TU WIEN Informatics

Optimization of Production Layouts in Multi-Floor Industrial Buildings

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Media and Human-Centered Computing

eingereicht von

David Suppan, BSc
Matrikelnummer 11813815

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Dr. Peter Kán

Mitwirkung: Assistant Prof. Dr. Julia Reisinger

Wien, 4. September 2025

David Suppan	Peter Kán

TU WIEN Informatics

Optimization of Production Layouts in Multi-Floor Industrial Buildings

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Media and Human-Centered Computing

by

David Suppan, BSc
Registration Number 11813815

to the Faculty of Informatics

at the TU Wien

Advisor: Dr. Peter Kán

Assistance: Assistant Prof. Dr. Julia Reisinger

Vienna, September 4, 2025

David Suppan | Peter Kán

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

David Suppan, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 4. September 2025

David Suppan

Kurzfassung

In einem Umfeld, in dem der Baugrund eine begrenzte Ressource ist und wirtschaftliche Schwankungen häufiger werden, wandeln sich flexible Produktionsketten für eine Vielzahl an Branchen von einem gewerblichen Vorteil zu einer unmittelbaren Notwendigkeit. Ein wichtiger Schritt in die Richtung hoher Anpassungsfähigkeit in Produktionsabläufen ist der vermehrte Einsatz flexibler Produktionsstätten. Diese Gebäude versprechen die Resilienz gegenüber sich ändernden Anforderungen zu maximieren und, oft sich widersprechende, geläufige Leistungsindikatoren wie Materialflusskosten und die benötigte Baufläche kontinuierlich zu optimieren. In den letzten Jahrzehnten hat sich die wissenschaftliche Gemeinde zunehmend sogenannten Facility Layout Problems (FLP) in einem mehrstöckigen Kontext (MFFLP) zugewandt. So vielversprechend digital optimierte mehrstöckige Produktionsstätten wirken mögen, so sehr bergen sie auch völlig neue Herausforderungen. Diese Arbeit präsentiert einen neuartigen, auf Ports basierenden, zweiphasigen evolutionären Algorithmus für das MFFLP mit Aufzügen (MFFLPE), der mit fünf individuellen Zielfunktionen, dynamisch platzierten Lifts und größenfixierten Abteilen arbeitet. Der in der C# Programmiersprache verfasste Algorithmus arbeitet in einem diskreten Suchraum bestehend aus möglichen Positionen und Verbindungen von Produktionsabteilen. Die Prozedur wird sowohl quantitativ als auch qualitativ evaluiert. Zunächst wird sie mittels der Analyse der umfangreichen Daten untersucht, die während Simulationsexperimenten mit zwei individuellen Eingabesets aufgezeichnet wurden. Dies inkludiert auch den Vergleich zwischen zwei gängigen Fitnessevaluierungsstrategien: Evaluierung basierend auf Pareto-Dominanz und Evaluierung basierend auf der (ungewichteten) Skalarisierung von Zielen. Zweitens hilft uns ein ergänzender Fragebogen, beantwortet von einem Experten im Bauingenieurwesen, die Stärken, Schwächen und grundsätzliche Performanz des Algorithmus weiter zu untersuchen. Unsere Ergebnisse suggerieren, dass ein auf Ports basierender Ansatz geeignet dafür ist kohärente mehrstöckige Produktionslayouts zu generieren, unabhängig von der verwendeten Evaluierungsmethode. Obwohl wir Unterschiede zwischen den zwei Strategien bezüglich der Zielerfüllung feststellen konnten, so stellten wir auch Probleme mit dem verwendeten Normalisierungsmechanismus für die skalarisierungsbasierte Implementierung fest. Dementsprechend geht aus unseren Daten keine Evaluierungsmethode für das anstehende Problem als besser geeignet hervor. Dadurch, dass beide Optimierungsansätze ihre eigenen Vorteile mit sich bringen, schlussfolgern wir, dass ein hybrider Ansatz für MFFLPEs als am vielversprechendsten scheint, vorausgesetzt es gibt eine intakte Normalisierung von Zielen. Obwohl die Ergeb-

nisse suggerieren, dass die skalarisierungsbasierte Implementierung schneller für kleinere Probleminstanzen ist, benötigen beide Evaluierungsstrategien grundsätzlich vergleichbare Laufzeiten. Das von dem Experten bereitgestellte Feedback offenbarte gewisse Ziele, die noch von zukünftigen Arbeiten untersucht werden müssen. Schließlich müssen zukünftige Werke eine größere Varietät von unterschiedlichen Eingabesets untersuchen, um fundierte Aussagen über die Robustheit von gleichartigen MFFLPE Optimierungsprozeduren machen zu können.

Abstract

In a world with limited real estate and a volatile economic landscape, flexible production chains are transitioning from a competitive advantage to an outright necessity for a variety of industries. One major step in the direction of high adaptability in production processes is the reliance on flexible production facilities. These facilities promise to maximize resilience against changing demands and consistently optimize, often conflicting, common key performance indicators such as material handling costs and total land required. In the last decades, the scientific community has increasingly considered so-called Facility Layout Problems (FLP) in a multi-floor context (MFFLP). As promising as digitally optimized multi-story production facilities seem, they also pose entirely new challenges. This thesis presents a novel port-based two-phase evolutionary algorithm to the MFFLP with Elevators (MFFLPE), considering five distinct objectives, dynamically placed elevators, and fixed-size departments. The algorithm operates on a discrete search space of possible cube locations and connections, and was written in the C# programming language. The procedure is evaluated both quantitatively and qualitatively. First, it is analyzed with respect to the extensive data recorded throughout simulation experiments conducted with two distinct input sets. This includes the comparison of two well-known fitness evaluation measures: Evaluation based on Pareto-dominance and evaluation based on the (unweighted) scalarization of objectives. Secondly, a supplementary questionnaire answered by a civil engineering expert helps to further investigate the strengths, weaknesses, and overall performance of the proposed algorithm. Our results indicate that the port-based approach is suitable to generate coherent multi-floor production layouts with satisfactory objective values, independent of the evaluation method utilized. While we observed differences between the two strategies in terms of objective completion, we also identified issues in the utilized normalization mechanism for the scalarization-based implementation. Thus, our data does not suggest any evaluation strategy to be more appropriate for the task at hand. As both optimization strategies pose their own advantages, we conclude that a hybrid approach seems the most promising for MFFLPEs, given proper objective normalization. Even though the results indicate that the scalarization-based implementation is faster for smaller problem instances, both evaluation measures generally required similar runtimes. Feedback provided by the expert revealed that certain objectives have yet to be investigated in future work. Lastly, future work needs to explore a wider variety of different input sets to make well-founded statements about the robustness of MFFLPE optimization procedures alike.

Contents

CHAPTER 1

Introduction

1.1 Problem Statement

With new technologies, requirements, and an ever-changing landscape of consumer needs, production processes have evolved in their complexity and diversity. This transformation poses new challenges in the logistics and industrial sectors, as production space is often a limited commodity. In recent years, new ways of adapting to this discrepancy between production space needs and availability have led to digitized factory design processes, promising new levels of flexibility. To successfully design a functioning and long-lasting facility, a lot of expertise from many different disciplines is required. Additionally, many of the building requirements are conflicting, like layout density and flexibility. All of these factors have led to an increased research volume regarding production layout optimization in the last decade [42]. The digitized optimization of factory production designs can be seen through many different lenses: The focus can be on improved flexibility of building structures [43], on reduction of life cycle costs [41] or carbon emissions [17], and on optimized production layouts with minimal material handling costs [29, 20, 28, 53, 52, 57].

The latter problem is commonly known as a Facility Layout Problem (FLP) [48]. As we will later see, this problem genre has a wide range of sub-formulations with different objectives, constraints, and solution approaches. Especially due to the conflicting nature of goals in FLPs, special stochastic algorithms called meta-heuristics, as well as multi-objective optimization models, have received encouragement among researchers in the past decades [52, 45, 46, 56, 48]. Also referred to as multi-objective meta-heuristics (MOMH) by Zitzler et al. [61], the main promise of these strategies is to approximate good, however not necessarily optimal, solutions for problems that would otherwise require exponential computation times when solved via traditional deterministic algorithms. FLPs are generally considered very complex (NP-hard) optimization problems [18].

Unsurprisingly so, the application of meta-heuristics for FLPs has been extensively discussed in previous research [22]. Still, related work covering many-objective formulations of multi-floor FLPs, i.e., formulations dealing with more than 3 objectives, is scarce. As we will see in Section 2.1.1, this especially holds for problem instances that explicitly model elevators. We aimed to close this gap by developing a discrete many-objective evolutionary algorithm that can handle multi-floor layouts with departments of fixed size and non-colliding elevators.

1.2 Aim of this work

In this thesis, we tried to answer the following research questions:

- *How can we find optimal multi-floor production layouts with fixed department dimensions from the Pareto set while considering multiple objectives using an evolutionary algorithm?*
- *How does a dominance-based evaluation measure influence the quality of the results compared to a traditional scalarization approach?*

In the context of these questions, our hypotheses were as follows:

Hypothesis 1 *Multi-floor production layouts with fixed department sizes calculated using scalarization reach better fitness values compared to dominance-based SPEA2+SDE evaluation.*

Hypothesis 2 *Multi-floor production layouts with fixed department sizes calculated using scalarization are generated faster compared to dominance-based SPEA2+SDE evaluation.*

Hypothesis 3 *Multi-floor production layouts with fixed department sizes, calculated using a dominance-based SPEA2+SDE evaluation, have a superior Hypervolume compared to scalarization evaluation.*

Concepts such as the Pareto set, dominance-based evaluation, SPEA2+SDE, and the Hypervolume measure will be explained shortly in Section 2.2.

