation of the objective value

The table shows that for a floor count of 100, the best initial and minimum temperatures are 500 and 1, respectively. The optimal number of steps was 1000. Among the tested cooling schedules, both exponential and logarithmic schedules appeared to perform best. Regarding the neighborhood size, the most effective values for the perturbed floors parameter were either 1 or a random count, and the neighborhood size ranged from 1 to 3.

4.3.2 Hidden Genes Genetic Algorithm

As explained earlier, a Hidden Genes Genetic Algorithm (HGGA) is used on the inner level to find the optimal zoning arrangement for a stack.

For each stack $j = 0, 1, \ldots, k$, the inner-level decision variable is an array of positive

integers representing the upper floors of the elevator zones within that stack. The solution in hence represented as

$$\mathbf{z}^{(j)} = \left(u_1^{(j)}, u_2^{(j)}, \ldots, u_{n_j}^{(j)}\right) \subseteq \mathcal{F}^{(j)},$$

where each $u_i^{(j)}$ represents the upper floor of zone i in stack j, and $\mathcal{F}^{(j)}$ is the set of floors in stack j.

In a genetic algorithm context, the array $\mathbf{z}^{(j)}$ is a chromosome consisting of zone genes. To implement the hidden layer, a corresponding binary tag is assigned to each gene $u_i^{(j)}$ in the solution, indicating whether a zone is active or hidden. This is best understood through the following example.

Figure 12 shows a chromosome encoding for a 40-floor building. The first and fourth genes are tagged as hidden, leaving three active zones. Their upper floors are 12, 26, and 40, respectively. These define the active zoning scheme: floors 0 – 12, 13 – 26, and 27 – 40.

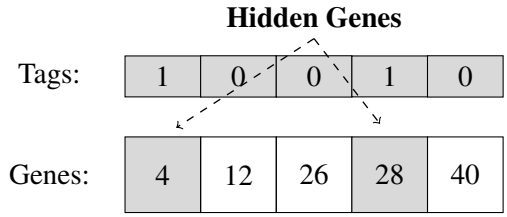


Figure 12: Adaptation of the hidden genes concept to elevator zoning. The chromosome encodes five candidate upper floors (4, 12, 26, 28, and 40) with corresponding tags. Genes with tag = 1 are hidden and ignored during zone construction. In this example, the first and fourth zones are hidden, resulting in three active zones

When evaluating a candidate solution by computing the fitness value of the chromosome, hidden zones are excluded from the zoning arrangement, and the active zones define the actual floor division of zones within the stack.

Since the number of genes in a chromosome sets an upper limit on the number of active zones, the chromosome must contain at least as many genes as the optimal solution requires. However, increasing the number of genes also increases the number of potential solutions that can be constructed. For example, if the optimal solution for a simple building contains only two zones, using a chromosome with 20 genes would lead to a population where most chromosomes have too many zones. Conversely, such a chromosome structure might be well suited for a larger building with a dense population, which may require more zones in the optimal solution.

Therefore, a chromosome length that adapts to the problem size would be desirable. To enable a scalable chromosome length, a parameter $Z_{\max}$ was introduced. $Z_{\max}$ denotes the maximum number of zones in a building and thus determines the chromosome length (as each gene corresponds to a zone). Five values for $Z_{\max}$ were selected –

4, 8, 12, 16, and 20 – based on optimal zone counts observed in previous studies (Ruokokoski et al., 2018; Viita-aho, 2019).

Furthermore, since the optimal number of zones is not known in advance, it is ideal for the initial population to contain chromosomes whose number of active zones is distributed across values likely to be near the optimum. To maximize the number of high-quality chromosomes in the initial population, the number of active zones in first-generation chromosomes was drawn from a binomial distribution with parameters $n = Z_{\max}$ and $p = 0.7$. The value of p was selected based on its empirical performance in preliminary trials. For example, if $Z_{\max} = 12$ is used to generate an initial population for a given instance, the resulting values of the probability mass function of active zone counts is shown in Figure 13.

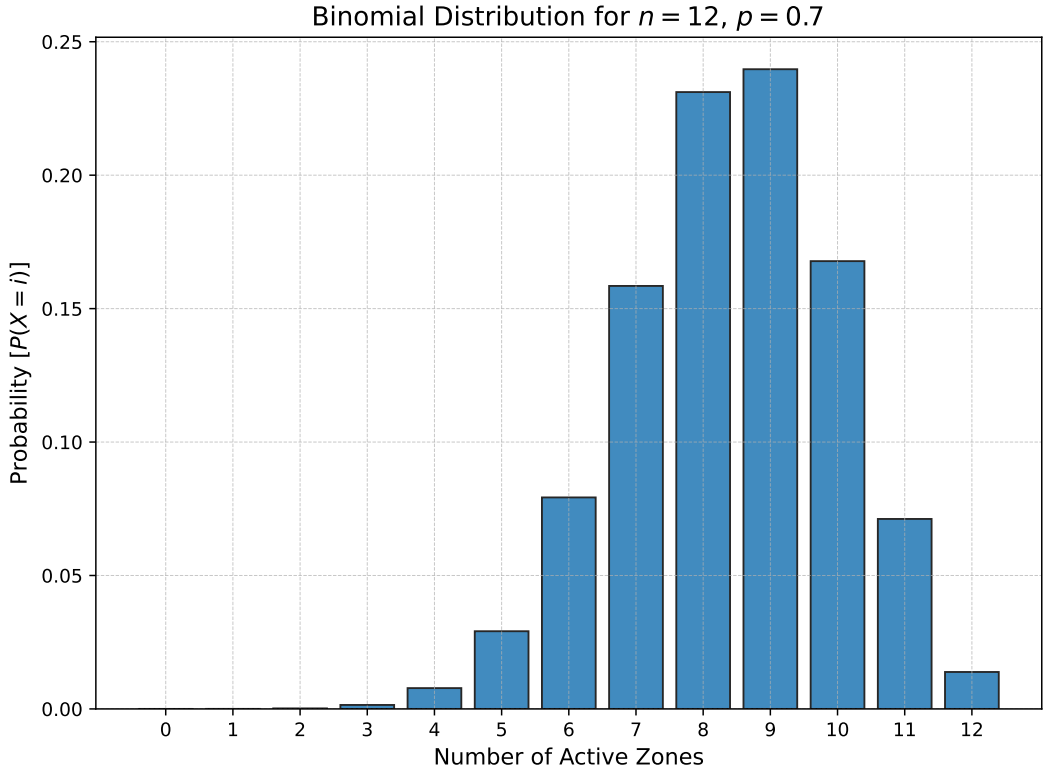


Figure 13: Probability mass function (PMF) of a binomial distribution with parameters $n = 12$ and $p = 0.7$, representing the number of active zones in a chromosome. This distribution models the sampling of active zones in the initial population of chromosomes, where each gene corresponds to a potential zone

The figure illustrates that, for a building with a maximum of 12 zones, the resulting distribution spans a wide range of active zone counts and is centered around values 8 – 9. This distribution supports the genetic algorithm in effectively identifying which zone counts are associated with promising regions of the search space. Empirical trials showed that this method yields a diverse set of initial solutions likely to be near the optimal configuration, thereby providing a solid foundation for the genetic algorithm

to perform effectively. However, the results also demonstrated that selecting a suitable value for $Z_{\max}$ is critical for different instances of the problem.

Another critical parameter was the population size, N_P. As noted by Viita-aho (2019), when zoning arrangements are not systematically explored across different zone counts – or more generally, across different subsets of the full search space – the value of N_P must be significantly increased. For small instances (e.g., a 20-story stack), a large population size was not necessarily required. However, for larger instances (e.g., an 80-story stack), a larger N_P was necessary to ensure sufficient diversity in the solution pool. The population sizes considered were selected from the set $\{300, 500, 1000, 3000\}$.

After drawing the number of active zones, Z_i, from a binomial distribution with parameters $n = Z_{\max}$ and $p = 0.7$, chromosomes for the initial population were constructed as follows. First, $Z_i - 1$ of the upper floors are randomly selected to correspond to active zones, while the remaining $Z_{\max} - Z_i$ are assigned to inactive zones. To ensure that the top floor of the stack is served, one additional active zone with the upper floor N is appended at the end. This procedure yields a chromosome with Z_i active zones, where the associated upper floors are arranged in strictly increasing order and the final zone always covers the top floor of the stack.

After generating the initial population of chromosomes, subsequent generations were produced through crossover and mutation. The maximum number of generations was set to 500. In addition, a stagnation limit of 100 generations was applied: if the best solution found thus far did not improve within 100 generations, the algorithm was considered to have converged and was terminated.

The selection method used was tournament selection with a tournament size of 2. Unlike fitness-proportionate selection, tournament selection is robust to fitness scaling, and a size of 2 is commonly used in the literature (Goldberg & Deb, 1991). Empirical testing during algorithm development confirmed that tournament selection consistently outperformed fitness-proportionate methods, and a tournament size of 2 yielded the best results among the sizes tested.

In addition to tournament selection, elitism was applied to directly preserve the top 5% of chromosomes based on fitness in each generation. This ensured that high-quality solutions were not lost during crossover or mutation operations and allowed the algorithm to converge to near optimal solutions.

The crossover operator used in the HGGA was single-point crossover, with the crossover probability selected from the set {0.7,0.85,0.9,0.95}. Mutation was applied using bitwise mutation (FlipBit) independently to both tags and genes. The mutation mechanism for tags followed Mechanism A described by Abdelkhalik and Darani (2018), where tag mutation occurs independently of gene mutation. This mechanism was selected due to its simplicity and its relatively strong performance in a comparative assessment of different tag evolution approaches conducted by Abdelkhalik and Darani (2018). For reference, all tag evolution approaches considered in the works of Abdelkhalik and Darani are listed in Appendix A.

Mutation probabilities for both genes and tags were selected from the set $\{0.01, 0.02, 0.05, 0.1\}$. During the first 30 generations, a higher mutation rate of 0.2 was used for both tags and genes to promote early exploration of the solution space. A similar strategy was applied by von Schantz and Ehtamo (2022), and the algorithm's performance improved in trials when this rule was used.

Since zones at random indices were swapped between chromosomes during crossover, the operation occasionally resulted in infeasible offspring. In some cases, the chromosome could be repaired; in others, it was penalized with a large constant to prevent it from being selected for reproduction in subsequent generations. A repairable infeasible chromosome was one in which the upper floors were no longer in strictly increasing order. In such cases, the upper floor values could simply be sorted in ascending order to restore feasibility. In contrast, chromosomes were penalized when they contained duplicate upper floors. The same types of infeasible chromosomes could also result from mutations in either the tags or the genes, and these cases were handled in the same way as those arising from crossover.

To improve computational efficiency, a gene bank was implemented to store previously computed fitness values. If a chromosome was encountered during evaluation and was already present in the gene bank, its fitness value was retrieved directly through a lookup. This significantly reduced the computational burden of the algorithm, as the evaluation of the fitness function – being the most expensive part of the process – could often be skipped, especially in convergence phase of the algorithm. The gene bank approach was inspired by Tyni and Ylinen (1999), who reported a 65% reduction in CPU time through a similar mechanism.

The optimal values for the maximum zone count $Z_{\max}$, population size N_P, crossover probability p_c, and mutation probabilities for both tags and genes ($p_{m,t}$ and $p_{m,g}$, respectively) were determined through an extensive grid search, similar to the hyperparameter tuning process used for SA. Table 5 details the candidate values considered during the grid search.

Hyperparameter	**Values**
Maximum zone count ($Z_{\max}$)	{8, 12, 16, 20}
Population size (N_P)	{300, 500, 1000, 3000}
Crossover probability (p_c)	{0.7, 0.85, 0.9, 0.95}
Tag mutation probability ($p_{m,t}$)	{0.01, 0.02, 0.05, 0.1}
Gene mutation probability ($p_{m,g}$)	{0.01, 0.02, 0.05, 0.1}

Table 5: Ranges of values used for each hyperparameter in the HGGA grid search. The parameters include the maximum zone count ($Z_{\max}$), population size (N_P), crossover probability (p_c), tag mutation probability ($p_{m,t}$), and gene mutation probability ($p_{m,g}$)

The hyperparameter grid shown in Table 5 was tested on a representative subset of instances, defined by the key parameters of the problem for stacks: the building's floor

count, and the population per floor. The tested floor count values were $10, 20 \ldots 80$, and the tested population per floor values were $25, 50, \ldots, 200$. Similarly to the SA hyperparameter grid, each combination of hyperparameters and problem instances was run using 10 different random number generator seeds. For each case, the mean and standard deviation of the objective function were computed across the resulting sample.

The optimal hyperparameter configuration for each instance was identified by sorting the configurations in ascending order according to the following priority: (1) mean objective value, (2) standard deviation of the objective value, (3) population size (N_P), and (4) maximum zone count ($Z_{\max}$). As with SA, a mapping structure was developed to generalize the optimal hyperparameter selection to unseen instances, including those not present in the original grid. The rule used to map any instance to its corresponding hyperparameters – by rounding up to the nearest instance in the grid – is identical to the one described in the previous subsection for SA.

Table 6 presents the optimal HGGA hyperparametrizations for floor counts 20, 40, and 60 and population per floor values 50, 100, 150, and 200.

FC	Pop	$Z_{\max}$	N_P	p_c	$p_{m,t}$	$p_{m,g}$
20	50	4	300	0.7	0.01	0.01
20	100	4	300	0.7	0.01	0.01
20	150	4	300	0.7	0.01	0.01
20	200	8	300	0.7	0.01	0.01
40	50	4	300	0.7	0.01	0.01
40	100	8	300	0.9	0.1	0.1
40	150	8	1000	0.7	0.01	0.1
40	200	8	300	0.7	0.01	0.01
60	50	4	300	0.7	0.01	0.01
60	100	8	1000	0.85	0.02	0.1
60	150	12	3000	0.9	0.01	0.05
60	200	20	3000	0.9	0.01	0.1

Table 6: Optimal HGGA hyperparametrizations for selected representative instances. The table is organized by floor count (FC) and population per floor (Pop), with the HGGA hyperparameters displayed to the right: maximum zone count ($Z_{\max}$), population size (N_P), crossover probability (p_c), tag mutation probability ($p_{m,t}$), and gene mutation probability ($p_{m,g}$). Optimal hyperparameters were chosen by ranking the hyperparameter combinations on aggregated results over 10 random number generator seeds with the following priority (1) mean objective value (2) standard deviation of the objective value (3) the population size (N_P) and (4) the maximum zone count (Z_{max})

The table shows that the optimal value of $Z_{\max}$ increases with both floor count (FC)

and population per floor (Pop). For a 20-story stack, $Z_{\max}$ is 4 when the population per floor is below 200, and increases to 8 when the population reaches 200. In the largest instances – a 60-story stack with population per floor values of 150 and 200 – the optimal values of $Z_{\max}$ are 12 and 20, respectively.

A similar trend is observed in the population size N_P: for the smallest instances, good results were achieved with a population size of 300, while for the largest instances, the best performance required the maximum considered size of 3000 chromosomes.

For the crossover probability p_c, 0.7 is the most frequent optimal value, although for the two largest instances, the optimal value is 0.9. For the mutation probability of tags $p_{m,t}$, 0.01 is clearly the most common value. In contrast, for the mutation probability of genes $p_{m,g}$, the optimal value varies across the considered range of 0.01 – 0.1.

4.3.3 Method Validation

In the proposed hybridization method for solving the zoning problem with sky lobbies, it is essential that the inner-level problem of finding the optimal zoning arrangement for individual stacks is solved effectively to ensure the overall quality of the outer-level solutions. Therefore, the effectiveness of the HGGA in solving the stack-level zoning problem was validated by benchmarking its results against those obtained using the dynamic programming method presented in Ruokokoski et al. (2018).

To ensure comparability of solutions, a few adjustments were made to both methodologies. In their method, the constraint requiring that the number of elevators between the largest and smallest (in terms of elevator count) elevator group must differ by no more than two was removed. In the proposed methodology, the lobby area was excluded from the evaluation of a zone's core area. In other words, Equation (20) was used to compute the core area of a stack. Furthermore, only elevator cars with a capacity of 21 passengers were considered, consistent with the setup in Ruokokoski et al. (2018).

Table 7 summarizes the benchmarking results by reporting both the absolute and relative optimality gap between the solutions of the HGGA and the dynamic programming method.

Table 7: Summary statistics of the absolute and relative differences for the sample. Columns represent the minimum (Min), first quartile (Q1), median, mean, third quartile (Q3), maximum (Max), and standard deviation (Std). The first row corresponds to absolute differences, and the second row to relative differences in percentages

	Min	Q1	Median	Mean	Q3	Max	Std
Absolute difference	0.00	0.00	0.00	6.76	0.00	317.30	27.91
Relative difference (%)	0.00	0.00	0.00	0.06	0.00	2.38	0.22

The results in Table 7 show that the HGGA successfully found the globally optimal

zoning arrangement in more than 75% of the instances. When suboptimal solutions occurred, the optimality gaps were generally small.

The mean and standard deviation of the absolute optimality gap were 6.76 m^2 and 27.91 m^2, respectively. For the relative optimality gap, the corresponding values were 0.06% and 0.22%. The largest observed absolute optimality gap was 317.30 m^2, while the largest relative optimality gap was 2.38%.

An analysis of the optimality gaps revealed that the relative optimality gap tends to increase with the floor count of the stack. This relationship is illustrated in Figure 14, which plots the relative optimality gap between the HGGA and the dynamic programming method of Ruokokoski et al. (2018) as a function of floor count. To highlight only those cases where the HGGA failed to find the exact solution, the figure includes only instances in which the solutions differed.

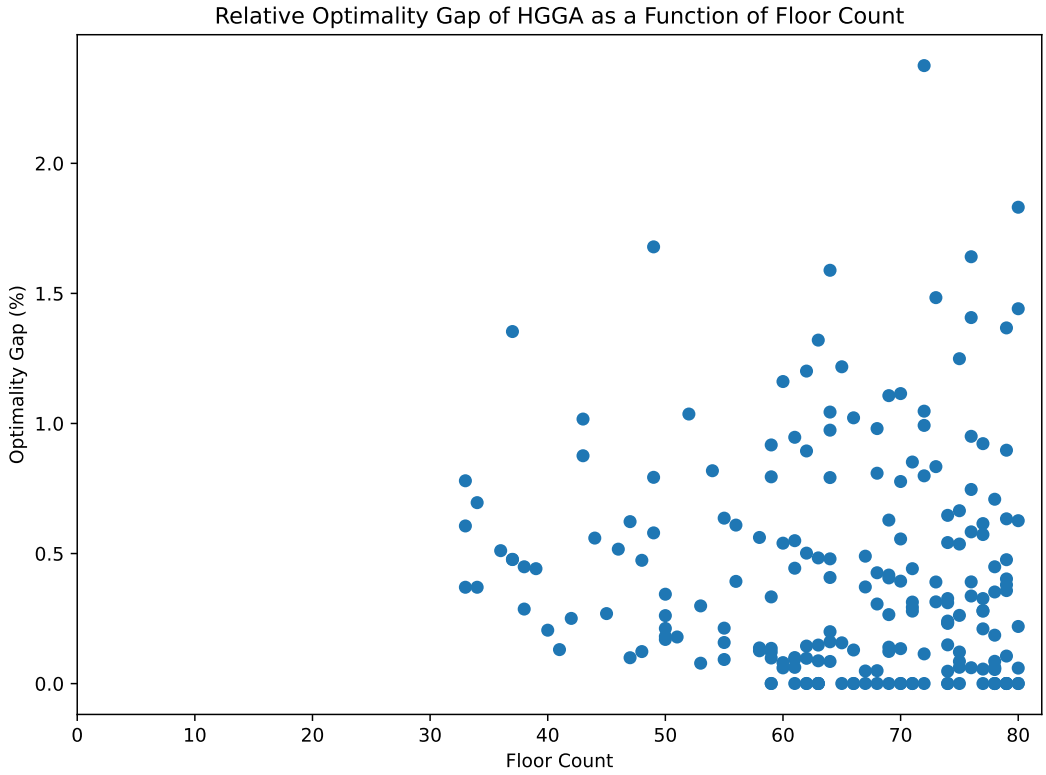


Figure 14: Relative optimality gap between the HGGA and the dynamic programming method of Ruokokoski et al. (2018) as a function of the floor count of the stack. Only instances where the HGGA solution differed from the exact solution are included

The results indicate that the heuristic's performance deteriorates in relative terms as the floor count increases or when the search space becomes very large. Nevertheless, even in these cases, the optimality gap typically remains within 1%. Moreover, in practice, it is unlikely that stacks in optimal solutions would exceed 60 floors, as introducing sky lobbies reduces the elevator core area and often results in smaller

stack sizes. Overall, the HGGA demonstrates strong performance and appears to be a suitable method for determining the optimal zoning configuration for stacks.

4.4 Computational Infrastructure

This subsection provides an overview of the computational environment and methods used to solve the nested optimization problem in this thesis. The hybrid metaheuristic approach of combining Simulated Annealing (SA) and Hidden Genes Genetic Algorithms (HGGA) relies on multiple strategies to reduce the time required for objective function evaluations. The subsections below explain each strategy and conclude with an acknowledgment of the high-performance computing (HPC) resources that were used.

4.4.1 Overview of Computational Techniques

The computational model described in Section 4.3 follows a nested structure, in which both the outer and inner levels involve extensive computation. Running the full optimization procedure in a naive manner – without any dedicated effort to reduce computational overhead – would render the study infeasible due to excessive runtime. The nested nature of the approach implies that the computational cost incurred at the inner level is magnified at the outer level. For this reason, efforts to reduce runtime were initiated from the innermost components of the model.

4.4.2 Precomputation of Zone and Shuttle Solutions

On the inner level of the model, the HGGA is responsible for identifying the optimal zoning arrangement within each stack. In a naive implementation, every proposed zone would require an explicit evaluation, including the computation of round trip time (RTT), which profiling showed to be the primary contributor to total runtime. To address this bottleneck, a relational database was constructed to store precomputed results for individual zones.

Each row in the database corresponds to a unique zone, defined by a combination of four instance-specific parameters: the lower and upper floor numbers, the population per floor, and the floor height. In addition to these, each entry includes four solution-specific parameters: the core area, the elevator car size, the elevator count, and the elevator velocity. This structure enables rapid retrieval of precomputed core area values for any zone configuration encountered during optimization.

A similar table was constructed for shuttle elevator configurations required when evaluating stacks served by a sky lobby. In this case, the lower floor column was replaced with a sky lobby floor field, and the car size field was omitted, as only a single shuttle car size (26-person capacity) was considered.

During chromosome evaluation in the HGGA, where up to Z_{max} zones may be active, each zone's contribution to the objective function can be retrieved through a simple database lookup. This approach substantially reduced evaluation time as thousands of

chromosomes had to be evaluated during a single run of the HGGA. Furthermore, it was crucial for scaling the inner-level optimization to the outer level, where hundreds to thousands of HGGAs were initiated during a single run of full computational model.

4.4.3 Runtime Optimizations

In addition to the precomputation steps described previously, several runtime optimizations were implemented to reduce computational effort during execution. First, a gene bank in the HGGA stored the fitness values of previously encountered chromosomes, thus avoiding repeated evaluations and accelerating convergence. A stagnation limit was also introduced, which terminated the algorithm if no improvement in fitness had been observed for a specified number of consecutive generations. This was particularly beneficial for smaller problem instances, where good solutions were often found within only a few iterations.

At the outer level, the SA procedure employed a similar stagnation limit and maintained a solution bank that stored both the objective value (often referred to as the "energy") and the corresponding zoning solutions for previously visited sky lobby configurations. If the same configuration reappeared, the solution bank provided an immediate lookup of the objective value, circumventing the need for a new evaluation. A further optimization allowed the SA algorithm to reset to the best solution found so far whenever no improvement was observed for 100 consecutive iterations. This reset mechanism helped the search escape unproductive trajectories and converge more rapidly.

4.4.4 Caching Stack and Shuttle Solutions

As explained earlier, a single run of the outer-level SA may trigger thousands of HGGA evaluations. During each iteration, the k sky lobby floors proposed by SA divide the building into $k+1$ stacks, for which the HGGA is used to determine the optimal zoning configuration. Although the use of precomputed zone and shuttle solutions, and the various runtime optimizations (such as gene banks and stagnation limits) significantly reduced the CPU time of individual HGGAs, each evaluation still required several seconds – occasionally up to 20 seconds – due to the scale of computation that was involved. When accumulated over many iterations, these evaluation times became a bottleneck in the convergence of the outer-level SA.

To address this, a second layer of precomputation was introduced in the form of stack and shuttle solution caches. For stack solutions, the optimal zoning configurations were computed in advance for all combinations of stack size (ranging from 8 to 80 floors) and population per floor (spanning 10 to 200 in increments of 10). For each such pair, the HGGA was executed using 10 different random number generator seeds, and the resulting solutions were stored in serialized format using Python's `pickle` module. A corresponding cache was also constructed for shuttle elevator configurations. In this

case, solutions were generated for all combinations of sky lobby floor, upper floor[9], and population per floor, again storing the results in pickle files.

When the SA algorithm evaluated a new set of sky lobby floors, the resulting stacks could be mapped to entries in the stack and shuttle caches. The corresponding zoning and shuttle configurations were then retrieved directly, bypassing the need to rerun the HGGA. This mechanism reduced the evaluation time of a single full model iteration to approximately 20 milliseconds, making it feasible to perform a large-scale hyperparameter grid search and to compute aggregate final results within a few hours.

4.4.5 High-Performance Computing Resources

Despite the reductions in CPU time achieved through the computational innovations described in the previous subsections, the computational scale of the study remained too large to be tractable on a single processor. Therefore, high-performance computing (HPC) resources provided by the Aalto University School of Science's "Science-IT" project were employed for the most computationally demanding parts of the thesis. Specifically, the HPC cluster *Triton*[10] was used.

Triton was first employed for the hyperparameter tuning of the HGGA. A representative and comprehensive batch of instances was run to explore the sensitivity of the objective value to different hyperparameter values. After identifying the most critical hyperparameters and defining the corresponding hyperparameter grid, Triton was used again to perform a full grid search. This was implemented as an array job, with each job allocated to a single CPU node and 800 MB of memory.

Triton was also used for the hyperparameter tuning of SA, following the same procedure as with HGGA. A representative batch of instances was used to identify the most influential hyperparameters, after which a full grid search was conducted. As before, the search was implemented as an array job. Each job was allocated to a single CPU node, with 2 GB of memory due to higher memory requirements.

Third, Triton was used to compute and populate the cache for stack and shuttle solutions across all combinations of floor counts, population counts, and seeds. This step, described in Section 4.4.4, was again implemented as an array job, with each job requiring one CPU node and 800 MB of memory.

Finally, Triton was utilized to compute the final results. A large batch of instances was defined, covering all combinations of floor counts, population per floor values, and seeds. Due to the use of the precomputed stack and shuttle solutions from the previous step, the final results batch could submitted as a single job allocated to one CPU node and 4 GB of memory.

[9] The upper floor is relevant because shuttle elevators were assumed to serve the population from the sky lobby to the top of the stack.

[10] A technical overview of the cluster is available at https://scicomp.aalto.fi/triton/overview/

Following the guidance of research engineers at Aalto Scientific Computing, each job in the array was designed to require 1 – 10 hours of runtime. This duration was optimal for the SLURM queuing system used to allocate jobs across available CPU nodes. Since most batch jobs involved different combinations of problem parameters, job durations were estimated by trial runs. The results were used to group problem combinations such that the execution time per job aligned with the recommended time window.

Before concluding this subsection, Table 8 provides an overview of the most computationally intensive steps executed on Triton. For each array job, we report the CPU time requested per job, the memory requested per job, the total number of jobs in the array, and finally, the total runtime both in hours and in days (with Runtime (h) = Time $\times$ Jobs and Runtime (d) = Runtime (h)/24).