1.3 Methodology

To answer the research questions and investigate our hypotheses, we applied 3 steps:

- First, we reviewed previous research regarding single- and especially multi-floor FLPs. We investigated many-objective optimization from a general viewpoint and subsequently narrowed down our focus to stochastic computational procedures called meta-heuristics, as they pose a popular approach for solving many-objective problems. Furthermore, we investigated the computer-aided design (CAD) software Rhinoceros 3D [4], its integrated visual programming environment Grasshopper [11], and their capabilities in terms of visualizing generated layouts.
- Secondly, we implemented a many-objective evolutionary algorithm for solving multi-floor FLPs. The algorithm is characterized by its two-phased design, its use of discrete positioning anchors called ports and its ability to consider non-colliding elevators. The base algorithm design is the Strength Pareto Evolutionary Algorithm 2 (SPEA2) [60] with Shift-based Density Estimation (SDE) [34], as previous research suggests this combination to be an effective method for similar optimization problems in the single-floor domain [56].
- The third and final step covers the evaluation of the proposed algorithm. This step is two-fold:
  - First, we evaluated the algorithm's performance quantitatively: We recorded and compared a wide range of data, including elapsed milliseconds per iteration, the average objective function values of solutions, and the Hypervolume over time.
  - Secondly, we designed a questionnaire that included a number of different layouts, with one half being generated via scalarization and the other half being generated via dominance-based evaluation. The questions focused on a number of different aspects that are commonly considered key performance indicators for production layouts, like material handling costs. This questionnaire was answered by a civil engineering expert.

In summary, the contribution of this thesis is a specialized many-objective evolutionary algorithm for generating fixed-size department multi-floor production layouts, which includes dynamically placed bi-directional non-colliding elevators. Additionally, this thesis investigates whether the evaluation measure based on Pareto-dominance is superior to a traditional scalarization approach, in the context of a many-objective optimization procedure focused on generating multi-floor production layouts. Lastly, this thesis gives additional insights about the general performance and feasibility of such an algorithm by presenting and interpreting the answers to a tailored questionnaire that were provided by a qualified expert.

1.4 Structure

The thesis begins with a literature review that details the various problem formulations of the commonly known Facility Layout Problem (FLP). In this chapter, we discuss the fundamentals of common solving approaches to multi-objective FLPs, with a focus on meta-heuristics and genetic algorithms. Chapter 3 describes the proposed evolutionary algorithm for solving FLPs in a multi-floor and multi-objective context. This chapter is divided into two sections: The formal description of the algorithm outlines, including constraints, objectives, and assumptions, and the specification of the genetic stages that make up the main iteration loop. In Chapter 4, we describe the simulation experiments conducted to evaluate the algorithm's performance. This chapter also includes the results obtained via the expert study. In Chapter 5, we present an interpretation for the collected data in the context of our research questions and the hypotheses. Here, we also discuss the potential limitations of our approach as well as aspects that remain to be investigated in future work. Finally, in Chapter 6, our findings are summarized and concluded.

CHAPTER 2

Literature Review

The undertaking of optimizing facilities with the help of software is not new. The oldest paper we have found dates back to 1963 by Armour and Buffa [3] and describes a simulation of a layouting problem written on 1100 punch cards with 15 subroutines. It is thus unsurprising that the research volume about facility optimization is both vast and many-faceted. The following sections briefly describe the different commonly known sub-types of said problem and will subsequently focus on recent literature dealing with layout problems in the multi-floor domain, the domain of this thesis.

2.1 The Facility Layout Problem

The aforementioned layouting problem is commonly referred to in literature as the Facility Layout Problem (FLP). Hosseini-Nasab et al. [22] define the FLP as "finding the most efficient arrangement of elements on the factory floor subject to different constraints in order to meet one or more objectives".

Usually, each pair of departments has a flow intensity assigned that describes how much material flows between the two departments in a given time frame. In the context of these relationships, the material handling cost (MHC) is the most prominent efficiency measure. It is calculated by summing up the distances the individual material flows need to travel, weighted by their corresponding intensities.

FLPs can be classified in a vast number of ways [22]. For instance, FLPs can be static, with constant material flows, or dynamic, where the material flows are considered variable, often with priced-in layout changes. Furthermore, FLPs can have departments with regular or irregular shapes, as well as fixed or variable dimensions. There are different layout configurations, like single- (SRFLP), double-row (DRFLP) and multi-floor FLPs (MFFLP). If a MFFLP considers elevators, it is usually labeled as a MFFLP with Elevators (MFFLPE). Depending on the types of elevators included, they can be

further classified as MFFLPE-A, denoting MFFLPEs with only full-service elevators, or MFFLPE-B, denoting problem instances that allow both partial- and full-service elevators [18]. Additionally, all of these formulations can be considered as Unequal Area variations (e.g. UA-FLP, UA-MFFLP, etc.), in which each of the departments has a fixed area that it needs to cover, instead of having exact length and width constraints [3, 22]. The spectrum of possible constraints can also be considered wide and includes, for instance, overlap prohibition, budget or elevator capacity limits, or the adherence to specific aspect ratios [29]. FLPs can be solved in the context of one or more objectives, such as, but not limited to, minimizing material handling [36, 25, 15] or rearrangement costs [25, 56], adhering to neighborhood/closeness conditions of departments [43, 2] and maximizing safety aspects [28]. Typical modeling approaches include the use of Mixed Integer Programming (MIP), the Quadratic Assignment Problem (QAP), Linear Programming (LP), Non-Linear Programming (NLP), and genetic algorithms. Lastly, solving approaches include the use of algorithms classified as exact (e.g., QAP solvers), (meta-)heuristic, stochastic, and artificial intelligence [22].

2.1.1 The Multi-Floor Facility Layout Problem (MFFLP)

Although a vast amount of research has been conducted on single-floor FLPs, there are significantly fewer papers available on multi-floor problem variations. This is especially true for problem instances that actively cover the sub-problem of finding fruitful elevator positions. Still, there is a lot of overlap between the multi-floor and previously discussed single-floor formulations [18].

				Department		Elevator			
Reference	**Problem**	**Cont.?**	**Objectives**	**Floor**	**Dim.**	**Pos.**	**Count**	**Has area?**	**Model**
Meller and Bozer [36]	UA-MFFLP	✗	MHC	D	D*	P	P	✗	QAP
Lee et al. [33]	UA-MFFLP	✓	MHC, ADJ	D	D*	P	P	✗	GA
Huang et al. [23]	UA-MFFLPE	✓	MHC, EC	D	D*	D	D	✓	QAP
Karateke et al. [29]	UA-MFFLP	✓	MHC	D	D*	P	P	✗	LP
Enayaty Ahangar et al. [15]	UA-MFFLP	✓	MHC, EC, DEN, FDEN	D	D*	C	1	✗	MIP
Ji et al. [28]	MFFLPE	✗	MHC, EMC	P	C	D	P	✓	MIP
Goetschalckx and Irohara [18]	MFFLPE-B	✓	MHC	D	P	D	P	✗	MIP
Hathhorn et al. [20]	MFFLPE-A	✓	MHC, DEN, FDEN	D	D	D	D	✗	MIP
Izadinia et al. [25]	MFFLPE-A	✗	MHC	P	C	D	P	✓	MIP
Ahmadi and Akbari Jokar [1]	UA-MFFLPE-A	✓	MHC	D	D*	D	P	✗	MIP
Ours	**MFFLPE-B**	✗	**MHC, ADJ, DEN, FDEN, PC**	**P**	**P**	**D**	**P**	✓	**GA**

C = Constant, D = Decision variable, P = Parameter,
D* = Decision variable limited by Unequal Area (UA) constraint

Table 2.1: Papers on multi-floor facility layout problems

Table 2.1 provides an overview of related literature about FLPs in a multi-floor context. We classified the problems that papers were dealing with as MFFLP with elevators (MFFLPE) if the position of elevators is defined by the algorithm, and not pre-determined by the inputs. We can see that most of the papers either provided fixed elevator positions or neglected the area occupied by placed elevators. Furthermore, we can see that most papers only consider full-service elevators (MFFLPE-A), covering all floors, instead of both full- and partial-service elevators, that might only cover a subset of all the layout levels (MFFLPE-B). Lastly, the MHC metric is the dominant choice for the main objective. Many-objective multi-floor optimization scenarios have yet to be investigated in-depth, as most literature only considers one or two goals.

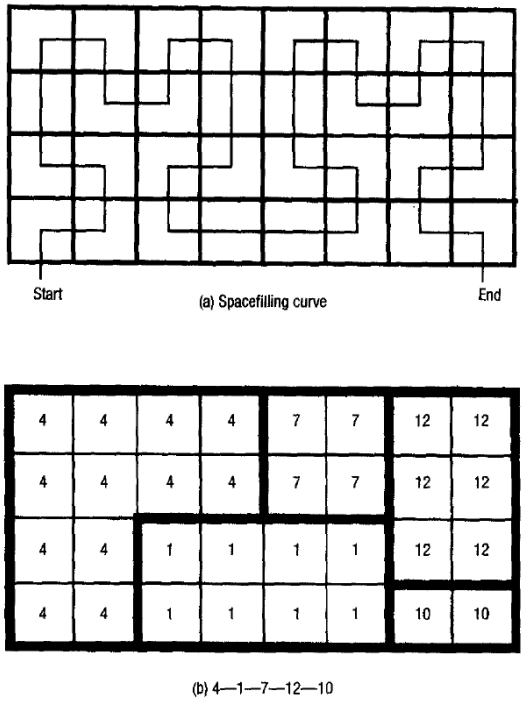


Figure 2.1: Discrete mapping of a solution vector to a block layout via spacefilling curve. Blocks with the same number are occupied by the same department. (Meller and Bozer [36])

Meller and Bozer [36] focus on problem instances with unknown department-to-floor assignments. The paper's main focus lies on algorithm speed when generating layouts: Because "run times are required to be reasonable and comparable", the authors propose a two-phase approach, in which the first phase focuses on determining good department-to-floor assignments and the second stage optimizes the individual facility positions, with departments permanently assigned to their respective floors. They use spacefilling curves for mapping solution vectors, like "4—1—7—12—10" in Figure 2.1, to a floor layout. The bounds of departments in the layouts are determined by their required areas specified

in the algorithm's input. It is highlighted that the constant unit-handling costs, for the vertical and horizontal directions, respectively, have a crucial impact. Especially in situations with a high "V/H ratio", that is, with high vertical material handling costs compared to the horizontal material handling costs, efficient floor assignments of departments are a deciding factor for cost-efficient layouts. The elevator positions are predefined by user input and their area is excluded from overlap-avoidance constraints. The results presented in the evaluation chapter indicate that splitting the tasks of optimizing department-to-floor assignments and constructing optimal layouts into two phases is a preferable strategy compared to a single-stage approach in which both problems are dealt with simultaneously.

Lee et al. [33] solve UA-MFFLPs that consider aisles and inner walls via a genetic algorithm. In their approach, the authors focus on two objectives: Minimizing MHC and maximizing adjacency requirements (ADJ) between departments. The objectives are merged into one function via the weighted-sum method (i.e., weighted scalarization). The number, positions and sizes of elevators are defined via input parameters. Overlaps with their area are not considered a constraint violation. They integrate the previously mentioned discrete spacefilling curve method proposed by Meller and Bozer [36] into a genetic algorithm. Comparative experiments demonstrate the efficiency of their approach.

Huang et al. [23] solve UA-MFFLPEs via three distinct stages. The first stage solves the department-to-floor assignment sub-problem via the Threshold Accepting method, a general-purpose algorithm proposed by Dueck and Scheuer [14] that is equally or better performing than the related Simulated Annealing procedure. Similar to the previous two papers, the layout generation in the second phase is done via spacefilling curves. Lastly, in stage three, the elevators are positioned via a heuristic that considers the center-of-mass of inter-floor material flows to guide placement decisions. Solutions are judged in respect to two objectives: The MHC and elevator (setup) costs (EC). The number of elevators in the final layouts is decided by the algorithm and their area is considered by resizing colliding departments post-hoc.

Karateke et al. [29] deal with a similar problem in their work. They also use a two-phase approach, first determining the department-to-floor assignments and then determining the layout of each floor. The departments have unequal areas (UA). For the second phase, they use the Dantzig-Wolfe decomposition algorithm, a linear programming (LP) method. They test their algorithm with more than eighty test cases, some of which include up to a hundred departments. They showed that the presented procedure was able to outperform similar algorithms used for this problem in related research, although frequently requiring higher runtimes.

A work that also investigates multi-floor layouts with unequal area departments is the one by Enayaty Ahangar et al. [15]. The authors utilize a Flexible Bay Structure (FBS), a special continuous UA-FLP formulation, that allows departments to be positioned only in parallel bays with varying widths. According to Konak et al. [32], FBS has the advantage of implicitly creating a desirable aisle structure, which eases the transfer from the block design to an actual facility layout. Besides the typical material handling metric,

the authors additionally consider elevator (EC), floor and land costs in their optimization process. The latter two objectives can be interpreted as functions aiming to maximize the density of the individual floors (FDEN) and of the overall layout (DEN), respectively. Furthermore, a maximum permissible aspect ratio is enforced to prevent deformed blocks. The procedure assumes exactly one full-service elevator located at the coordinates $(0, 0)$. The area of the elevator is being neglected in the optimization process.

In their paper, Ji et al. [28] also cover a FLP with multiple floors. They describe their problem variation as a "DFDRLP_SHVE": A double-floor double-row layout problem with separate human and vehicle elevators. Although the work is only about dual-floor layouts, it can broadly be classified as a MFFLPE, as (goods) elevators are being placed dynamically and their area is considered in the optimization procedure. Included are two main objectives: The classical material handling cost (MHC) metric and a novel employee movement cost (EMC) function, which is a flow handling cost metric for specific departments and elevators for humans. Consequently, they deal with two distinct elevator types in their model: One for goods and one for employee transportation. The elevator for employees is in a fixed location on the left side of a corridor. The goods elevators are placed dynamically. Just like the typical material flows between machines, the authors introduce human flows between machines. The proposed algorithm is tested in different scenarios that differ in their prioritization of the objectives. Depending on the prioritization, the algorithm places cubes near the human elevator, to minimize EMC, or shows a more even distribution of machines, to optimize MHC.

The work by Goetschalckx and Irohara [18] presents different approaches to problems of type MFFLPE-A and -B. Besides the core optimization, the department-to-floor assignment sub-problem is solved by the algorithm as well. Elevators "do not consume any area" and department dimensions are predefined by the input. To mitigate the increased complexity posed by multi-floor FLPs, the work presents and compares a wide variety of "acceleration techniques". Some of these optimization methods have been presented before by Sherali et al. [47], but only in the context of single-floor FLPs. Lastly, the presented experiments not only showed a noticeable performance boost of the optimizations but also highlighted the general superiority of solutions that allowed partial-service elevator types.

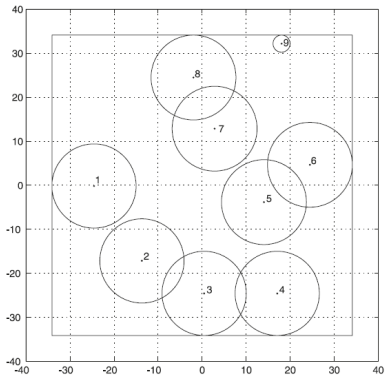
Hathhorn et al. [20] propose a very extensive MIP formulation of MFFLPE-A instances. The decision variables include "position, length and width of departments, the number of floors and size of the facility, the number and location of elevators". Decision variables regarding the property are rarely included in related literature on MFFLP(E)s. The objectives considered are MHC and building costs (BC). The latter goal can be interpreted as a blend of the two objectives for (floor) density maximization (DEN & FDEN). The authors apply lexicographic ordering to the objectives by assigning each of the objectives a priority. The algorithm then solves single-objective sub-problems in sequence. Elevators "are assumed to have an area of zero units" and are placed at the edges of departments. For the numerical experiments, the BC objective was prioritized over the MHC goal. Notably, the proposed algorithm is given a slack parameter that defines how much a

layout can additionally cost compared to solutions obtained in the first phase. This allows the procedure to trade a certain level of deterioration in the first objective (here BC) for improvements in the second goal (here MHC). Their tests not only identified suitable levels for the aforementioned slack parameter, but also showed that the acceleration techniques presented by Goetschalckx and Irohara [18] result in considerable runtime speedups.

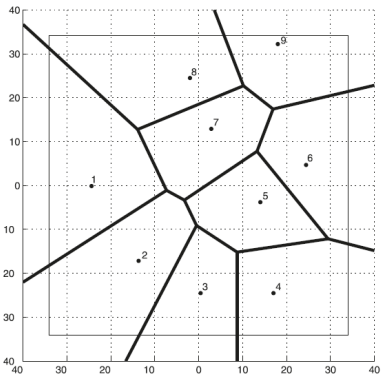
Izadinia et al. [25] cover a variation of the multi-floor layout problem, in which departments are of fixed size and can only be placed in predefined slots on each floor. Elevators are placed arbitrarily, and their area is considered in the non-overlap constraints. The focus of the paper is on generating robust multi-floor layouts. Consequently, instead of returning definitive values, their implementation returns ranges of values to account for uncertainties and unexpected limitations when translating layouts to the real world. They define a multi-floor layout to be robust if a rearrangement of departments is not necessary, as long as changes in the department's material demands stay below a predefined threshold referred to as the "protection level". The authors evaluated their approach by randomly generating different problem instances, with differing block dimensions and material demands. Although computationally more expensive, robust layouts outperform traditional layouts when it comes to handling uncertain MHC.

The extensive work by Ahmadi and Akbari Jokar [1] describes a three-staged approach for solving UA-MFFLPE-A instances. The first stage solves the department-to-floor assignments by employing a floor assignment formulation (FAF), originally presented by Meller and Bozer [36], to minimize vertical MHC. In the second and third stage the authors determine the relative positions of the departments by applying an adapted version of the JLAV method, presented by Jankovits et al. [27]. The abbreviation JLAV refers to the author's last names. In this method's first step, the department's relative positions are determined by simulating circles that attract and repel with respect to a certain target distance. In the second stage, to obtain the final layout, these circles are converted into a planar graph and subsequently into a Voronoi diagram. The diagram can then be interpreted as a two-dimensional matrix that describes the relative positions between every unique pair of departments. For every slot, the matrix contains a set of certain numbers that encode the relative horizontal and vertical positions of a pair of departments. Figure 2.2 shows an example of this procedure. Jankovits et al. [27] showed that their framework returns excellent results while keeping algorithm runtimes exceptionally low. Notably, Ahmadi and Akbari Jokar [1] investigate cases with fixed and cases with dynamic elevator positions. Especially for the latter case, in which the elevator positions are part of the decision variables, they adapted the evaluation to promote elevator dispersion. Like the paper by Jankovits et al. [27], their work showed outstanding results compared to existing approaches. Still, elevators are considered area-less, i.e. they are not respected in the non-overlap constraints.