Table 8: Summary of computational steps executed on Triton, detailing the CPU time and memory requirements for a single job, the number of jobs in the array, and the total runtime of the array job in both hours and days. The "Total" row reports the sum of "Jobs" and the sum of the total runtime, illustrating the overall HPC usage

Step	**Time (h)**	**Mem (GB)**	**Jobs**	**Runtime (h)**	**Runtime (d)**
HGGA Grid Search	1	0.8	256	256	10.67
SA Grid Search	5	2	90	450	18.75
Precomputation	3	0.8	10	30	1.25
Final Results	2	4	1	2	0.08
Total	–	–	357	738	30.75

The SA grid search required the greatest total runtime (450 hours), making it the most computationally demanding step. In contrast, the final results calculation, benefiting from extensive caching, required only 2 hours in total and was therefore the least time-consuming.

When aggregated, these tasks required a total of 738 CPU hours (approx. 31 days). Executing them on a single CPU core would have required a month of dedicated runtime, whereas Triton's distributed resources allowed these tasks to be completed in parallel and thus in a fraction of the wall-clock time. This clearly justifies the use of HPC resources for this thesis.

5 Results

The results section is organized as follows. Finally, Section 5.1 defines the scope of the problem and outlines the main assumptions in this work. Section 5.2 examines the optimal number of sky lobbies across a range of building heights and population densities, highlighting general trends and outlier configurations. Section 5.3 investigates the optimal placement of sky lobbies for different combinations of floor count and population per floor, under two- and three-lobby configurations. Section 5.4 quantifies the relative core area savings achieved through the use of sky lobbies by comparing optimized solutions to baseline cases without them. Finally, Section 5.5 evaluates the elevator core efficiency by analyzing the ratio of elevator core area to total office area, offering practical insight into the impact of different sky lobby configurations on usable space. Finally, Section 5.6 provides results on runtime.

5.1 Scope and Assumptions

Before computing results, a set of assumptions had to be made to scope the problem and to make it computationally tractable. Furthermore, ISO standards set restrictions on the zoning problem. What follows is a list of assumptions that was made for the generation of the computational results hereinafter.

1. The building type is an office building.
2. The traffic type is up-peak traffic.
3. The building has 0 – 3 sky lobbies.
4. The floor height is 3.3 m.
5. The building population is assumed to follow a uniform distribution (population per floor is a constant number).
6. The floor count of a stack is between 8 – 80 floors.
7. Each elevator stack has an entrance floor that all elevators in the stack serve. For the first stack, the entrance floor is floor 0. For all subsequent stacks, the entrance floor is the respective sky lobby.
8. Zones within a stack only share the entrance floor.
9. Zones within a stack are contiguous.
10. All elevators are single-deck.
11. The maximum allowed rated speed for any elevator is 10 m/s.
12. Possible kinematic values (nominal speed, acceleration, and jerk) for elevators are presented in Table 9.
13. Elevator door and transfer timings are presented in Table 10.

14. Elevator car sizes follow ISO standards and the set of considered elevator sizes are presented in Tables 11 and 12.
15. Shuttle elevators have a rated capacity of 26 persons.
16. RTT is calculated with the method by Roschier and Kaakinen (1978).
17. Elevators within the same elevator group share the same characteristics (capacity and nominal speed).

Table 9 presents the possible kinematic parameter combinations for elevators, as defined in KONE's Building Traffic Simulator (BTS). These parameters include nominal speed, acceleration, and jerk, which together characterize the velocity profile of an elevator. The choice of kinematic parameters directly affects the travel time of elevators and thereby influences key traffic performance indicators such as RTT, INT, and NTT.

Table 9: Kinematic parameter options for elevators used in KONE's Building Traffic Simulator (BTS). The table lists feasible combinations of nominal speed, acceleration, and jerk values, which define the elevator velocity profiles and influence travel times

Nominal speed [m/s]	**Acceleration [m/s^2]**	**Jerk [m/s^3]**
1.0	0.8	1.2
1.6	0.8	1.2
2.0	0.8	1.2
2.5	1.0	1.6
3.0	1.0	1.6
3.5	1.0	1.6
4.0	1.0	1.6
5.0	1.0	1.6
6.0	1.0	1.6
7.0	1.0	1.6
8.0	1.0	1.6
9.0	1.0	1.6
10.0	1.0	1.6

To simplify presentation, two groups of elevators based on their capacity (in persons) are defined:

$$E_1 = \{13,\ 17,\ 18,\ 21\},$$
$$E_2 = \{24,\ 26\}.$$

For each elevator group, the door closing time, door opening time, passenger transfer time, photocell delay, and start delay are identical. These parameters directly influence

the RTT and INT of the groups, and thus play a key role in determining the feasibility of assigning a given elevator group to a zone. Table 10 summarizes the shared parameter values for each group.

Table 10: Door operation and transfer timing parameters for elevator groups $E_1 = \{13, 17, 18, 21\}$ and $E_2 = \{24, 26\}$, as used in KONE's Building Traffic Simulator (BTS). These parameters influence the total transfer time per stop and, consequently, the RTT and INT of each elevator group

Parameter	E_1	E_2
Door closing time [s]	3.1	3.4
Door opening time [s]	1.4	1.4
Passenger transfer time [s]	1.0	0.95
Photocell delay [s]	0.9	0.9
Start delay [s]	0.7	0.7

In addition, the shaft and car dimensions for elevator groups E_1 and E_2 are presented in Tables 11 and 12, respectively. These dimensions influence the core area calculations and thus affect the car size selected by the algorithm for the elevator group assigned to a given zone.

Table 11: Shaft and car dimensions for the elevators in group E_1

Parameter	**13 pers.**	**17 pers.**	**18 pers.**	**21 pers.**
Shaft area [m^2]	5.28	5.98	6.4925	6.75
Shaft width [mm]	2400	2600	2650	2700
Shaft depth [mm]	2200	2300	2450	2500
Car width [mm]	1700	2000	2000	2100
Car depth [mm]	1400	1400	1500	1600

Table 12: Shaft and car dimensions for the elevators in group E_2

Parameter	**24 pers.**	**26 pers.**
Shaft area [m^2]	7.5	7.8
Shaft width [mm]	3000	3000
Shaft depth [mm]	2500	2600
Car width [mm]	2350	2350
Car depth [mm]	1600	1700

In the next section, results are presented from solving the optimization problem in

Equation (42). The problem is solved using the computational model introduced in Subsection 4.3. All results are obtained under the assumptions outlined above.

5.2 Sky Lobby Count

One of the key design questions when using sky lobbies in high-rise buildings is determining their optimal number. Naturally, this depends on several factors. Among these, the two most critical are the total floor count of the building and the population per floor.

To investigate this, the optimal number of sky lobbies was determined for different combinations of floor count and population density by comparing the resulting core areas of solutions with varying numbers of sky lobbies against a baseline solution without sky lobbies. Figures 15, 16, and 17 present the results for buildings with 40 – 80, 80 – 140, and 140 – 200 floors, respectively.

In each heatmap, the cells correspond to a specific combination of floor count and population per floor. The optimal sky lobby count is indicated by the number shown within each cell and is further highlighted using color coding for visual clarity.

Figure 15 reveals several patterns regarding the optimal number of sky lobbies in buildings with 40 to 80 floors. The optimal sky lobby count ranges from zero to three, depending on key parameters. It generally increases with both floor count and population density, although population density appears to have a greater influence.

For instance, when the population per floor reaches 200, three sky lobbies are optimal across the entire range of floor counts. In contrast, even at the maximum of 80 floors, fewer (one or two) sky lobbies may suffice if the population density is moderate. These findings underscore that population density can outweigh building height when determining how many sky lobbies are needed.

When the population per floor is very low (10–20 persons), sky lobbies are not always necessary. For instance, with a population of 10 persons per floor, sky lobbies become beneficial only in buildings with more than 75 floors. However, a small increase in population density changes this. At 20 persons per floor, sky lobbies become advisable from 50 floors upward. When the population per floor reaches 40 or more, sky lobbies should be used in all cases.

The figure exhibits several outlying solutions. For the floor count – population per floor pairs (46, 70), (52, 60), and (56, 50), the optimal solution includes three sky lobbies, even though the neighboring configurations suggest only two. Another notable exception is the solution for a 78-story building with a population of 10 persons per floor: it recommends two sky lobbies, while the adjacent instances – 76 floors and 80 floors – suggest zero and one sky lobby, respectively, for the same population density.

One possible explanation is that these outliers correspond to local minima. Another is that they reflect the discrete and non-smooth nature of the search space, where small

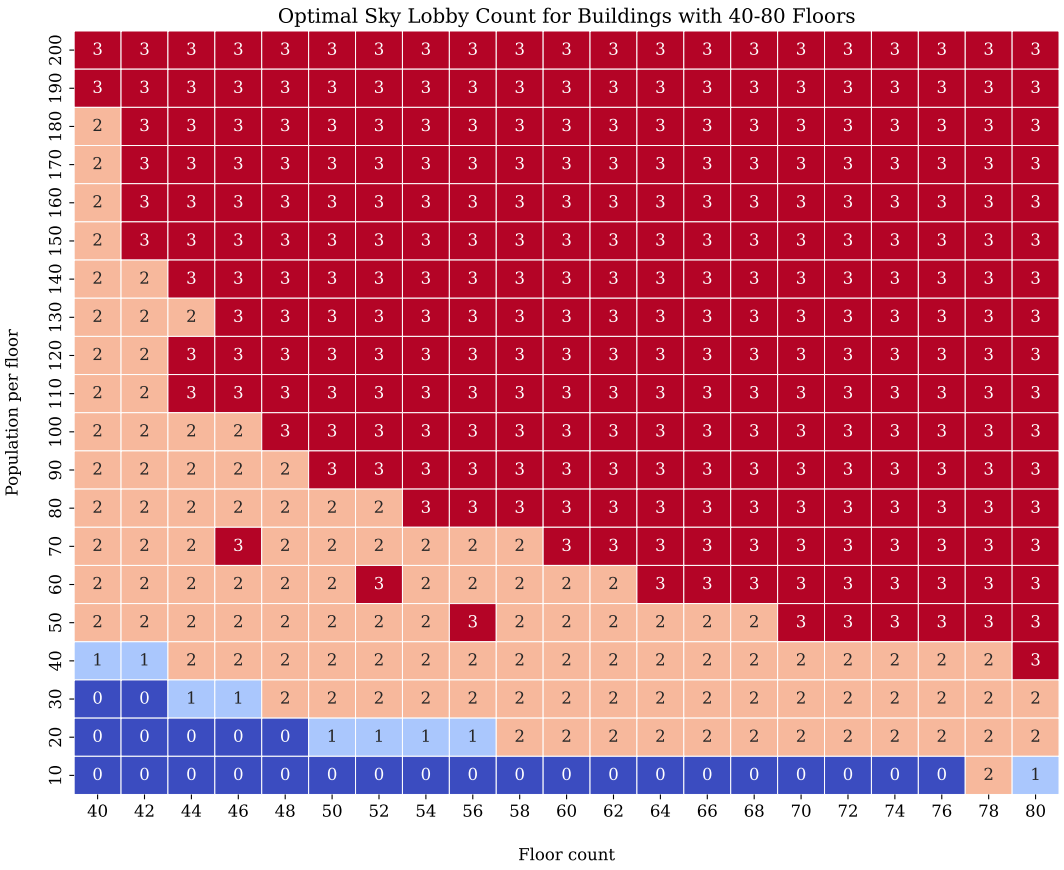


Figure 15: Optimal number of sky lobbies for buildings with 40 – 80 floors, as a function of floor count and population per floor. The number in each cell was determined as the mode of the solutions obtained across 10 runs with different random seeds

input changes can lead to abrupt shifts in the solution. For example, the addition of a single person on one floor might necessitate an extra elevator in a group, resulting in a substantial increase in core area and thus a significantly worse objective function value. As a result, globally optimal solutions may sometimes exhibit unintuitive behavior.

Figure 16 illustrates how the optimal number of sky lobbies varies with population per floor and total floor count in buildings with 80 – 140 floors. With a very low population per floor (10 persons), one sky lobby is preferred up to approximately 86 floors, after which two sky lobbies are optimal up to around 130 floors, followed by three sky lobbies beyond that. When the population per floor increases to 20, two sky lobbies are generally optimal up to 110 floors, after which three are required. For a population of 30 persons per floor, the switch to three sky lobbies occurs earlier, at around 90 floors. From a population of 40 persons per floor and above, three sky lobbies are optimal across the entire considered range of floor counts.