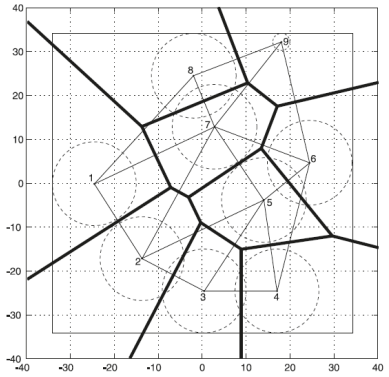
A general pattern can be identified when looking at the related research: MFFLP(E)s are mainly solved in respect to one or two objectives, via MIP formulations and in an Unequal Area context. Furthermore, the majority of papers analyzed do not consider



(a) Solution of the first stage model



(b) Corresponding Voronoi diagram



(c) Superposition of Delaunay triangulation

$$\begin{pmatrix} 0 & 21 & 11 & 11 & 11 & 11 & 11 & 22 & 11 \\ 0 & 0 & 11 & 11 & 11 & 11 & 22 & 22 & 22 \\ 0 & 0 & 0 & 11 & 22 & 22 & 22 & 22 & 22 \\ 0 & 0 & 0 & 0 & 22 & 22 & 22 & 22 & 22 \\ 0 & 0 & 0 & 0 & 0 & 11 & 22 & 22 & 22 \\ 0 & 0 & 0 & 0 & 0 & 0 & 12 & 12 & 22 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 22 & 22 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(d) Resulting relative position matrix

Figure 2.2: Layout generation process of a 9-department layout via the JLAV method (Jankovits et al. [27][p. 206])

elevator areas in their non-overlapping constraints.

Finally, in Table 2.1 we appended our proposed algorithm with 5 objectives for solving multi-floor facility layout problems with full- and partial-service elevators (MFFLPE-B). Most of the goals have been previously dealt with in related work; however, the port-connectivity (PC) objective is a novelty, because it is related to our discrete port-based algorithm design, as we will see in Chapter 3. Because we are dealing with more than 3 objectives, our problem is formally considered a many-objective optimization problem. This problem genre will be the focus of the next sections.

2.2 Many-objective Optimization

A multi-objective optimization problem (MOP) is comprised of two or more potentially conflicting objectives that need to be optimized at the same time. Many-objective optimization problems (MaOP) are types of MOPs that include more than 3 objectives. Throughout this thesis, we assume all problems to be minimization problems, that

is, lower objective values are more desirable. Formally, multi-objective minimization problems can be described as follows [58]:

$$\min_{\vec{x} \in X}(f(\vec{x}) = (f_1(\vec{x}), f_2(\vec{x}), ..., f_m(\vec{x})))$$

where

- $f(\vec{x})$ is an objective function vector that consists of m objective functions,
- X is the decision space and
- $\vec{x}$ is a decision vector.

2.2.1 Dominance-based Search

Because objectives are often conflicting, solutions that improve in one objective might worsen in another [61]. Thus, there are usually many trade-off solutions instead of one definitive best solution. Optimal solutions are part of the Pareto set and are considered Pareto optimal. These solutions are "mutually nondominated" and make up the Pareto front. The principle of domination will be explained shortly. The Pareto front is usually unknown, thus approximation algorithms are commonly used for said problems. Without any prioritization by decision makers, all solutions in the Pareto set are considered equally good [58]. Figure 2.3 shows an example of a Pareto front in the context of a bi-objective minimization problem.

In Pareto-based MOPs, the fitness assignment is based on the dominance relation between solutions and the density, i.e. the crowdedness of the objective space area a solution finds itself in. Algorithms utilizing dominance-based evaluation measures follow two ultimate goals regarding the solution set: They try to minimize the distance to the real Pareto front and to maximize the distribution over the Pareto front. These goals are often simply referred to as "convergence" and "diversity", respectively. A solution is considered Pareto optimal, or "non-dominated", when it is impossible to improve one objective without worsening another one. The dominance relation can formally be described as follows [56, 61]:

Solution $\vec{x_1} \in X$ dominates solution $\vec{x_2} \in X$ if and only if

- $f_i(\vec{x_1}) \leq f_i(\vec{x_2})$ for all $i \in \{1, 2, ...m\}$ and
- $f_j(\vec{x_1}) < f_j(\vec{x_2})$ for at least one $j \in \{1, 2, ..., m\}$

In contrast, aggregation-based algorithms compress multiple objective functions into a single scalar fitness value. This evaluation measure essentially considers the problem as single-objective and is usually more suitable for problems with fewer conflicting goals, as

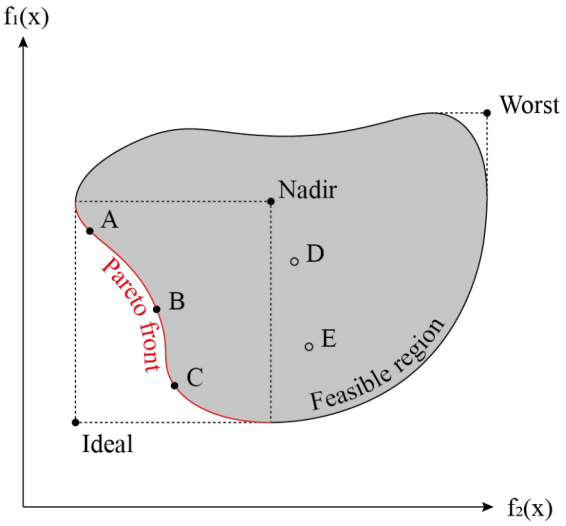


Figure 2.3: Example of a solution set of a bi-objective minimization problem. Solutions A, B & C are part of the Pareto front (highlighted red), and thus non-dominated. D is dominated by B & C. E is dominated by C. The figure also shows the ideal, nadir and worst points (see Section 2.2.3). Note that the ideal and the worst points are non-reachable as they are outside the feasible region.

aggregation techniques, e.g. the weighted-sum approach, often "miss" trade-off solutions in non-convex Pareto fronts. Furthermore, Pareto- or dominance-based evaluation has the advantage of guiding the search in multiple directions at the same time in a single simulation run [58]. For the remainder of this thesis, we will refer to the fitness evaluation by aggregating objectives into a single scalar value as scalarization-based evaluation. Likewise, the fitness evaluation based on Pareto-dominance (strength & diversity values) will subsequently be referred to as dominance-based evaluation.

2.2.2 Meta-Heuristics, Genetic Algorithms & SPEA2

As there are no unknown variables, a general FLP can be classified as a Deterministic Combinatorial Optimization Problem (DCOP) [8]. Furthermore, FLPs are known to be NP-hard [18]. For many NP-hard DCOPs, algorithms that guarantee to find the optimal solution may need exponential computation time [8], rendering them impractical. Thus, to overcome these complexities, many research fields that deal with NP-hard problems, including the field of digitally optimizing production layouts, are focused on the design of more capable procedures, procedures that can approximate the Pareto front. Regarding FLPs, so-called meta-heuristics are the most prevalent solving approach in recent research [22].

Meta-heuristics belong to the class of approximate procedures that find "as good as possible" solutions in reasonable computation times. Their popularity stems from their underlying philosophy of balancing "the exploitation of the accumulated search experience"

(intensification) and the "exploration of the search space" (diversification) [8]. There are several different sub-genres of meta-heuristics, many of them are bio-inspired, for example, by ant colonies [13], bird flocks [30], corals [45], immune systems [31] or evolutionary processes as a whole [52]. The classification review by Hosseini-Nasab et al. [22] found that the most popular choice for solving FLPs is the use of genetic algorithms, a subclass of evolutionary algorithms.

Evolutionary algorithms consider a working memory of potential solution candidates [12]. A subclass, the genetic algorithms (GA), imitates the Darwinian process of species evolution. Here, the working memory is usually referred to as the population and the solution candidates are called individuals or chromosomes. In genetic algorithms, an iteration, often referred to as a generation, usually consists of the following stages [61, 49]:

- **Evaluation**: In the evaluation stage, each solution is assigned a fitness value according to the objective functions.
- **Selection**: The selection stage updates the population by considering the fitness of solutions in the existing population and in the newly generated offspring of the recombination/mutation stages.
- **Crossover**: The recombination, or crossover stage "takes a certain number of parents and creates a predefined number of children by combining parts of the parents" [61].
- **Mutation**: Finally, the mutation stage randomly modifies parts of solutions to promote diversity in the population.

Recently, there has been growing interest of the research community in elitist variations of genetic algorithms, due to their superior performance compared to traditional GA implementations [12]. Elitism prevents the loss of good solutions by keeping a separate memory of promising solutions, often referred to as the archive [61].

One of the latest variations of genetic algorithms is the Strength Pareto Evolutionary Algorithm 2 (SPEA2) by Zitzler et al. [60]. Like many more recent algorithms, SPEA2 is an elitist procedure: Each iteration t includes an external archive $\overline{P}_t$, besides the traditional population P_t, that contains the fittest solutions. What makes SPEA2 (and SPEA) unique is the special fitness assignment it uses: Each solution i in the archive is assigned a strength value $S(i)$ which represents the number of solutions i dominates, both in the archive and in the population. With $\succ$ denoting the dominance relation, the strength of the i-th solution is formulated as follows:

$$S(i) =\mid \{j \mid j \in P_t \cup \overline{P_t} \wedge i \succ j\} \mid$$

On the basis of the strength values, a solution i is assigned a raw fitness value $R(i)$ by summing up the strengths of it's dominators:

$$R(i) = \sum_{\mathbf{j} \in P_t \cup \overline{P_t}, j \succ i} S(j)$$

Naturally, solutions with lower raw fitness are better.

Furthermore, density information is included to promote diversity in the approximated Pareto front. The density $D(i)$ of a solution i is calculated by taking the inverse of the distance to the k-th nearest neighbor, denoted as σ_i^k. A common setting for k is $\sqrt{\mid P_t \mid + \mid \overline{P_t} \mid}$. $D(i)$ is then calculated via the following formula:

$$D(i) = \frac{1}{\sigma_i^k + 2}$$

Finally, the fitness value $F(i)$ for the i-th solution is obtained as follows:

$$F(i) = R(i) + D(i)$$

2.2.3 Shift-based density, ideal & nadir point estimation

One common issue that arises with MaOPs is an insufficient selection pressure towards the Pareto front. This is due to the fact that the proportion of non-dominated solutions in a population tends to rapidly grow with the number of objectives incorporated in the optimization process. This ultimately leads to a "stalemate situation" between many of the solutions, resulting in subpar convergence. A usual symptom of this phenomenon is that the final solution set of an optimization simulation may contain many solutions that perform very well in most objectives, but very poorly in a single other one. These so-called "dominance resistant solutions" might be in low-density areas; however, they are nowhere near the desired Pareto front [34].

A suitable strategy to counter dominance resistant solutions is by using shift-based density estimation (SDE), initially proposed by Li and Liu [34]. This density estimator unifies diversity as well as convergence information into one measure. It is based on moving solutions with poor convergence but good (i.e. low) density into more populated areas. After shifting, these solutions will more likely be discarded due to their now inflated density values.

To keep the density estimations unbiased, SDE depends on the scaling, or normalization, of objective functions. However, because the Pareto front is usually unknown, we need to estimate the ideal and nadir points (see Figure 2.3 for a visualization of said points). In related literature, one can find a variety of different estimation strategies, ranging from naive to extremely intricate approaches [21]. The vector $z^* = (z_1^*, z_2^*, ..., z_m^*)$ describes the ideal point and consists of the minimum value of each objective over the decision space. The worst point $z^w = (z_1^w, z_2^w, ..., z_m^w)$ is the opposite of the ideal point and holds the maximum values for each objective. The vector $z^{\text{nad}} = (z_1^{\text{nad}}, z_2^{\text{nad}}, ..., z_m^{\text{nad}})$ is similar to the worst point, however, it only considers the worst values found in the Pareto front

[21]. The following sections describe the ideal and nadir point estimation approaches we will subsequently use in the dominance-based implementation. It is important to note that the scalarization-based implementation will assume extreme values encountered in previous simulations for z_i^{nad} and z_i^* to normalize the individual objectives. These values will be constant throughout the optimization procedure. This is commonly known as offline normalization [21].

Ideal point estimation

For each iteration t, we update z_i^* for each objective i via the following strategy described by He et al. [21]:

$$z_i^* = \min\{z_i^*, \min_{x' \in P_t} f_i(x')\}, i \in \{1, 2, ..., m\}$$

where

- P_t is the offspring population generated in iteration t.

In other words, to acquire the new ideal value for each objective i, we take the minimum of the currently known ideal objective value and the minimum objective value found in the newly generated offspring.

Nadir point estimation

For each iteration t, we update z_i^{nad} for each objective i via the following strategy described by He et al. [21]:

$$z_i^{\text{nad}} = \max_{x \in \overline{P_t} \cup P_t} f_i(x), i \in \{1, 2, ..., m\}$$

where

- $\overline{P_t}$ is the archive in iteration t and
- P_t is the population in iteration t.

In other words, to acquire the new nadir value for each objective i, we take the maximum objective value found in the newly generated population and the existing archive.

Objective Normalization

For every objective function f_i we apply a standard Min-Max-Normalization described as follows:

$$f_i^{Normalized} = \max\left(0, \min\left(1, \frac{f_i^{Raw} - z_i^*}{z_i^{\text{nad}} - z_i^*}\right)\right)$$

where

- f_i^{Raw} is the raw value of objective i,
- z_i^* is the (current) ideal value for objective i and
- z_i^{nad} is the (current) nadir value for objective i.

2.2.4 Performance Measures

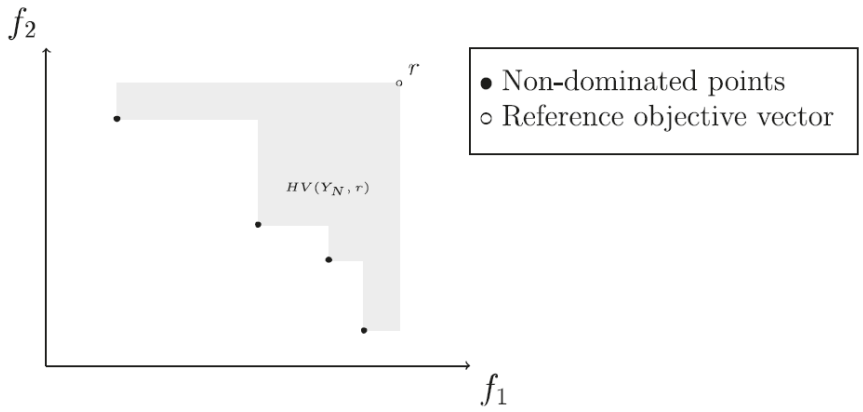


Figure 2.4: Illustration of the Hypervolume of a Pareto front approximation Y_N via a reference point r in the context of a bi-objective minimization problem (Audet et al. [7][p. 412])

To measure the performance of algorithms approximating Pareto fronts, there are a number of different indicators to be found in related research [7]. Popular examples are the Hypervolume (HV) [59], Inverted Generational Distance Plus (IGD+) [24] and Spacing [7]. Initially proposed by Zitzler and Thiele [58], the HV indicator is commonly used to evaluate Pareto-based algorithms and describes the volume of the dominated space of the (approximated) Pareto front, relative to an arbitrary reference point $r \in \mathbb{R}^m$, that is in any case dominated. It is called "Hyper"-volume because in MaOPs we deal with more than 3 objectives. Figure 2.4 shows an illustration of the Hypervolume of a Pareto front approximation in the context of a bi-objective minimization problem. It is one of the few indicators that guarantee Pareto compliance. Pareto compliant measures of a Pareto front approximation a are guaranteed to be greater than those of another approximation b whenever front a dominates front b [9]. IGD+ represents the average minimal distance from a solution of a Pareto front to the closest point of an existing approximated Pareto front and guarantees weak Pareto compliance. Lastly, according to Audet et al. [7], the Spacing indicator "captures the variation of the distance between elements of a Pareto front approximation". For the HV, higher values are better; for the IGD+ and Spacing indicators lower values are desirable.

2.3 Rhinoceros 3D & Grasshopper

The computer aided design (CAD) software Rhinoceros 3D [4] and its integrated visual programming environment Grasshopper [11] are tools commonly used by civil engineers and architects. The integrated Grasshopper plugin allows for parametric node-based "programming" of dynamic layouts, and is an easy-to-use no-code alternative for users with limited coding knowledge to generate elaborate geometry. Initially, we planned on implementing the whole optimization algorithm inside Grasshopper; however, after comparing execution runtimes between Grasshopper and native C# code, we decided to split the optimization code from the visualization fragments and design the algorithm to run separately in a console app. We observed that native execution is more than twice as fast. Lastly, this decision was influenced by the limited debugging capabilities of the integrated Grasshopper code editor, which had a noticeable impact on the development process. Still, there are many different plugins available for Grasshopper that provide parametric optimization capabilities out of the box, like Galapagos [51], Octopus [54], Opossum [50] and Wallacei [55]. These tools offer a number of implementations of state-of-the-art evolutionary algorithms, like SPEA2 (Galapagos) or NSGA-II (Wallacei) [56]. However, due to the previously mentioned performance concerns and our highly specialized use case, we decided to implement an independent console app in the C# programming language (see Section 3.7.3 for details on the implementation environment). Thus, in our thesis, Rhinoceros 3D and Grasshopper were mainly utilized for importing, visualizing, and interacting with the generated output files.

CHAPTER 3

Multi-floor Production Layout Optimization with a Port-based Evolutionary Algorithm

3.1 Algorithm Overview

Because we are dealing with multi-floor layouts, we need to place production cubes as well as elevators in the layout. Compared to elevators, validly placing production cubes is less cumbersome: To prevent potential collisions, the algorithm has to respect the boundaries of cubes on a single floor only. However, as elevators naturally span across multiple floors, the necessary collision checks will be much more extensive. More importantly, elevator positions play a crucial role in minimizing the material handling costs of a layout effectively. These two reasons suggested a two-phase optimization procedure: One phase for a parallel optimization of production cube as well as elevator positions, with elevators being allowed to overlap with other layout elements, and another phase solely focused on optimizing production cube positions, with the elevators fixed in place. Figure 3.1 shows a schematic of our two-phased evolutionary algorithm design. The following sub-sections explain the individual phases in more detail.

3.1.1 Phase 1: Transient elevator phase

In the first phase, the positions of elevators and production cubes are optimized in parallel. However, there is a catch: Instead of regular collisions, so called transient elevators are allowed to collide with production cubes. This initial relaxation of the elevator overlap constraint allows for efficient testing of many different elevator position variations, while keeping the production cube placements unbothered. Still, transient elevators are not allowed to overlap with other elevators. In this phase, no solid elevators are being used,