Some exceptions to these general trends can be observed. For instance, the solution

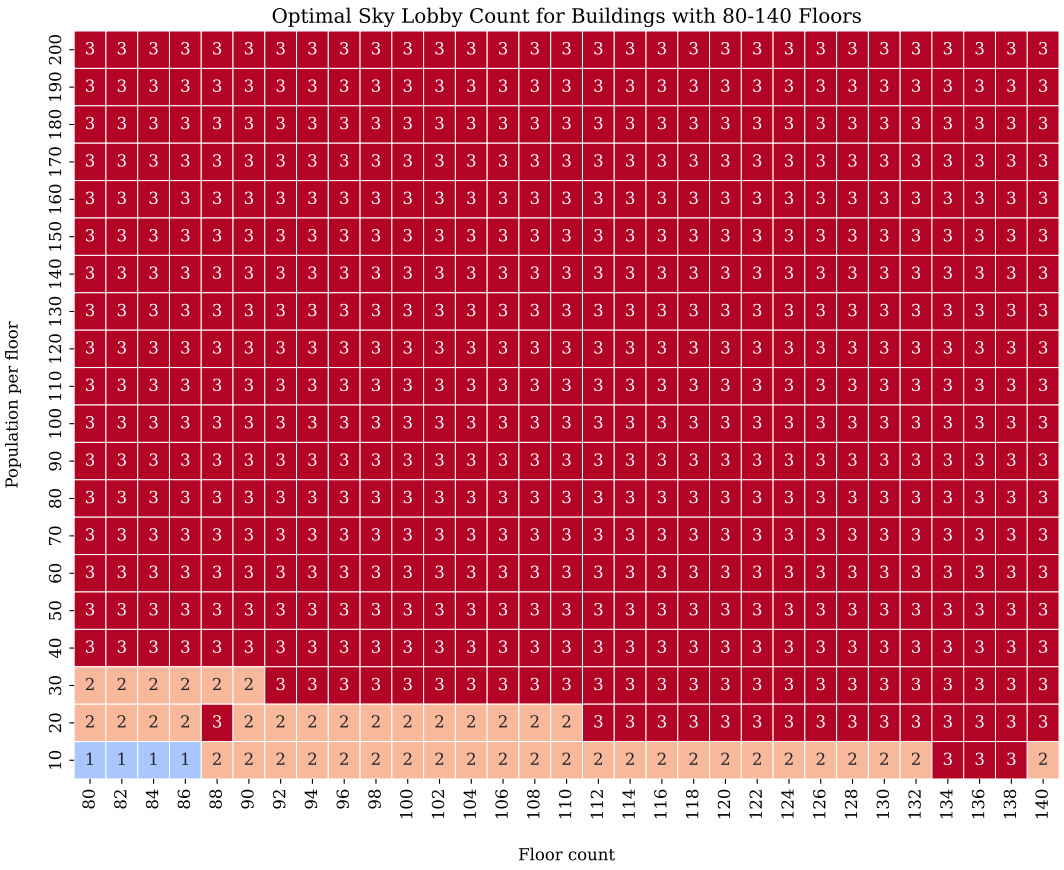


Figure 16: Optimal number of sky lobbies for buildings with 80 – 140 floors, as a function of floor count and population per floor. The number in each cell was determined as the mode of the solutions obtained across 10 runs with different random seeds

for 88 floors and a population of 20 persons per floor recommends three sky lobbies, even though the next instance with the same population requiring three sky lobbies does not appear until 112 floors. Another outlier is the solution for 140 floors and 10 persons per floor, which suggests two sky lobbies, despite the previous three instances (134, 136, and 138 floors) recommending three sky lobbies for the same population density.

Figure 17 shows the optimal number of sky lobbies for various combinations of floor count and population per floor in buildings with 140 to 200 floors. Almost all solutions suggest three sky lobbies. The only exceptions are a few instances with a population of 10 persons per floor and floor counts between 140 and 166, where two sky lobbies are occasionally preferred. Among these, there are some outlying solutions that revert to three sky lobbies, indicating that the objective values for configurations with two or three sky lobbies are very close among these instances. These alternations further suggest that small differences in input parameters can lead to shifts between nearly equivalent optima.

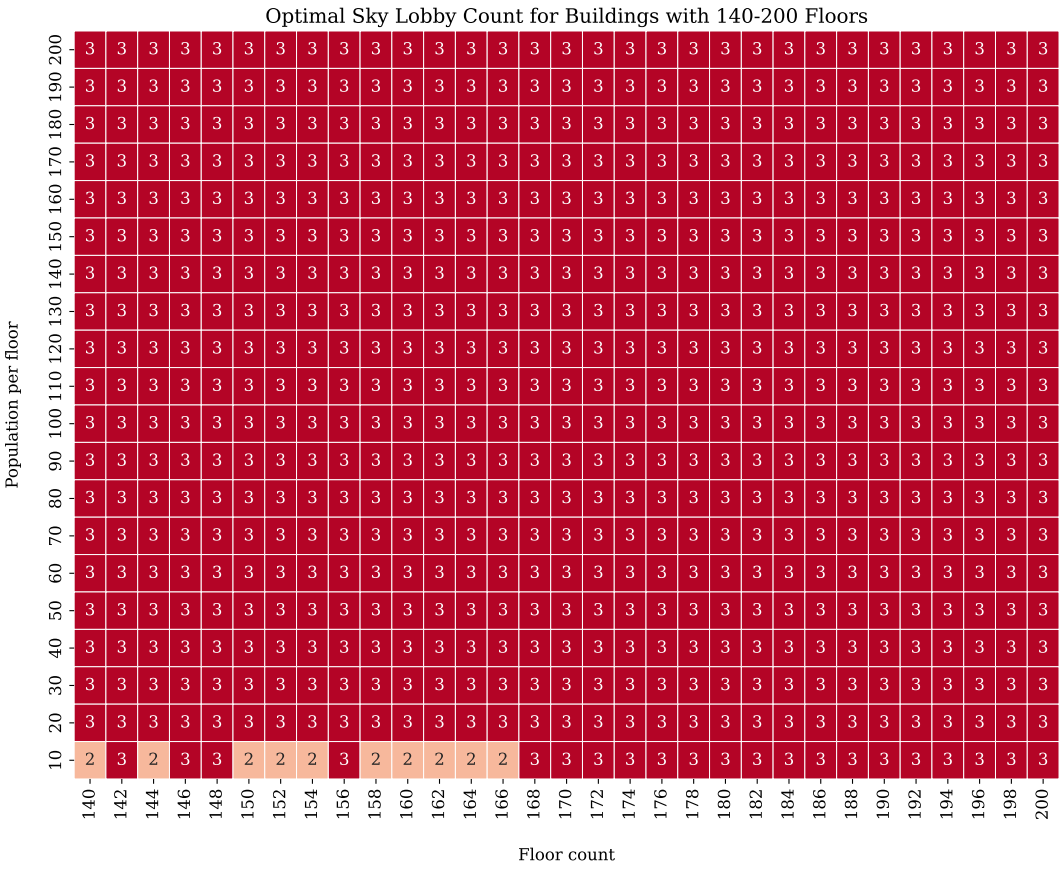


Figure 17: Optimal number of sky lobbies for buildings with 140 – 200 floors, as a function of floor count and population per floor. The number in each cell was determined as the mode of the solutions obtained across 10 runs with different random seeds

Overall, based on the above results in Figures 15, 16, and 17, a few generalizations can be made. Buildings with very low population per floor (10–20 persons) do not always require a sky lobby. Furthermore, they can be served with one to two sky lobbies up to more than 100 floors. Buildings with relatively low population density (60 persons per floor or less) typically benefit from having two sky lobbies. Buildings with very high population density (190 – 200 persons per floor), or those with more than 60 floors and a moderate to high population density (60+ persons per floor), should always have three sky lobbies. This last observation suggests that four or more sky lobbies could be advantageous for very tall buildings with large populations.

As noted in Section 2.3.3, a wide range of recommendations has been made regarding when sky lobbies should be introduced. Barney and Al-Sharif (2015) stated that buildings up to 60 floors can be served using single-deck elevators without the need for sky lobbies. Similarly, Fortune (1985) claimed that sky lobbies become necessary only beyond 60 floors. While these statements may reflect feasibility in terms of performance criteria, the results of this thesis suggest otherwise. Specifically, for

60-floor buildings, two to three sky lobbies are optimal whenever the population per floor exceeds 10, which is is practically always the case. Therefore, these findings indicate that sky lobbies may need to be introduced at lower building heights than commonly assumed.

Among the reviewed sources, Schroeder (1989) proposed the lowest threshold, suggesting that sky lobbies should be considered for buildings between 40 and 50 stories. The results of this study lend support to Schroeder's position. Unless population density is exceptionally low, two to three sky lobbies are recommended even for buildings starting at 40 floors.

5.3 Sky Lobby Placement

After determining the number of sky lobbies for a building, the next design question concerns their placement. It remains an open question which floors are optimal for locating sky lobbies. Current industry practice relies heavily on the designer's expertise and informal rules of thumb. In the worst cases, architects may decide the placement without consulting the elevator manufacturer (Viita-aho, 2019). The key parameters influencing the optimal placement of sky lobbies are the building's floor count and the population per floor. Therefore, the optimal placement was studied across various combinations of these parameters.

As shown in Figure 15, the optimal number of sky lobbies in smaller buildings with 40 to 80 floors varies significantly, ranging from zero to three. However, the majority of solutions recommend either two or three sky lobbies. Consequently, the optimal placement was assessed under two configurations: (i) two sky lobbies, and (ii) three sky lobbies. For the two-lobby case, population per floor values of 20, 40, 60, and 80 were used. For the three-lobby case, population values of 20, 50, 100, and 200 were selected. These values were chosen to illustrate the variability in optimal placement patterns.

Figure 18 shows the optimal floor locations for two sky lobbies in buildings with 40 to 80 floors. The population per floor increases across the subfigures, starting from 20 persons per floor in the top-left and proceeding to 40, 60, and 80 persons per floor in reading order (left-to-right, top-to-bottom).

The figure reveals several noteworthy patterns in the placement of sky lobbies. First, the lowest sky lobby is generally placed at a relatively low level in the building, regardless of population density. For example, when the population per floor is 20, the first sky lobby is located around floor 10 across the entire range of floor counts. Even when the population per floor reaches the maximum considered value of 80 persons, the first sky lobby remains relatively low – for instance, near floor 20 in an 80-story building.

Second, a distinct shift in the placement of the first sky lobby occurs when the total floor count exceeds approximately 60 – 65 floors. For example, in the bottom-left subfigure (population per floor of 60), the first sky lobby is located at about floor

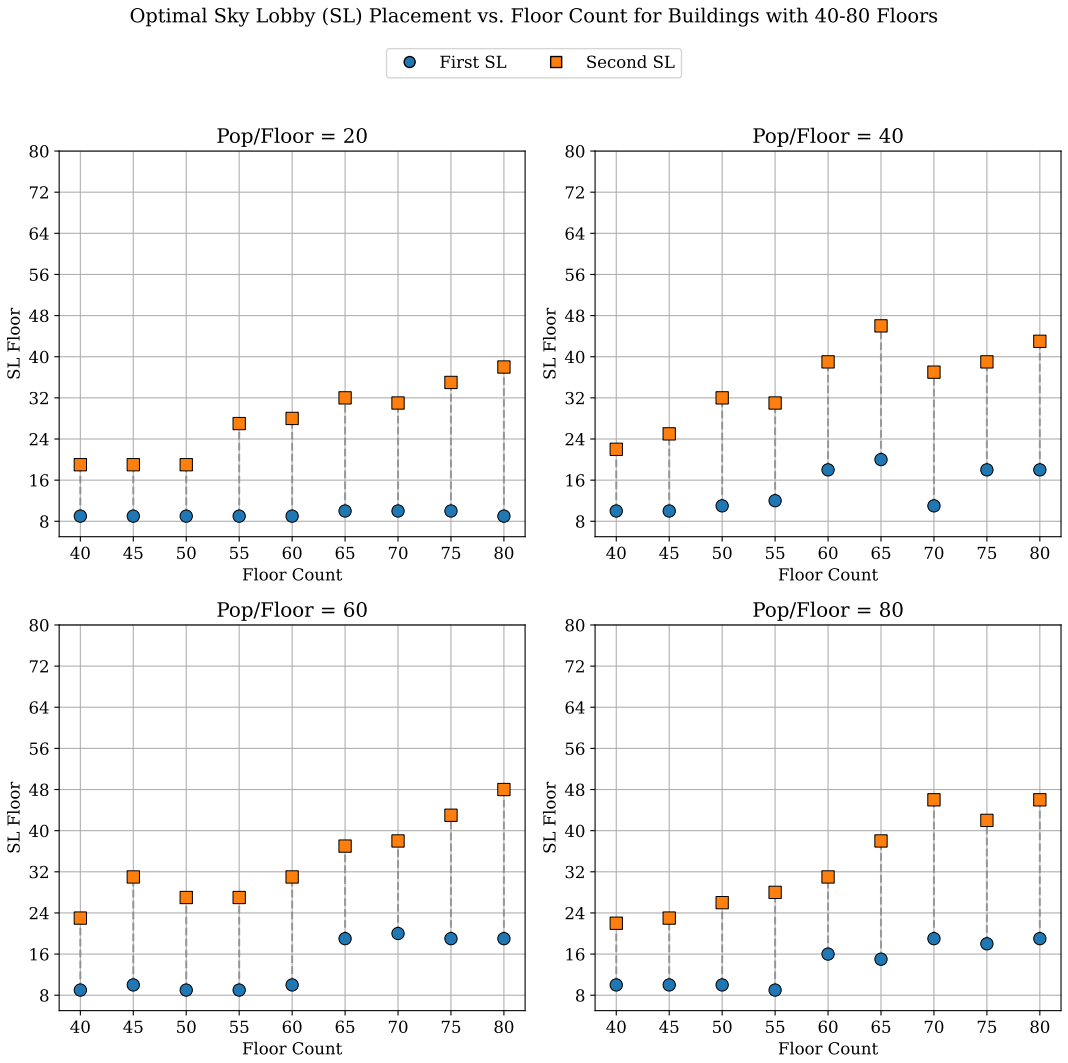


Figure 18: Optimal sky lobby floor locations for buildings with 40 to 80 floors with two sky lobbies. The subfigures correspond to increasing population per floor: 20 (top-left), 40 (top-right), 60 (bottom-right), and 80 (bottom-left) persons per floor

10 when the building has 60 floors, but it jumps to floor 20 when the floor count increases to 65. Beyond this point, its position remains roughly constant up to 80 floors. A similar jump can be observed in the top-right and bottom-right subfigures as well.

Third, the vertical distance between the first and second sky lobbies tends to increase with floor count. In the top-right subfigure (population per floor of 40), the spacing between the first and second sky lobbies is approximately 15 floors for buildings with 40 and 45 floors, but increases to nearly 30 floors when the building has 70 floors. This trend is also visible in the other subfigures.

Finally, the placement of the second sky lobby does not follow a strictly increasing pattern with respect to floor count. For instance, in the top-right subfigure with a

population of 40 persons per floor, the second sky lobby is placed on floor 47 in a 65-story building. However, for the next considered floor count of 70, it is positioned on floor 38 – representing an almost 10-floor downward shift. After this, the second sky lobby floor increases again with floor count, but does not return to the previous maximum level of 47.