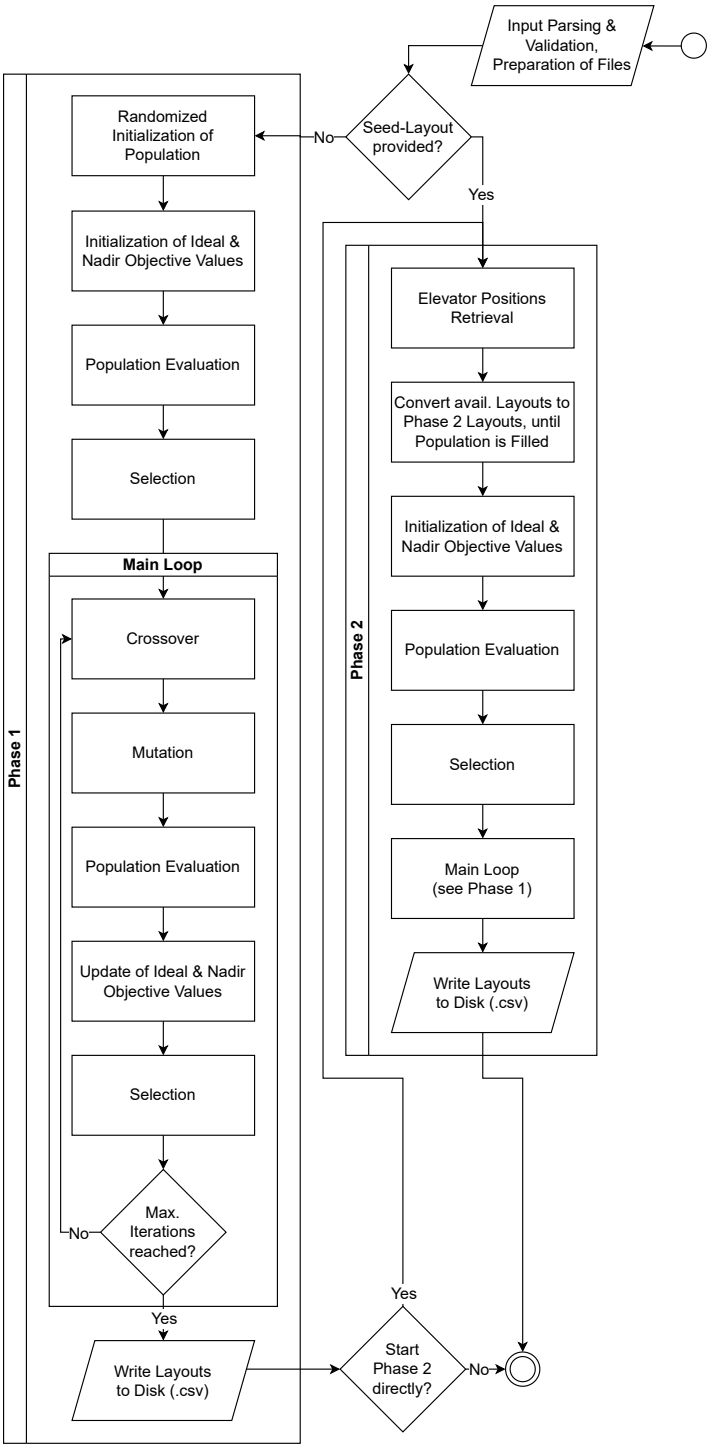


Figure 3.1: Overview of the Two-Phased Evolutionary Algorithm

i.e. immovable elevators that require an overlap-free placement ($N^{SE} = \emptyset$). The phase starts by randomly initializing layouts to fill up the population set. Then the ideal and nadir values of the individual objectives, which are later needed for the evaluation stages, are initialized via the freshly generated population according to the policy laid out in Section 2.2.3. After this bootstrapping procedure is completed, the algorithm enters the main loop that contains the typical elements of a genetic algorithm: Crossover, Mutation, Evaluation and Selection. Like in the beginning, the algorithm updates z_i^{nad} and z_i^* for each objective i in each iteration. Once the iterations have reached the configured limit, the main loop is exited and phase 1 finalizes by writing the found results onto the disk in CSV format. This also includes additional diagnostic data about the algorithm's performance, for instance, the milliseconds required for each iteration. The procedure then continues with phase 2 or stops, depending on the configuration. In the case of continuation, the final phase is started by being handed over the layouts found in the first phase.

3.1.2 Phase 2: Solid elevator phase

For the initialization of the second phase, we must differentiate between two cases:

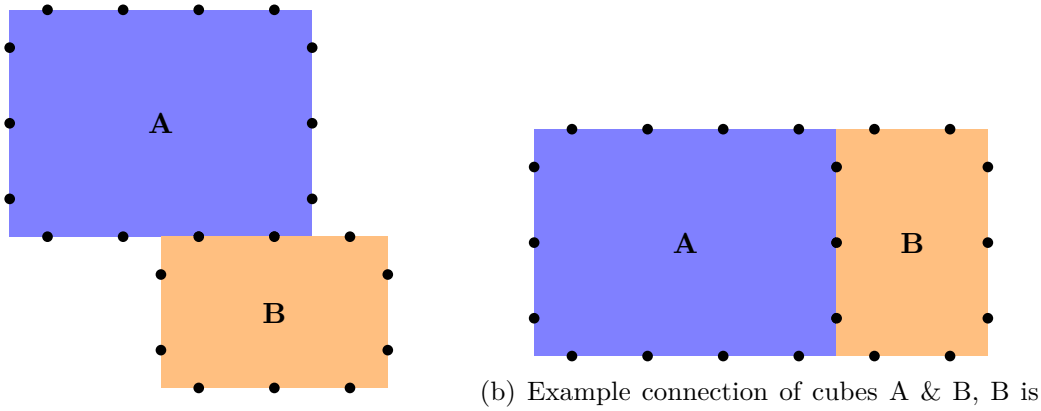
- If the algorithm continued right away, the second phase starts by retrieving the elevator positions of the overall best layout of the handed-over phase 1 layouts. All of the layouts are then converted via a conversion sub-procedure and stored in the population set. For each existing solution, this sub-procedure creates a new layout by first placing the solid elevators onto their respective positions and then identifying the production cubes that would have conflicting bounds with these elevators. The mechanism tries to mitigate the deterioration of objective scores by focusing on minimal positional changes and the preservation of relative positions between production cubes.

- If the algorithm was bootstrapped via a seed layout specified in the configuration file, it selects the elevators from that solution. The population is then filled via repeatedly converting the seed layout. As this procedure is partly randomized, solutions will slightly differ in their cube positions.

The algorithm then proceeds normally with the main loop, which is structurally identical to the one from the first phase. Naturally, for the second phase, solid elevators are not considered in the crossover and mutation stages, and no transient elevators are being used ($N^{TE} = \emptyset$).

3.2 Ports

Initial tests showed that, without the assistance of discrete positioning anchors, validly snapping cubes to each other is difficult to implement and computationally expensive. As



(a) Example connection of cubes A & B

(b) Example connection of cubes A & B, B is rotated

Figure 3.2: Example arrangements of (port-)cubes

move operations for cubes are a core sub-procedure of the algorithm, with thousands of such operations happening in each iteration, early implementations were suffering from unacceptably long runtimes. Because of this, we decided to implement ports. Ports are discrete positioning anchors at each side of a cube. They start at the $\frac{r}{2}$ coordinate of each side and are then placed incrementally with a distance of exactly r, where r is the port resolution. This limits the dimensions cubes can have, as side lengths have to be at least r and divisable by r without remainder for ports to be placed correctly. In our work, we assumed a port resolution of 1 (meter). The use of ports not only simplifies internal snapping operations, it converts the problem into a discrete one. This drastically limits the problem complexity, as cubes now no longer have infinite options to be placed in relation to each other. Figure 3.2 shows two example constellations of two so-called port-cubes. Ports are only used for positioning and do not affect objective functions, for instance, the material flow cost calculations.

3.3 Constraints

This section details which constraints a layout needs to fulfill to be considered valid. There are a total of 4 constraints. First, we define the variables stated below. We omit denoting to which layout l a set belongs to, e.g. N_l, if it is not strictly necessary.

- N^{TE} is the set of all transient elevators in a layout,
- N^{SE} is the set of all solid elevators in a layout,
- N^{E} is the set of all elevators in a layout, i.e. $N^{E} = N^{TE} \cup N^{SE}$,
- N^{PC} is the set of all production cubes in a layout,
- N^{v} is the set of cubes present in the v-th floor of a layout,

- N is the set of all cubes in a layout, i.e. $N = N^E \cup N^{PC}$,
- A_D is the set of adjacency relationships with a value set to 1 (= desired),
- A_U is the set of adjacency relationships with a value set to -1 (= undesired),
- B_{C_i} is the 2D bounding rectangle of the i-th cube,
- $B_{(X_1,X_2)}$ is the 2D bounding rectangle spanned by the points X_1 (lower left) and X_2 (upper right),
- B_P is the 2D bounding rectangle of the property,
- $Area_B$ is the area of the 2D bounding rectangle B and
- $v_{C_i}^{Start|End}$ is the start/end floor index of the i-th cube. For production cubes $v^{Start} = v^{End}$ holds.

As the following formulas will reappear in multiple constraints, we define them here:

$$Overlapping(C_1, C_2) = \neg(x_{\max}^{B_{C_1}} \leq x_{\min}^{B_{C_2}} \vee x_{\min}^{B_{C_1}} \geq x_{\max}^{B_{C_2}} \vee y_{\max}^{B_{C_1}} \leq y_{\min}^{B_{C_2}} \vee y_{\min}^{B_{C_1}} \geq y_{\max}^{B_{C_2}})$$

The formula above returns true if and only if the 2D bounding rectangles of two cubes C_1 and C_2 overlap.

$$HaveSharedFloors(C_1, C_2) = v_{C_1}^{Start} \leq v_{C_2}^{End} \wedge v_{C_1}^{End} \geq v_{C_2}^{Start}$$

The formula above returns true if and only if two cubes C_1 and C_2 share some floors.

3.3.1 c_1: Production cubes must not overlap with other production cubes

Production cubes are not allowed to overlap with each other. This constraint can be formulated as follows:

$$c_1 = \forall C_1, C_2 \in N^{PC}(C_1 \neq C_2 \wedge HaveSharedFloors(C_1, C_2) \Rightarrow \neg Overlapping(C_1, C_2))$$

3.3.2 c_2: Elevators must not overlap with other elevators

Elevators are not allowed to overlap with other elevators. This constraint can be formulated as follows:

$$c_2 = \forall E_1, E_2 \in N^{E}(E_1 \neq E_2 \wedge HaveSharedFloors(E_1, E_2) \Rightarrow \neg Overlapping(E_1, E_2))$$

3.3.3 c_3: Solid elevators must not overlap with other cubes

Solid elevators are not allowed to collide with other cubes. This constraint is only relevant for layouts in phase 2, as $N^{SE} = \emptyset$ holds for all layouts in phase 1. This constraint can be formulated as follows:

$$c_3 = \forall E \in N^{SE} \forall C \in N(E \neq C \wedge HaveSharedFloors(E, C) \Rightarrow \neg Overlapping(E, C))$$

3.3.4 c_4: Cubes must be fully inside the property

Cubes, production cubes as well as elevators, are not allowed to be, partially or fully, outside the property bounds. This constraint can be formulated as follows:

$$c_4 = \forall C \in N(x_{\min}^{B_P} - x_{\min}^{B_C} \leq 0 \wedge x_{\max}^{B_C} - x_{\max}^{B_P} \leq 0 \wedge y_{\min}^{B_P} - y_{\min}^{B_C} \leq 0 \wedge y_{\max}^{B_C} - y_{\max}^{B_P} \leq 0)$$

The algorithm tries to fix outside cubes "on the spot" after each iteration. Layouts that could not be fixed after an iteration completed it will be discarded immediately (see Section 3.6.2).

3.4 Objectives

The algorithm considers a total of 5 objective functions in the fitness assignment. For all objectives lower values are desirable. The following sections describe the individual objectives in more detail.

3.4.1 f_1: Maximize Port Connectivity Objective

This objective rewards layouts with a high number of occupied ports. Ports are explained in Section 3.2. High port connectivity usually results in layouts with higher density and less unused gaps. The raw value is calculated via the following formula:

$$f_1^{Raw} = \sum_{C \in N} P_C^{Open}$$

where

- P_C^{Open} is the number of open ports of a cube C.

3.4.2 f_2: Minimize Material Flow Distances Objective

This objective rewards layouts with small material flow distances and is one of the most commonly used objectives related to solving FLPs [36, 25, 15]. Material flows in the input are initially denoted as triplets (C_1, C_2, i), with C_1 being the source production

cube, C_2 being the destination production cube, and i being the intensity. Section 3.7.1 describes the input for material flows in more detail. For inter-floor material flows, these triplets are extended by a set of intermediary elevator stops, depending on the routes calculated by the greedy strategy in the algorithm's evaluation sub-procedure. For each cube participating in a material flow that has an elevator as the next required stop, the mentioned greedy strategy uses the closest elevator with sufficient free capacity to complete the necessary floor transition. The order of route calculation is decided by the intensity of the material flows. In each iteration, routes are recalculated. Ultimately, a processed material flow can have up to $2 + (v - 1) = v + 1$ participating cubes, where v is the number of floors the facility has. Same-floor material flow distance calculations are done via centroid-to-centroid Rectilinear distances. The distances of vertical floor transitions of materials are considered 0, as they are constant values equal for all flows and can thus be neglected in the optimization. The raw value is calculated via the following formula:

$$f_2^{Raw} = \sum_{j=1}^{k} \left(\left(\sum_{p=1}^{q_j - 1} \underbrace{(\mid x_{C_p^{Center}} - x_{C_{p+1}^{Center}} \mid + \mid y_{C_p^{Center}} - y_{C_{p+1}^{Center}} \mid)}_{\text{Rectilinear distance}} \right) * \left(\frac{i_j}{\sum_{l=1}^{k} i_l} \right) \right)$$

where

- k is the number of material flows in the layout,
- q_j is the number of participating cubes in the material flow j (e.g. Cube A $\rightarrow$ Elevator A $\rightarrow$ Cube B),
- C_p^{Center} is the 2D center point of cube p and
- i_j is the flow intensity of material flow j

This formula sums up the rectilinear distances of the cubes that are participating in each material flow and then weighs the respective distances by their share compared to the overall material flow volume. This means that distances of high intensity material flows have a more severe impact on a solution's fitness measure.

3.4.3 f_3: Maximize Adjacency Fulfillment Objective

This objective rewards layouts that comply with adjacency requirements. In the input, adjacency requirements are modeled as triplets (C_1, C_2, s), with C_1 being the first production cube in the relationship, C_2 being the second production cube in the relationship and s being the goal state of the two cubes. This goal state can either be -1, 0 or 1, corresponding to an "undesired", an "indifferent", or a "desired" adjacency, respectively. Section 3.7.1 describes the input for adjacencies in more detail. The raw value is calculated via the following formula:

$$f_3^{Raw} = (\mid A_D \mid + \mid A_U \mid) - \left(\sum_{j=1}^{A_D} (C_1^j \smile C_2^j) + \sum_{j=1}^{A_U} (C_1^j \frown C_2^j) \right)$$

where

- C_1^j is the first cube of the adjacency relationship j,
- C_2^j is the second cube of the adjacency relationship j and
- $\smile$ describes a predicate that is 1 (true) if and only if cubes C_1^j and C_2^j are connected via ports ("touching"), otherwise 0 (false).
- $\frown$ describes a predicate that is 1 (true) if and only if cubes C_1^j and C_2^j are not connected via ports ("touching"), otherwise 0 (false).

3.4.4 f_4: Maximize Multi-Floor Layout Density Objective

This objective rewards high-density layouts, i.e. layouts that have a small overall area. The overall area is is obtained by calculating the area of the 2D rectangle that is created via the two outmost points of all cubes. The raw value is calculated via the following formula:

$$f_4^{Raw} = 1 - \frac{\sum_{i=1}^{n}(Area_{B_{C_i}})}{Area_{B_{(L^l,L^u)}}}$$

$$L^l = (\min_{C_i \in N} x_{min}^{B_{C_i}}, \min_{C_i \in N} y_{min}^{B_{C_i}})$$

$$L^u = (\max_{C_i \in N} x_{max}^{B_{C_i}}, \max_{C_i \in N} y_{max}^{B_{C_i}})$$

where

- L^l is the 2D outmost lower left point of the multi-floor layout and
- L^u is the 2D outmost upper right point of the multi-floor layout.

3.4.5 f_5: Maximize Floor Layouts Density Objective

This objective rewards layouts that have dense floors. It is similar to the previous objective, however, only considers the areas of the individual floors and does not put these areas in relation to the multi-floor layout area. The raw value is calculated via the following formula:

$$f_5^{Raw} = \sum_{v=0}^{w-1}\left(1 - \frac{\sum_{i=1}^{N^v}(Area_{B_{C_i}})}{Area_{B_{(L_v^l,L_v^u)}}}\right)$$

$$L_v^l = (\min_{C_i \in N^v} x_{min}^{B_{C_i}}, \min_{C_i \in N^v} y_{min}^{B_{C_i}})$$

$$L_v^u = (\max_{C_i \in N^v} x_{max}^{B_{C_i}}, \max_{C_i \in N^v} y_{max}^{B_{C_i}})$$

where

- w is the number of floors the layout has,
- L_v^l is the 2D outmost lower left point of the sub-layout for floor v and
- L_v^u is the 2D outmost upper right point of the sub-layout for floor v.

3.5 Assumptions

The algorithm is based on the following assumptions:

- The property dimensions are the same for all floors
- The dimensions of cubes and elevators are fixed
- Elevators are assumed to have a square 2D surface
- Because we assume a port resolution of 1 meter, all dimensions must be of $\mathbb{N}$ and larger than 0
- Cubes are pre-assigned to their respective floors
- All provided elevators will be placed, even if they are not necessary capacity-wise
- All elevators are considered to be bi-directional
- Production cubes must use elevators to complete inter-floor material flows

3.6 Genetic Stages of the Main Iteration Loop

As mentioned before, genetic algorithms are fundamentally based on three stages: Crossover, Mutation and Selection. After the algorithm is primed, the main loop is entered and executed as many times as specified in the configuration file (see Section 3.7.2). The following sections describe the genetic stages in more detail, including the utilized crossover and mutation operators as well as important sub-algorithms.

3.6.1 Crossover

In the crossover phase, the algorithm generates $\mid P_t \mid$ new solutions by conducting binary tournaments on the archive. A binary tournament is a popular choice in genetic algorithms for choosing suitable "parents" to create new solutions [61]: Two randomly chosen individuals compete against each other and the one with the better (i.e. lower) fitness wins the tournament. This is done twice for each potential child to obtain the two, potentially identical, parents. These solutions are then used in the crossover procedure to generate a new offspring layout. The upcoming sub-sections discuss the crossover operators denoted with χ_i, which gradually generate child layouts from scratch by appending cubes from different parents until all expected elements are present.

$\chi_1(n, p)$: Add percentage of remaining production cubes

This procedure accepts two arguments: One is a float $n \in [0, 1]$ and the other is a set of references to parent layouts, e.g. $p = \{P_2\}$. The former parameter describes the percentage of yet-to-be-placed production cubes that should be added to the new child layout, and the latter parameter describes the source parent from which to take the positional data for the randomly selected production cubes. If the second parameter contains both parents, then for each production cube, the source parent is selected randomly. This method tries to be "restorative", by attempting to recreate (port-)connections cubes previously had in parent layouts (see Algorithm 3.1).