Figure 19 shows the optimal floor locations for three sky lobbies in buildings with 40 to 80 floors. The population per floor increases across the subfigures: 20, 100, 150, and 200 persons per floor, starting from the top-left and proceeding left-to-right and top-to-bottom.

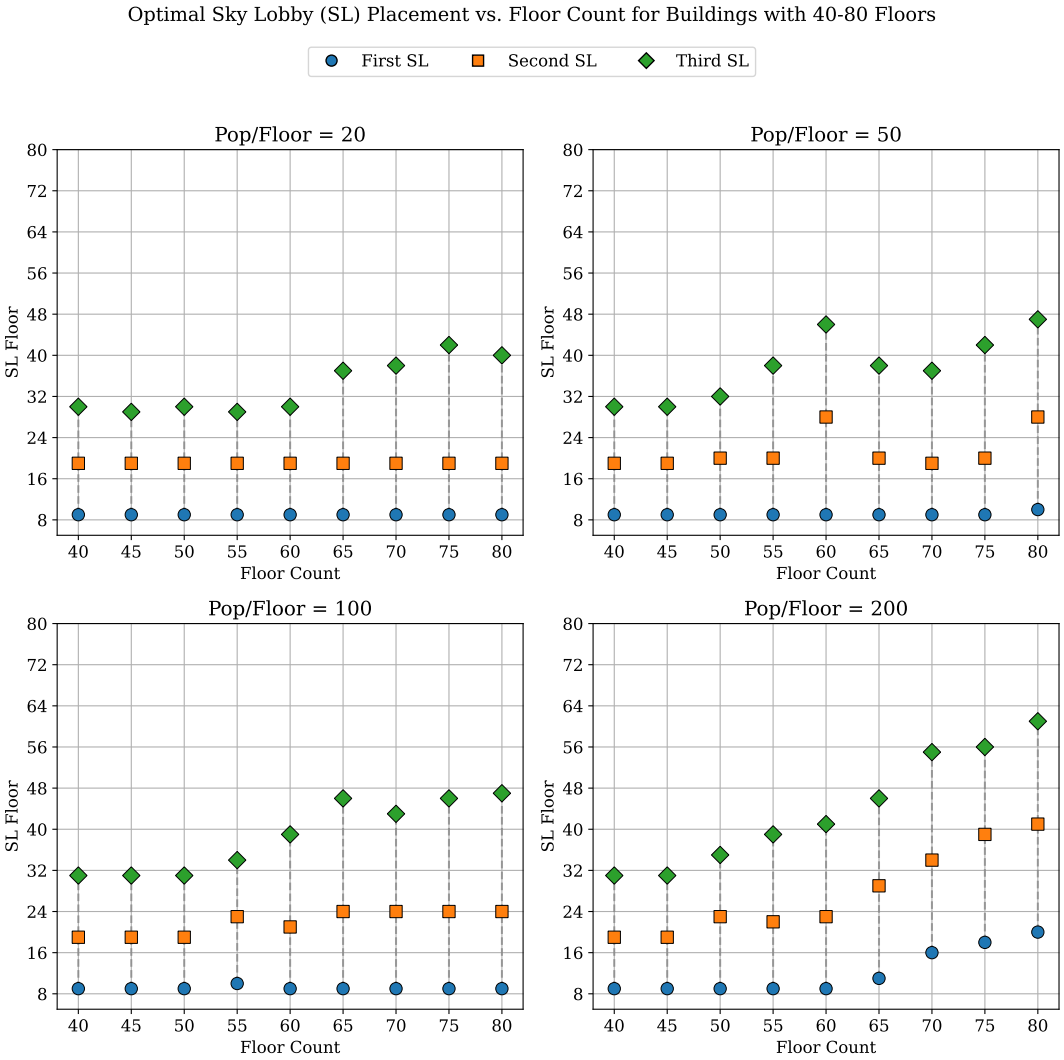


Figure 19: Optimal sky lobby floor locations for buildings with 40 to 80 floors with three sky lobbies. The subfigures correspond to population per floor values of 20 (top-left), 100 (top-right), 150 (bottom-right), and 200 (bottom-left) persons per floor

Similar trends are observed as in Figure 18, where two sky lobbies were used. There appears to be a cutoff point around 60 – 65 floors, beyond which the placement of

sky lobbies changes significantly. For example, in the top-left subfigure (population per floor of 20), the third sky lobby shifts from approximately floor 30 to floor 40, while the first and second sky lobbies remain relatively stable across the floor count range. In contrast, at the highest considered population per floor of 200 (bottom-right subfigure), all three sky lobbies increase in height as the floor count grows. At 60 floors, the first, second, and third sky lobbies are located approximately at floors 10, 25, and 40, respectively. By 80 floors, these placements have risen to about floors 20, 40, and 62.

Another observation is that the first sky lobby is generally placed very low in the building, except in cases where both the floor count exceeds 70 and the population per floor reaches the maximum of 200. In these cases, the first sky lobby shifts upward – from around floor 10 to floors 16, 20, and 22 for buildings with 70, 75, and 80 floors, respectively. This suggests that at sufficiently high population densities, sky lobbies must be placed higher in the building, despite the associated increase in core area due to taller elevator shafts.

As in the previous figure, some outlying solutions are present, where sky lobby floor levels do not increase monotonically with floor count. For instance, in the top-right subfigure, the configuration for a 60-floor building places the second and third sky lobbies on floors 30 and 47. However, in the 65-floor configuration, they are positioned lower – on floors 20 and 39 – even though the building is taller by five floors.

As demonstrated in Figures 16 and 17, buildings with more than 80 floors predominantly require three sky lobbies in the optimal solutions. Therefore, only configurations with three sky lobbies were considered for the taller building classes. Population per floor values of 20, 100, 150, and 200 were selected to capture variability in the solutions. Since total building population can vary substantially, these cases offer a representative set of examples for different practical scenarios and audiences.

Figure 20 shows the optimal floor locations for three sky lobbies in buildings with 80 to 140 floors. The subfigures correspond to increasing population per floor values of 20, 100, 150, and 200 persons, starting from the top-left and proceeding left-to-right and top-to-bottom.

The figure illustrates that in buildings with 80 to 140 floors, the placement of sky lobbies generally increases with floor count across all considered population densities. However, under low population density (20 persons per floor), the first sky lobby is consistently placed very low in the building – around the minimum of 8 floors – regardless of total floor count. This reflects the principle that, when feasible within performance criteria, sky lobbies should be positioned as low as possible to avoid excessively tall elevator shafts.

For higher population densities (100 and 200 persons per floor), the patterns in sky lobby placement are more regular. The sky lobby floors increase steadily with floor count, and the vertical spacing between sky lobbies expands as the building height increases. When the population per floor is 200, the placement closely follows the structure of the initial solution, in which sky lobbies divide the building into evenly

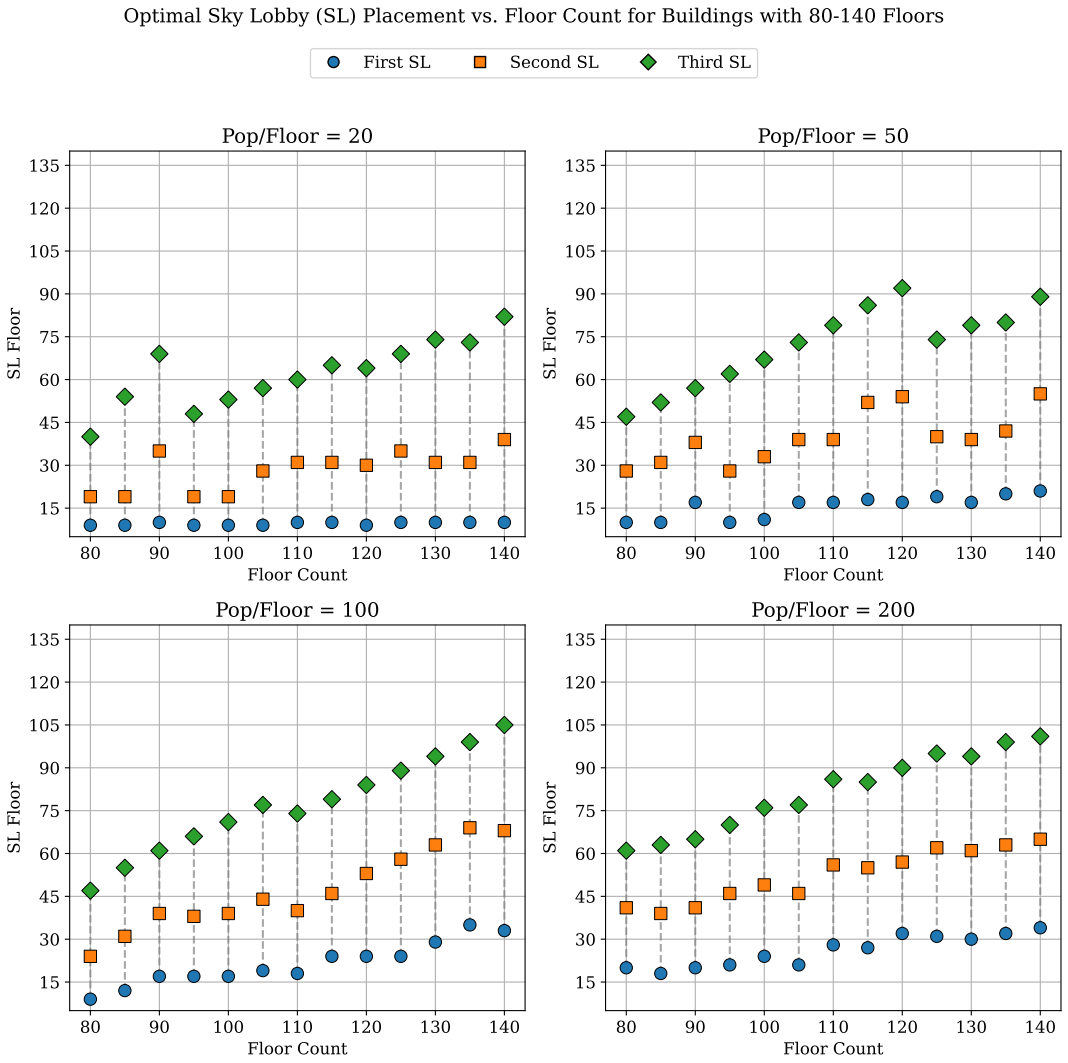


Figure 20: Optimal sky lobby floor locations for buildings with 80 to 140 floors with three sky lobbies. The subfigures correspond to population per floor values of 20 (top-left), 100 (top-right), 150 (bottom-right), and 200 (bottom-left) persons per floor

spaced stacks using the rule $S_i \cdot N/(k+1)$, where N is the floor count, k is the number of sky lobbies, and $S_i \in \{1, \ldots, k\}$ denotes the index of the sky lobby. For example, in a 60-floor building, the first, second, and third sky lobbies align closely with the respective values (20, 40, 60). Similarly, for a building with 140 floors and a population of 200, they are located near (35, 70, 105).

In contrast, for lower population densities (10 and 50 persons per floor), some outlying configurations are observed. For instance, in the top-right subfigure (population per floor of 50), there is a marked shift in sky lobby placement between 120 and 125 floors, particularly for the second and third sky lobbies. At 120 floors, these are located around floors 55 and 92, but at 125 floors, both shift downward to approximately floors 40 and 75. They then remain near these levels up to 135 floors, before the 140-floor

configuration returns to a pattern more similar to that seen at 115 floors—despite a 25-floor increase. A potential explanation is that with lower population densities, there are more local minimums in different regions of the search space.

Figure 21 shows the optimal floor locations for three sky lobbies in buildings with 140 to 200 floors. The subfigures correspond to increasing population per floor values of 20, 50, 100, and 200 persons, starting from the top-left and proceeding left-to-right and top-to-bottom.

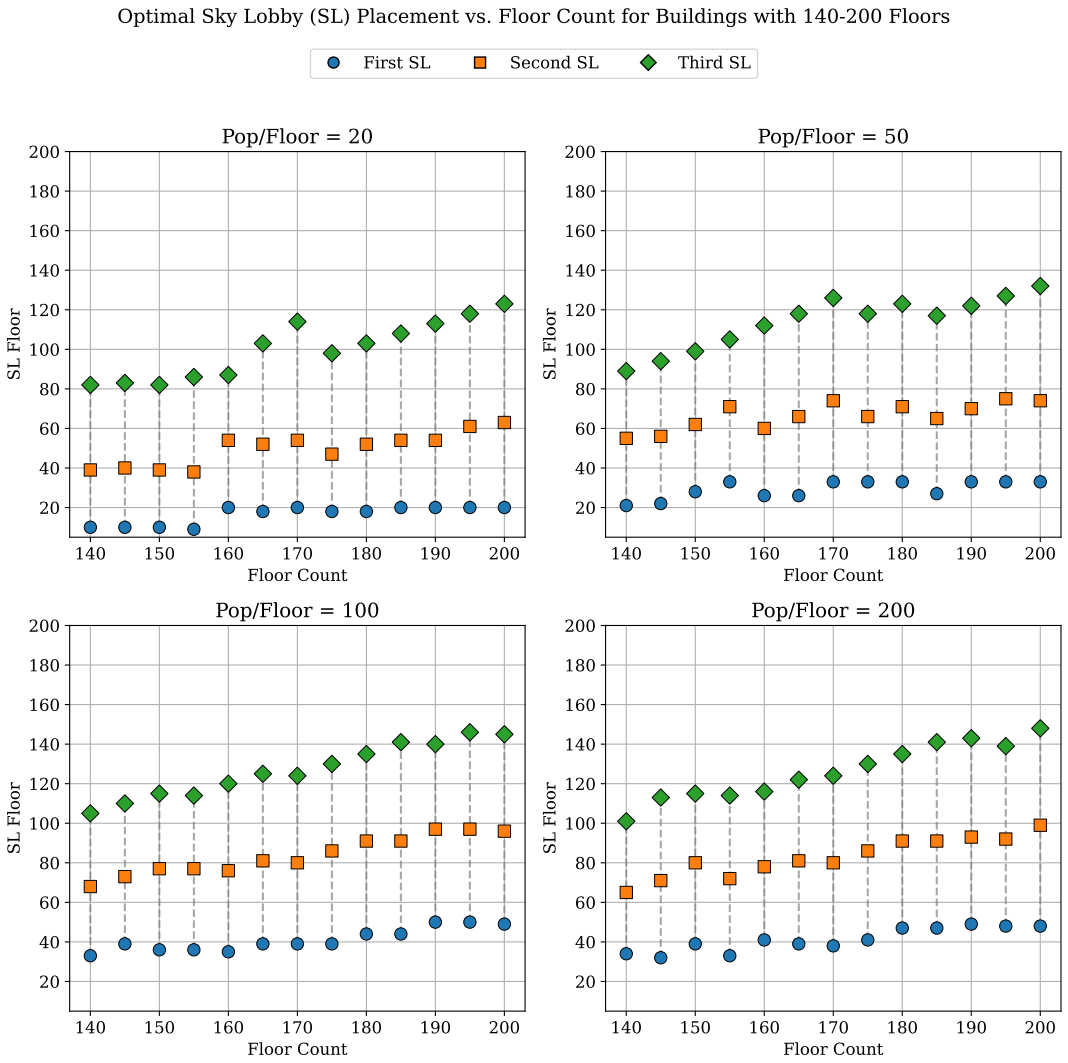


Figure 21: Optimal sky lobby floor locations for buildings with 140 to 200 floors with three sky lobbies. The subfigures correspond to population per floor values of 20 (top-left), 50 (top-right), 100 (bottom-right), and 200 (bottom-left) persons per floor

The figure demonstrates that sky lobby placement patterns become more pronounced as the building height increases to the largest class of 140 to 200 floors, and there are less outlying solutions. In general, the first and second sky lobbies are positioned relatively low in the building. Rather than shifting the lower sky lobbies upward with

increasing floor count, the adjustment is primarily made by moving the third sky lobby higher. This effect is particularly evident at lower population densities. For instance, when the population per floor is 20 (top-left subfigure), and the floor count increases from 160 to 200, the first sky lobby remains fixed at floor 20, while the third sky lobby moves upward from approximately floor 85 to 122 – a shift of nearly 40 floors. Similar trends can be observed in the other subfigures.

As in earlier figures, when the population per floor is 200, sky lobby placements tend to align closely with the initial solution based on evenly divided stacks. For example, at the maximum floor count of 200, the first, second, and third sky lobbies are located almost exactly at floors 50, 100, and 150, respectively. This suggests that, as population density increases, the optimal placements of sky lobbies converge toward evenly spaced stacks.

5.4 Core Area Savings

In this subsection, the potential core area savings from using sky lobbies is evaluated in two distinct ways. First, the resulting core area of a building with sky lobbies is compared to that of a single-stack configuration optimized using the HGGA. Second, this result is compared to a configuration derived from industry rules of thumb proposed by practitioners.

5.4.1 Comparison to a Single Stack

The primary motivation for introducing sky lobbies in a building is their potential to reduce the elevator core area. However, as discussed earlier, the extent of these potential savings has remained an open question. To address this, the core area of optimal solutions with sky lobbies was compared to that of solutions without sky lobbies, and the relative savings were computed across various combinations of floor count and population per floor. Figure 22 illustrates the results for population counts of 20, 50, 100, and 200 persons per floor.

The figure demonstrates that core area savings range from approximately −50% to +32%. This implies that a well-designed sky lobby configuration can reduce the core area by nearly one-third. In practice, the savings potential is likely even greater, as most buildings are not equipped with optimally zoned elevator systems to begin with. On the other hand, if sky lobbies are misconfigured – for example, by introducing an excessive number of them in buildings where they are not beneficial – the required elevator core space can increase by up to 50%. All in all, it is crucial to identify a suitable configuration depending on parameters of the building.

In the top-left figure, representing a population per floor of 20, introducing sky lobbies becomes advantageous only after approximately 50 floors—marking the point where core area savings first become positive for some configurations. Before this, the inclusion of sky lobbies increases the required core area. For instance, at around 40 floors, the core area is approximately 10%, 13%, and 50% greater than the baseline

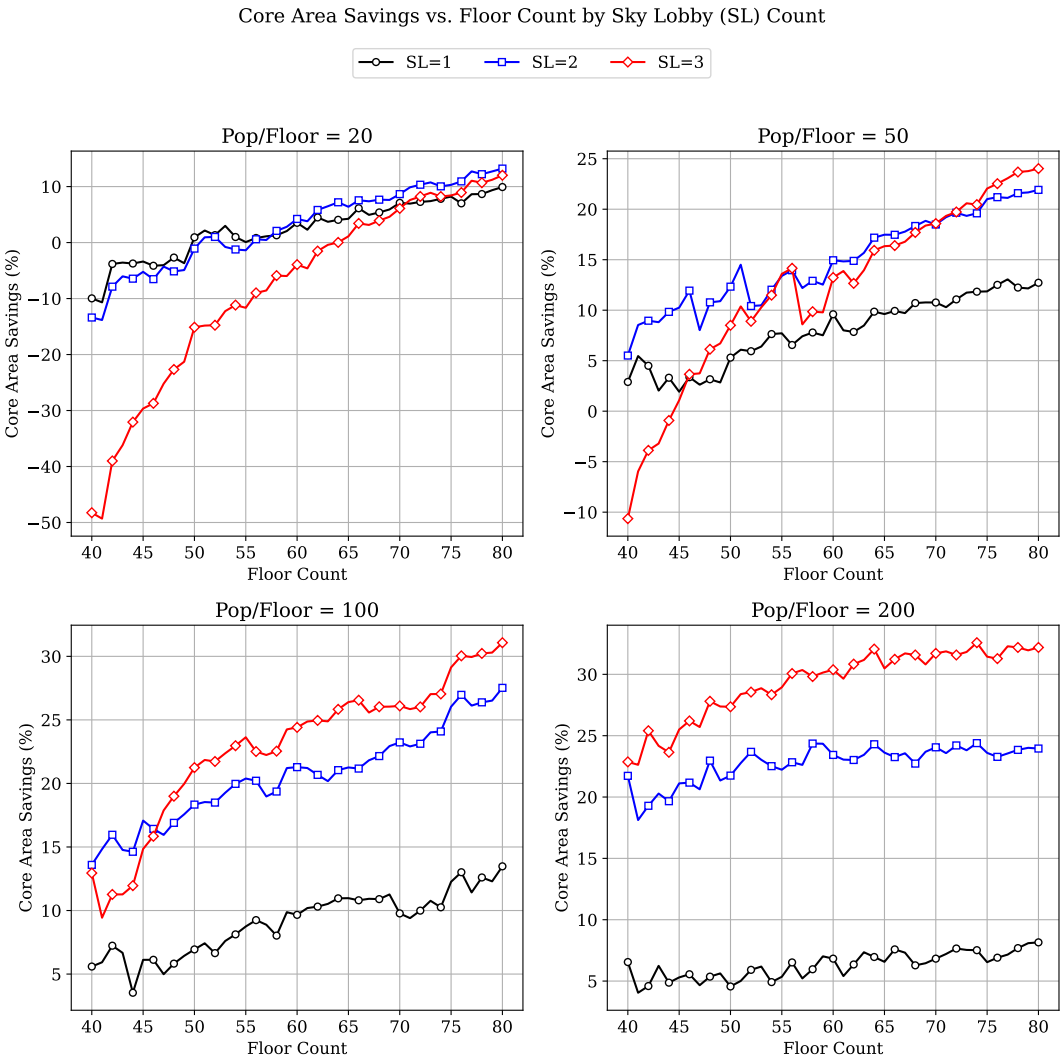


Figure 22: Relative savings in elevator core area for configurations with varying numbers of sky lobbies. Each line represents a different number of sky lobbies, indicated by color: black (one sky lobby), blue (two sky lobbies), and red (three sky lobbies). The savings are calculated as the relative difference in core area between the optimal solution with sky lobbies and the corresponding optimal solution without sky lobbies

(no sky lobbies) when using one, two, and three sky lobbies, respectively. By the 80-floor mark, all sky lobby configurations achieve core area savings exceeding 10%, with the greatest savings – around 12% – achieved using two sky lobbies.

When the population per floor increases to 50 (top-right figure), sky lobbies become beneficial for all floor counts except in the case of three sky lobbies, which only begin to yield savings after approximately 45 floors. Once the floor count exceeds 70, the configuration with three sky lobbies outperforms the two-lobby configuration and becomes the most effective. At 80 floors, the core area savings for one, two, and three

sky lobbies are approximately 13%, 22%, and 24%, respectively.

In contrast to the previous cases, when the population per floor is 100, the configuration with one sky lobby is the least effective from the outset, offering only about 5% savings. Initially, the two-lobby configuration performs best, with savings of around 14%. However, the configuration with three sky lobbies quickly surpasses it and becomes the dominant option beyond 45 floors. Core area savings increase steadily with floor count for all three configurations, reaching approximately 13%, 28%, and 32% for one, two, and three sky lobbies, respectively, at 80 floors.

When the population per floor is very high (200 persons), the three-lobby configuration is dominant from the beginning, offering a core area savings of about 23% already at 40 floors. The two-lobby configuration follows closely, with 22% savings at the same floor count. The one-lobby configuration again performs worst, with savings starting around 7% and remaining at that level across all considered floor counts. As the floor count increases, savings for both the two- and three-lobby configurations continue to grow, reaching approximately 24% and 33%, respectively.

Direct comparison with earlier studies is challenging due to differing assumptions, such as the use of double-deck elevators and varying design criteria. The closest benchmark is Schroeder (1989), who examined a 40-story building with a population per floor of 100. In their Configuration A – single-deck shuttles combined with single-deck local elevators – they reported core area savings of 28% using a single sky lobby. In contrast, the savings potential observed in this study for a comparable setting was only around 6%. However, the benchmark study imposed a higher handling capacity requirement (15%), applied different constraints to shuttle elevators, and did not clearly specify whether the elevator lobby area was included in the core area calculation. Most importantly, their reference case without sky lobbies was based on an arbitrary zoning arrangement whereas the present study uses optimally zoned configurations. Thereby, the core area savings in my study are conservative and provide a lower bound. The scarcity of comparable results in the literature highlights the need for more standardized studies to evaluate the benefits of sky lobbies under consistent modeling assumptions.

5.4.2 Comparison to Rules of Thumb

Another way to assess the core area savings potential is to compare a solution produced by the hybrid metaheuristic to a configuration based on heuristic rules used in the industry. According to Al-Sharif et al. (2017), "it is unusual to have more than four zones in a building. Thus, the number of zones for a zoned building could range from two to four." Furthermore, Barney and Al-Sharif (2015) state that, typically, up to 60 floors can be served without a sky lobby.

In addition, rules of thumb also exist for allocating the building population across zones (Al-Sharif et al., 2017). For two zones, 57% of the population is typically assigned to the lower zone and 43% to the upper zone. For three zones, the recommended allocation is 43% to the lower zone, 30% to the middle zone, and 27% to the upper

zone. For four zones, the population shares are 29%, 27%, 22%, and 22% from the lowest to the highest zone, respectively.

To illustrate this comparison, consider a building with 100 floors and a population of 100 people per floor. Following the rules of thumb, a single sky lobby is assumed to be placed on floor 60, and the number of zones in the stacks ranges from two to four. The corresponding rule-of-thumb configurations are presented for the first and second stack in Tables 13 and 14, respectively.

Table 13: Rule-of-thumb zone configurations for Stack 1 (Floors 0 – 60). Pop. share indicates the percentage of total population allocated to each zone

Zone	Pop. share	Lower floor	Upper floor
i) 2 zones			
1	57%	0	35
2	43%	36	60
ii) 3 zones			
1	43%	0	26
2	30%	27	44
3	27%	45	60
iii) 4 zones			
1	29%	0	18
2	27%	19	35
3	22%	36	49
4	22%	50	60

Table 14: Rule-of-thumb zone configurations for Stack 2 (Floors 61 – 100). Pop. share indicates the percentage of total population allocated to each zone

Zone	Pop. share	Lower floor	Upper floor
i) 2 zones			
1	57%	61	83
2	43%	84	100
ii) 3 zones			
1	43%	61	77
2	30%	78	89
3	27%	90	100
iii) 4 zones			
1	29%	61	72
2	27%	73	83
3	22%	84	92
4	22%	93	100

These configurations are then compared to the solution generated by the hybrid metaheuristic, with a focus on the resulting core area. The results are summarized in Table 15. The optimal solution produced by the metaheuristic has a core area of 17,748.5 (m^2), with the sky lobby placed on floor 49. The lower and upper stacks, spanning floors 0 – 49 and 50 – 100, respectively, each contain seven zones – a substantially higher number than the 2 – 4 zones recommended for a building (or stack) based on industry rules of thumb.

The benchmark configurations based on these rules of thumb result in considerably larger core areas. The configuration with the lowest core area among them is RoT (4L, 3U), with a core area of 24,066.5 (m^2), which is 36% higher than that of the metaheuristic solution. The configuration with the highest core area is RoT (2L, 2U), with a core area of 28,793.3 (m^2), representing a 62% increase compared to the metaheuristic solution.

Overall, these results lend further support to the conclusion that if core area is to be minimized, sky lobbies should be introduced for lower buildings than is currently assumed in the industry. A second conclusion is that it appears to be beneficial to introduce more than 2 – 4 zones per stack.

In comparison to the results presented in the previous section, the core area savings potential appears even greater when an optimized sky lobby configuration is compared

to a configuration based on rules of thumb. It is likely that these savings would increase further in taller buildings (e.g., with 200 floors) or in configurations with multiple sky lobbies. In such cases, it is conceivable that the optimized core area could be less than half of that resulting from a poorly chosen configuration. The economic implications of core area savings of this magnitude would be substantial.

Table 15: Resulting elevator core area (in m^2) for a building with 100 floors, a population of 100 persons per floor, and one sky lobby. Elevator configurations were determined either using the (hybrid) metaheuristic or the rules of thumb proposed by Al-Sharif. *Method* indicates the solution approach; *Config. ID* provides a descriptive label for each Rule-of-Thumb configuration (e.g., RoT (4L,3U) indicates 4 zones in the lower stack and 3 zones in the upper stack); Z_L and Z_U denote the number of zones in the lower and upper stacks, respectively; *SL* denotes the sky lobby floor; *CA* denotes the resulting core area; and CA% denotes the core area relative to that of the configuration produced by the hybrid metaheuristic. The first row corresponds to the metaheuristic solution, while the subsequent rows (sorted in ascending order of core area) correspond to Rule-of-Thumb (RoT) configurations

Method	Config. ID	Z_L	Z_U	SL	CA (m^2)	CA%
Metaheuristic	–	7	7	49	17,748.5	100
Rule of Thumb	RoT (4L,3U)	4	3	60	24,066.5	136
Rule of Thumb	RoT (4L,4U)	4	4	60	24,310.0	137
Rule of Thumb	RoT (4L,2U)	4	2	60	25,029.5	141
Rule of Thumb	RoT (3L,3U)	3	3	60	25,273.1	142
Rule of Thumb	RoT (3L,4U)	3	4	60	25,516.6	144
Rule of Thumb	RoT (3L,2U)	3	2	60	26,236.2	148
Rule of Thumb	RoT (2L,3U)	2	3	60	27,830.3	157
Rule of Thumb	RoT (2L,4U)	2	4	60	28,073.8	158
Rule of Thumb	RoT (2L,2U)	2	2	60	28,793.3	162

5.5 Elevator Core Efficiency

In office buildings, any reduction in elevator core area due to the introduction of one or more sky lobbies translates into additional rentable office space. To quantify this, the ratio of elevator core area to total office area (core-office ratio) was computed for the optimal configurations across various combinations of total floor counts and population per floor. The office area per person was assumed to be 15 m^2, following Schroeder (1985). Figure 23 presents the results for population densities of 20, 50, 100, and 200 persons per floor.

The figure shows that as the population per floor increases, the configuration with

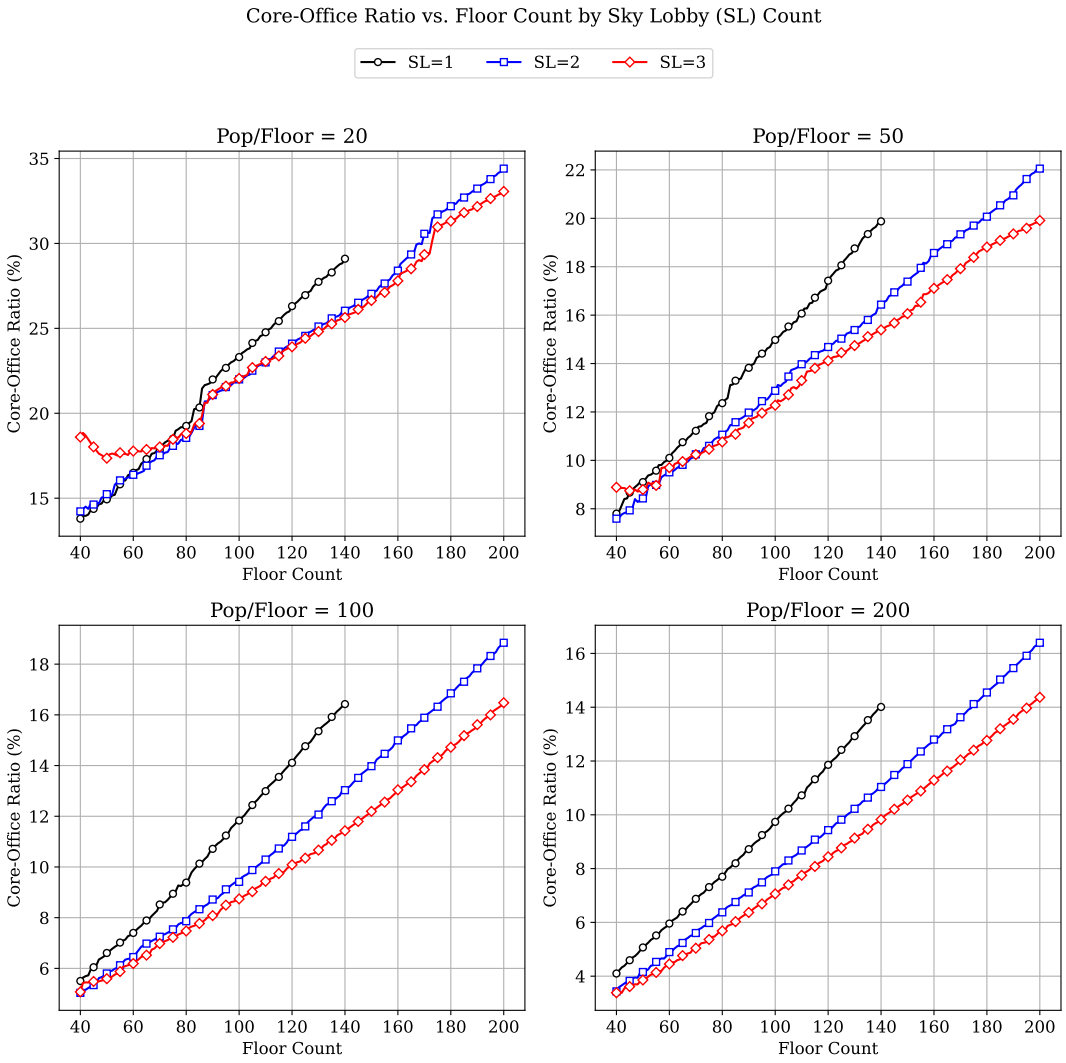


Figure 23: Ratio of elevator core area to total office area, assuming 15 m^2 of office space per person. The colored lines correspond to different numbers of sky lobbies: black represents configurations with one sky lobby, blue with two sky lobbies, and red with three sky lobbies. The plotted values are based on the optimal solutions across varying building heights and population densities

three sky lobbies becomes increasingly attractive, as reflected in lower core-to-office ratios. When the population per floor is 20 (top-left figure) and the floor count is at the minimum of 40, the three-lobby configuration performs the worst, yielding a core-to-office ratio of approximately 18%. In contrast, the best-performing configuration – one sky lobby – has a ratio of around 14%. However, by approximately 70 floors, the one-lobby configuration begins to underperform relative to the others. Up to 200 floors, the two- and three-lobby configurations yield very similar results, with the three-lobby configuration slightly outperforming, reaching a core-to-office ratio of about 33% at the 200-floor mark.

At a population per floor of 50 (top-right figure), the one-lobby configuration is no longer optimal at the outset. Instead, the two-lobby configuration performs best at 40 floors, with a core-to-office ratio of approximately 8%, while the three-lobby configuration lags slightly behind at around 9%. As the floor count increases, however, the three-lobby configuration becomes more favorable. By around 80 floors, it outperforms the other configurations. At higher floor counts, the one-lobby configuration performs significantly worse than both the two- and three-lobby configurations. For example, at 140 floors, the difference is 4 – 5 percentage points. At 200 floors, the two- and three-lobby configurations reach core-to-office ratios of 20% and 22%, respectively.

When the population per floor increases to 100 (bottom-left figure), the three-lobby configuration quickly becomes the most effective. At 40 floors, it already yields a low core-to-office ratio of 5%, increasing steadily to approximately 16% at 200 floors. From 120 floors onward, it consistently outperforms the other configurations, with a 1 – 2 percentage point advantage over the two-lobby configuration and a 3 – 5 percentage point advantage over the one-lobby configuration between 100 and 140 floors. Slightly lower core-to-office ratios are observed when the population per floor is 200, but the overall trends remain qualitatively similar.

Schroeder (1985) produced a similar graph, in which the core-to-office ratio for single-deck elevators (both local and shuttle) with a single sky lobby ranged from approximately 14% (50 floors) to 38% (142 floors), assuming a population per floor of 100. In comparison, the corresponding results in this study are approximately 7% and 17%, suggesting a substantially lower core-to-office ratio. However, several differences in modeling assumptions must be noted. In Schroeder (1985), the 5-minute handling capacity requirement was set at 15%, necessitating a greater number of elevators. Furthermore, their study used arbitrarily selected sky lobby floors and non-optimized zoning arrangements, which contributed to a higher elevator core area. Once again, these findings highlight the need for more standardized studies with aligned modeling assumptions to advance the understanding of elevator core efficiency.

5.6 Runtime

As described in Section 4.4, the computational overhead of this study was substantially reduced through an extensive set of precomputation steps, caching mechanisms, and other runtime optimizations. Table 16 reports the final overall runtime statistics for the outer-level simulation procedure after these optimizations were applied. Notably, the marginal runtime for each full model iteration was reduced to well under one second on average.

Despite the initially high computational complexity, the final runtime across all instances and seeds remained relatively low, reflecting the successful integration of caching, runtime optimization, and HPC resources. Using a naïve approach, optimal hyperparameters could not have been determined, and a single iteration of the full model would likely have taken hours or even days. Compared to such an implementation,

Table 16: Aggregate statistics for runtime (in ms) over all problem instances. Columns: 'Min (ms)' is the minimum runtime, 'Q1 (ms)' is the first quartile (25th percentile), 'Med (ms)' is the median runtime, 'Mean (ms)' is the average runtime, 'Q3 (ms)' is the third quartile (75th percentile), 'Max (ms)' is the maximum runtime, and 'Std (ms)' is the standard deviation of the runtime

Min (ms)	Q1 (ms)	Med (ms)	Mean (ms)	Q3 (ms)	Max (ms)	Std (ms)
25.69	39.72	42.51	45.67	46.48	1572.63	18.18

these measures reduced the runtimes by several orders of magnitude.

Although the reported runtimes may appear trivial in light of the problem's large scale, they are the result of a carefully optimized computational infrastructure and the efficient use of HPC resources. Consequently, there is relatively little to elaborate on – once all optimizations were in place, runtime ceased to be a limiting factor.

6 Conclusions

This thesis has demonstrated that the problem of determining the optimal placement of sky lobby floors in high-rise buildings can be formulated as a nested optimization problem. A hybrid metaheuristic approach was found to be effective in solving this problem for general office buildings. Results suggest that sky lobbies should be introduced for lower building heights than currently assumed in industry practice. Specifically, for buildings of 40 floors or more, the use of sky lobbies significantly reduces elevator core area in most cases.

The placement of sky lobbies in solutions was found to depend primarily on the number of floors in the building and its population per floor. For buildings with low population per floor, the first sky lobby tends to be located near the base, whereas for very tall buildings with high population per floor, the sky lobbies tend to be evenly spaced, forming building stacks of equal floor length. Core area reductions of up to one-third were observed in buildings with 80 floors or fewer. For high-density buildings, the ratio of elevator core area to rentable area can be reduced to 3–14%, depending on the number of floors.

To ensure computational tractability, several practical techniques were employed in this thesis. These include the precomputation of partial solutions to accelerate objective function evaluation, the implementation of gene banks and cached solutions to improve runtime, and the execution of a large-scale hyperparameter grid search on a high-performance computing cluster. Without these measures, the problem would likely have been intractable. Future work should be mindful of these computational challenges and may benefit from the methodological techniques detailed in Section 4.4.

Several novel contributions to the literature arise from this work. First, it provides a formal mathematical formulation for the problem of optimal sky lobby configuration, which has not previously been established. While earlier studies typically investigated a limited number of configurations, this study analyzed a broad range of instances, enabling the identification of general patterns. To date, questions such as when to introduce sky lobbies, how many to use, where to place them, and the magnitude of potential core area savings had not been systematically addressed. This thesis offers comprehensive answers to these questions and underscores the importance of population density as a design parameter, which has received limited attention in prior studies.

In addition to its contributions to the elevator systems literature, this thesis also advances the field of metaheuristics. The proposed heterogeneous high-level relay hybrid, combining evolutionary algorithms (EAs) with simulated annealing (SA), demonstrates that single-solution-based metaheuristics can effectively be employed at the outer level of a hybrid framework—contrary to the common practice of assigning this role to EAs.

Further contributions are made to the emerging literature on Hidden Genes Genetic

Algorithms (HGGA), a relatively underexplored concept. The results support the use of HGGA for complex problems with variable-sized design spaces. Instead of relying on an iterative and systematic search, the HGGA approach can yield effective solutions in a single run. Moreover, the thesis shows that a binomial distribution can be employed to generate an initial population that is both diverse and focused, which is critical for performance in genetic algorithms.

The extensive hyperparameter grid search conducted for both the HGGA and SA methods provides valuable insights into effective parameter settings, offering a useful foundation for future research in elevator zoning and other related optimization problems. Due to the hybrid nature of the algorithm, the number of tunable parameters is considerable. While most were optimized through grid search, some were tuned based on empirical experimentation, which constitutes a limitation of this study. However, the scale of the grid search conducted here is uncommon in the literature, and extending it to include all parameters would have been computationally prohibitive. This limitation is inherent in many studies employing metaheuristics or other approximate methods. Developing exact methods remains an important direction for future research, as such approaches would also enable a more rigorous validation of the current results – particularly in understanding whether apparent outliers in the solution space are due to local optima or reflect true properties of the problem's discrete structure.

A simplifying assumption throughout this study was the use of a constant population per floor. While practical for computational purposes, this assumption may not hold in real-world mega high-rise buildings, which often include mixed-use functions such as retail, hotel, residential, and office spaces. These different functions imply heterogeneous population distributions. Accounting for this variation would have rendered precomputation infeasible and likely made the problem computationally intractable. In practice, however, solving a single instance is computationally manageable, and real-time solutions are typically not required in zoning problems.

Although elevator core area was used as the primary objective, other objectives such as system cost are also important in practice. Prior work has suggested that minimizing core area and system cost are generally aligned objectives. Nevertheless, future work could benefit from extending the current model into a multi-objective framework, incorporating additional criteria such as evacuation time, which has become increasingly relevant since the events of 9/11.

In this study, the sizing of the shuttle elevator groups was based on uppeak traffic conditions. However, some studies have proposed more conservative sizing guidelines. For example, Schroeder (1984) suggests that shuttle elevator groups should be 40 – 50% larger than what an uppeak analysis alone would indicate. This recommendation stems from the observation that local elevator groups are typically capable of handling 40 – 50% more traffic during downpeak conditions. If the shuttle elevator group is undersized in such scenarios, sky lobbies may become congested and create a bottleneck in the system. Consequently, future studies should validate the solutions produced by the presented method through simulation-based analysis.

Previous studies on elevator zoning have typically considered only the shaft area of elevators, while neglecting the lobby area required for each elevator. In contrast, lobby area was explicitly included in this study, thereby preventing the underestimation of the additional core area resulting from the introduction of new elevators. A sensitivity analysis on the impact of lobby area assumptions was beyond the scope of this thesis. However, it would be valuable to examine whether the results would differ substantially if lobby area were excluded or determined using an alternative rule. Such an analysis is recommended for future research.

Finally, this thesis focused on single-deck elevator systems for the sake of simplicity. However, double-deck systems are widely used in practice and offer further potential for core area savings. Extending the current formulation to include double-deck elevators would be a natural and important avenue for future research, requiring careful attention to how building populations are segmented and served by distinct elevator configurations.

References

Aarts, E. H. L., Korst, J. H. M., & Laarhoven, P. J. M. v. (1997). Simulated annealing. In E. H. L. Aarts & J. K. Lenstra (Eds.), *Local Search in Combinatorial Optimization* (pp. 91–120). Wiley-Interscience.

Abdelkhalik, O. (2013). Hidden Genes Genetic Optimization for Variable-Size Design Space Problems. *Journal of Optimization Theory and Applications*, *150*(450), 468.

Abdelkhalik, O., & Darani, S. (2016). Hidden Genes Genetic Algorithms for Systems Architecture Optimization. *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 629–636. https://doi.org/10.1145/2908812.2908819

Abdelkhalik, O., & Darani, S. (2018). Evolving Hidden Genes in Genetic Algorithms for Systems Architecture Optimization. *Journal of Dynamic Systems, Measurement, and Control*, *140*(10), 101015. https://doi.org/10.1115/1.4040207

Al-Sharif, L. (2017). The Design of Elevator Systems in High Rise Buildings, Part 1. *Lift Report*, *43*, 46–62.

Al-Sharif, L., Al Sukkar, G., Hakouz, A., & Al-Shamayleh, N. A. (2017). Rule-based calculation and simulation design of elevator traffic systems for high-rise office buildings. *Building Services Engineering Research and Technology*, *38*(5), 536–562. https://doi.org/10.1177/0143624417705070

Al-Sharif, L., Sukkar, G. A., Hakouz, A., & Al-Shamayleh, N. A. (2016). Towards a Systematic Methodology for the Design of Elevator Traffic Systems in High Rise Office Buildings.

Barney, G. C., & Dos Santos, S. M. (1985). Elevator Traffic Analysis. *Design and Control (Peter Peregrinus Ltd, 2nd edition, London, 1985)*.

Barney, G. C., & Al-Sharif, L. (2015, September). *Elevator Traffic Handbook: Theory and Practice* (2nd ed.). Routledge. https://doi.org/10.4324/9781315723600

Blum, C., Puchinger, J., Raidl, G. R., & Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, *11*(6), 4135–4151. https://doi.org/10.1016/j.asoc.2011.02.032

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, *35*(3), 268–308. https://doi.org/10.1145/937503.937505

Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, *10*(2), 141–171.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, *45*(1), 41–51. https://doi.org/10.1007/BF00940812

Connolly, D. T. (1990). An improved annealing scheme for the QAP. *European Journal of Operational Research*, *46*(1), 93–100. https://doi.org/10.1016/0377-2217(90)90301-Q

Council on Tall Buildings and Urban Habitat. (2024, January). *Year in Review: Tall Trends of 2023* (tech. rep.). Council on Tall Buildings and Urban Habitat. https://www.skyscrapercenter.com/year-in-review/2023

Fortune, J. W. (1985). Elevatoring High-rise Buildings. *Elevator World*, *33*(5), 36–42.
Fortune, J. W. (1995). Mega High-Rise Elevators. *Elevator World*, *43*(7), 63–69.
Gendreau, M., & Potvin, J.-Y. (Eds.). (2019). *Handbook of Metaheuristics* (Vol. 272). Springer International Publishing. https://doi.org/10.1007/978-3-319-91086-4
Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, *13*(5), 533–549. https://doi.org/10.1016/0305-0548(86)90048-1
Goldberg, D. E., & Deb, K. (1991). A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In *Foundations of Genetic Algorithms* (pp. 69–93, Vol. 1). Elsevier. https://doi.org/10.1016/B978-0-08-050684-5.50008-2
Holland, J. H. (1992). Genetic Algorithms. *SCIENTIFIC AMERICAN*.
Husbands, P., Mill, F., & Warrington, S. (1991). Genetic algorithms, production plan optimisation and scheduling. In H.-P. Schwefel & R. Männer (Eds.), *Parallel Problem Solving from Nature* (pp. 80–84). Springer. https://doi.org/10.1007/BFb0029735
Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671
Laarhoven, P. J. M. v., Aarts, E. H. L., & Lenstra, J. K. (1992). Job Shop Scheduling by Simulated Annealing. *Operations Research*, *40*(1), 113–125. http://www.jstor.org/stable/171189
Lasserre, J. B. (1992). An Integrated Model for Job-Shop Planning and Scheduling. *Management Science*, *38*(8), 1201–1211. https://doi.org/10.1287/mnsc.38.8.1201
Lin, F.-T., Kao, C.-Y., & Hsu, C.-C. (1991). Incorporating genetic algorithms into simulated annealing. *Proceedings. International Symposium on Artificial Intelligence*, 290–297.
Liu, Y., Hu, Z., Su, Q., & Huo, J. (2010). Energy Saving of Elevator Group Control Based on Optimal Zoning Strategy with Interfloor Traffic. *2010 3rd International Conference on Information Management, Innovation Management and Industrial Engineering*, *3*, 328–331. https://doi.org/10.1109/ICIII.2010.399
Lodi, A., Martello, S., & Vigo, D. (1999). Neighborhood Search Algorithm for the Guillotine Non-Oriented Two-Dimensional Bin Packing Problem. In S. Voß, S. Martello, I. H. Osman, & C. Roucairol (Eds.), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization* (pp. 125–139). Springer US. https://doi.org/10.1007/978-1-4615-5775-3_9
Matsuo, H., Juck SUH, C., & Sullivan, R. S. (1989). A controlled search simulated annealing method for the single machine weighted tardiness problem. *Annals of Operations Research*, *21*(1), 85–108. https://doi.org/10.1007/BF02022094
Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, *21*(6), 1087–1092. https://doi.org/10.1063/1.1699114
Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press.

Olympio, J. T., Marmorat, J.-P., & Izzo, D. (2007). Global trajectory optimisation: Can we prune the solution space when considering deep space maneuvers? *ARIADNA study*, *6*, 4101.

Otis Elevator Company. (1967). Otis Bulletin: Vertical Transportation for the World Trade Center [Archived online in 2006]. Retrieved March 18, 2025, from https://web.archive.org/web/20061115072321/http://www.otis.com/otis150/section/1,2344,ARC3066_CLI1_RES1_SEC5,00.html

Powell, B. A. (1971). Optimal Elevator Banking Under Heavy Up-Traffic. *Transportation Science*, *5*(2), 109–121.

Powell, B. A. (1975). Elevator Banking for High Rise Buildings. *Transportation Science*, *9*(3), 200–210. https://doi.org/10.1287/trsc.9.3.200

Roschier, N.-R., & Kaakinen, M. J. (1978). New formulae for elevator round trip time calculations. *Elevator World supplement*.

Rudolph, G. (1994). Convergence analysis of canonical genetic algorithms. *IEEE transactions on neural networks*, *5*(1), 96–101.

Ruokokoski, M., & Siikonen, M.-L. (2017). Lift Planning and Selection Graphs [ISSN: 2052-7233 Issue: 1]. *Proceedings of the 7th Symposium on Lift and Escalator Technologies*, *7*, 234–244.

Ruokokoski, M., Sorsa, J., & Siikonen, M.-L. (2018). A Systematic Methodology for Analysing Zoning Options for a Building Using Dynamic Programming. *Proceedings of the 9th Symposium on Lift and Escalator Technologies*, *9*, 17–29.

Schroeder, J. B. (1984). "To shuttle, or not to shuttle". *Elevator World*.

Schroeder, J. B. (1985). ELEVATOR CORE SPACE: A Limiting Factor For Very Tall Buildings? *Elevator World*, 52–54.

Schroeder, J. B. (1989). SKY LOBBY ELEVATORING. *Elevator World*, *37*(1).

Siikonen, M.-L. (1997a). *Elevator Group Control with Artificial Intelligence* (Research Report). Helsinki University of Technology. KONE Corporation.

Siikonen, M.-L. (1997b). *Planning and Control Models for Elevators in High-Rise Buildings* [Doctoral Thesis]. Helsinki University of Technology.

Siikonen, M.-L. (2021, October). *People Flow in Buildings*. John Wiley & Sons.

Siikonen, M.-L. (2024). Current and future trends in vertical transportation. *European Journal of Operational Research*, *319*(2), 361–372. https://doi.org/10.1016/j.ejor.2024.05.016

Sorsa, J., Siikonen, M.-L., & Ehtamo, H. (2003). Optimal control of double-deck elevator group using genetic algorithm. *International Transactions in Operational Research*, *10*(2), 103–114. https://doi.org/10.1111/1475-3995.00397

Sorsa, J. (2002). *Optimal group control of double-deck elevators* [Master's thesis, Helsinki University of Technology]. https://urn.fi/URN:NBN:fi:aalto-2020120448820

Starr, B. (2013). Spooled DNA and hidden genes: The latest finding in how our DNA is organized and read [Archived online in 2018]. Retrieved March 25, 2025, from https://web.archive.org/web/20180310123030/http://genetics.thetech.org/original_news/news31

Strakosch, G. R. (1967). *Elevators and escalators* (1st ed.). Wiley.
Strakosch, G. R. (1984). Improve your elevatoring capability-Part IV. *Elevator World*, (2), 8–9.
Talbi, E.-G. (2002). A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics*, *8*(5), 541–564. https://doi.org/10.1023/A:1016540724870
Talbi, E.-G. (2009, May). *Metaheuristics: From Design to Implementation*. John Wiley & Sons.
Tyni, T., & Ylinen, J. (1999, August). *Method and apparatus for allocating landing calls in an elevator group* (US5932852A).
Tyni, T., & Ylinen, J. (2001, July). Genetic Algorithms in Elevator Car Routing Problem. In L. Spector & e. al et (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (pp. 1413–1422). Morgan Kaufmann Publishers.
Viita-aho, P. (2019). *Systematic methods for analyzing elevator group zoning options for high-rise buildings* [Master's thesis, Aalto University]. https://urn.fi/URN:NBN:fi:aalto-201908254927
von Schantz, A., & Ehtamo, H. (2022). Minimizing the evacuation time of a crowd from a complex building using rescue guides. *Physica A: Statistical Mechanics and its Applications*, *594*, 127011. https://doi.org/10.1016/j.physa.2022.127011
Yang, X.-S. (2010, July). *Engineering Optimization: An Introduction with Metaheuristic Applications*. John Wiley & Sons.

A Tag Evolution Mechanisms in HGGA

The Hidden Genes Genetic Algorithm (HGGA) enables genes to be conditionally excluded from evaluation using binary tags. These tags evolve over generations using dedicated mechanisms. Two general classes of mechanisms are proposed by Abdelkhalik and Darani (2018): logical evolution and stochastic evolution.

A.1 Logical Evolution of Tags

In logical evolution, the tags of the offspring are computed directly from the tags of the parents using fixed logical rules:

- **Logic A (Mixed Logic)**:
  - Child 1: Hidden-OR — a gene is hidden if it is hidden in either parent.
  - Child 2: Active-OR — a gene is active if it is active in either parent. This is equivalent to Hidden-AND.
- **Logic B**: Hidden-OR is applied to both children.
- **Logic C**: Active-OR is applied to both children.

A.2 Stochastic Evolution of Tags

In stochastic evolution, tags are treated as discrete binary variables that evolve via mutation and/or crossover. Eight mechanisms have been proposed:

- **Mechanism A**: Tags are stored separately from the chromosome and updated using mutation with a fixed mutation probability. No crossover is applied to the tags.
- **Mechanism B**: Tags are appended to the design variables in the chromosome and evolve via both mutation and crossover. This increases the chromosome length, but not the cost function complexity.
- **Mechanism C**: Tags evolve via crossover only. They are treated as discrete variables in the chromosome, with mutation disabled.
- **Mechanism D**: Tags evolve via mutation only. As in Mechanism C, tags are included in the chromosome but only undergo mutation.
- **Mechanism E**: Tags and genes undergo independent crossover. Mutation is applied to tags before crossover. This creates a two-dimensional crossover operator.
- **Mechanism F**: A fitness-guided arithmetic crossover is used. Intermediate offspring are generated from single-point crossover on genes and Active-OR on tags. Final offspring is chosen based on the fitness of the intermediates.

- **Mechanism G**: Arithmetic crossover is applied using a modified cost function biased toward parents with more hidden genes. The cost function becomes:

$$f_{\text{modified}}(X) = f(X) - \sum_{i=1}^{M} \text{flag}_i$$

- **Mechanism H**: Similar to Mechanism G, but biased toward fewer hidden genes:

$$f_{\text{modified}}(X) = f(X) + \sum_{i=1}^{M} \text{flag}_i$$

For full technical details, see Abdelkhalik and Darani (2018) and Abdelkhalik and Darani (2016).