Algorithm 3.1: Generic graceful connection restoration sub-procedure $SnapAddGracefully$ for a passive cube $c^{Passive}$, an active to-be-moved cube c^{Active}, a desired port constellation p, a floor v, a fallback set of cubes $C \subseteq N_l^v$ and a layout l

```
   /* Try via exact ports p & and desired partner c^Passive        */
1  if ¬SnapAdd(c^Passive, c^Active, p) then
       /* Try via any ports & and desired partner c^Passive        */
2      if ¬SnapAdd(c^Passive, c^Active) then
           /* Try via any ports & any partner cube of C            */
3          if ¬SnapAdd(C, c^Active) then
               /* Try via any ports & any same-floor cube (always
                  succeeds)                                         */
4              SnapAdd(N_l^v, c^Active)
5          end
6      end
7  end
```

χ_2: Crossover adjacency add

This procedure exclusively considers production cubes that are a part of the desired adjacency requirements (A_D). Its main focus is to pass on as many of the cube constellations that led to satisfied adjacency requirements (f_3) in the parent layouts onto the child. In each iteration of the function's main while-loop, the production cubes already present in the child layout are being looked at. The algorithm then selects the production cube that has the highest number of yet-to-be placed (i.e. not in N_{Child}^{PC}) desired adjacency partners. Next, the procedure determines the parent in which the selected department is connected to the most desired adjacency partner cubes, and tries to attach these to the selected production cube via the exact port constellations present in said parent. This is by far the most complex crossover operation in our implementation.

χ_3: Crossover elevator add

This procedure is only applied in phase 1 and chooses a random parent for each elevator from which to copy the position for the new elevator in the child layout.

Crossover structure

Preliminary experiments showed that generating offspring solutions of parent layouts via the procedure described in Algorithm 3.2 is effective.

Algorithm 3.2: Crossover procedure structure for generating offspring layouts

```
1 χ1(0.2, {P1});
2 χ1(0.2, {P2});
3 χ2();
4 χ1(1.0, {P1, P2});
5 χ3(); /* Omitted in phase 2                                    */
```

3.6.2 Mutation

In the mutation phase, we apply different mutation operators denoted as μ_i to production cubes and transient elevators. Production cube mutations are identical for both phases. Elevator mutations are only applied to (transient) elevators in the first phase, as elevators are considered final in their positions in phase 2. The main goal of the mutation operations for transient elevators is to explore potentially promising positions that improve the material handling costs (objective function f_2) of layouts. The following sub-sections give an overview of the individual mutation operators.

μ_1: Re-Add to random connected member

This operation re-attaches the mutation candidate to a random production cube that it is already connected to. It promotes higher levels of port connectivity (objective function f_1) as well as higher floor layout density (objective function f_5), and is only applied to production cubes (N^{PC}).

μ_2: Add randomly

This operation attaches a random production cube or elevator to any other cube in the layout. This is a high-entropy operation, as cubes are completely disintegrated from their current position. It promotes an extensive exploration of the search space and is applied to all movable cubes (N^{PC}, N^{TE}).

μ_3: Add to desired member (forced)

This operation snaps a random production cube to any other cube to which it wishes to be connected, according to the adjacency input (see Section 3.7.1). If the selected cube

has more than 50% of (desired) adjacency requirements already fulfilled, this mutation is aborted, as repositioning the production cube would most likely worsen the layout quality. This is a forced operation, which means that potential overlaps between the repositioned cube and existing passive production cubes are being resolved by repositioning the latter ones. Thus, this is an operation that potentially affects multiple cubes. Positions that result in overlaps with solid elevators in phase 2 are prohibited. This operation is designed to improve the adjacency fulfillment rate (objective function f_3) and is applied to all movable cubes (N^{PC}, N^{TE}).

μ_4: Re-Add tightly

This operation attempts to determine the position of the to-be mutated production cube relative to all other cubes it is connected to, which yields the highest port connectivity. This is essentially a greedy sub-procedure and can thus be considered computationally expensive. However, it strongly promotes higher levels of port connectivity (objective function f_1). It is only applied to production cubes (N^{PC}).

μ_5: Swap (forced)

This operation selects two random production cubes of the same floor and swaps their positions. This is a forced operation: It moves passive production cubes that now overlap with the newly positioned production cubes to a different location. For elevators, two random elevators with the same floor range (e.g. elevators ranging from floor 0 to floor 2) are being randomly selected and swapped. If no candidates for swapping exist, this operation is aborted. This operation promotes an extensive exploration of the search space and is applied to all movable cubes (N^{PC}, N^{TE}).

Mutation structure

Early tests showed that mutating cubes via the procedures described in Algorithm 3.3 and Algorithm 3.4 is effective. We use the same annotations as in Section 3.3. For both phases, the production cubes are mutated identically. Elevator mutations are naturally omitted in phase 2. The two procedures are very similar structurally: Both differentiate between single-cube and dual-cube operations. Each of those sets has a 50% chance of being selected. Then a random operation of the selected set is applied to the current cube. As previously stated, elevators must have an identical floor range ($v^{Start|End}$) to qualify for a dual mutation operation.

A common byproduct of mutation operations is the frequent appearance of connected sub-groups of production cubes that are not directly adjacent to the rest of the layout. Unsurprisingly, this is problematic in several respects: Layouts have longer material transport distances, worse density and unused adjacency potential. To repair these artifacts, there is a sub-procedure in place called $FixIslands$. The function attempts to repair the aforementioned groups by moving them to a pre-determined pivot island. This pivot island is determined by selecting the largest island present in the layout/floor.

Algorithm 3.3: Mutation procedure for production cubes of a layout l

```
1  μ^PC_SingleOp ← {μ1, μ2, μ3, μ4};/* Set of
     available single-cube mutations   */
2  μ^PC_DualOp ← {μ5};/* Set of available
     dual-cube mutations               */
3  X ← N^PC_l;/* Set of production cubes of
     layout l                          */
4  while X ≠ ∅ do
5      m ← False;/* Variable tracking if
         a mutation was applied        */
6      r ← RandomInt(0, | X | −1);
7      c1 ← X[r];
8      X ← X\{c1};
9      r ← RandomFloat(0, 1);
10     if r < μ^PC_Rate then
11         r ← RandomFloat(0, 1);
12         if r < 0.5 then
13             r ← RandomInt(0, |
                 μ^PC_SingleOp | −1);
14             o ← μ^PC_SingleOp[r];
15             o(c1);
16             m ← True;
17         else
18             X2 ← {c2 ∈ X |
                 HaveSharedFloorRange(c1, c2)};/* Set
                 of production cubes on
                 the same floor as c1   */
19             if X2 ≠ ∅ then
20                 r ← RandomInt(0, |
                     μ^PC_DualOp | −1);
21                 o ← μ^PC_DualOp[r];
22                 r ← RandomInt(0, | X2 |
                     −1);
23                 c2 ← X2[r];
24                 X ← X\{c2};
25                 o(c1, c2);
26                 m ← True;
27             end
28         end
29         if m then
30             FixIslands(l);/* Considers
                 all floors affected by
                 mutation, see Algorithm
                 3.5                   */
31         end
32     end
33 end
   /* Repair procedure for bringing
      outside production cubes back
      inside the property bounds       */
34 if FixNotFullyInsideProductionCubes(l)
     then
35     return True;/* Repair was successful
         (or unnecessary)              */
36 end
37 return False;/* Some cubes are still
     outside the layout, this layout is
     marked for removal                */
```

Algorithm 3.4: Mutation procedure for transient elevators of a layout l

```
1  μ^TE_SingleOp ← {μ2};/* Set of available
     single-cube mutations             */
2  μ^TE_DualOp ← {μ5};/* Set of available
     dual-cube mutations               */
3  X ← N^TE_l;
4  while X ≠ ∅ do
5      r ← RandomInt(0, | X | −1);
6      c1 ← X[r];
7      X ← X\{c1};
8      r ← RandomFloat(0, 1);
9      if r < μ^TE_Rate then
10         r ← RandomFloat(0, 1);
11         if r < 0.5 then
12             r ← RandomInt(0, |
                 μ^TE_SingleOp | −1);
13             o ← μ^TE_SingleOp[r];
14             o(c1);
15         else
16             X2 ← {c2 ∈ X | v^Start_c1 =
                 v^Start_c2 ∧ v^End_c1 = v^End_c2};
17             if X2 ≠ ∅ then
18                 r ← RandomInt(0, |
                     μ^TE_DualOp | −1);
19                 o ← μ^TE_DualOp[r];
20                 r ← RandomInt(0, | X2 |
                     −1);
21                 c2 ← X2[r];
22                 X ← X\{c2};
23                 o(c1, c2);
24             end
25         end
26     end
27 end
```

Algorithms 3.5 and 3.6 explain the most important computational steps partaking in said repair procedure. For the placements, the $SnapAddGracefully$ sub-procedure is being used (see Algorithm 3.1). Frequently repairing islands is computationally expensive; however, it is beneficial for almost all objective functions, which is an acceptable tradeoff. The repair procedure for fixing outside cubes $FixNotFullyInsideProductionCubes$, mentioned at the bottom of Algorithm 3.3, is very similar to the island fixing step, and utilizes $SnapAddGracefully$ as well, to minimize layout distortions. It attempts to move cubes that are, partially or fully, outside the facility property back inside the layout, to serve constraint c_4 (see Section 3.3.4). Depending on the number of cubes involved and the property size, the sub-procedure's overall success rate can strongly vary. As this is not a forced procedure, other cubes can block an outside cube from being validly placed back inside the layout. In this case, the solution is marked for removal and will be discarded in the selection step. All offspring solutions of the population are being mutated, thus the overall mutation rate is 1. However, the mutation rates for the production cubes μ_{Rate}^{PC} and transient elevators μ_{Rate}^{TE} are assumed to be 0.4 and 0.25, respectively. This means that on average 40% of production cubes and 25% of elevators of each layout are being mutated.

3.6.3 Selection

The selection stage considers both the generated offspring population and the existing stored archive, which contains the best solutions. The archive is updated depending on the applied evaluation method: A dominance-based or a scalarization-based measure. In the case of the dominance-based evaluation, we follow the SPEA2 specification by Zitzler et al. [60]. Said measure is based on strength values, meaning that it depends on how "strong" a dominator solution is. The strength of a solution is calculated by the number of solutions it dominates. The stronger the dominators of a solution are, the worse its fitness will be. See Section 2.2.2 for a more complete explanation of the SPEA2 evaluation principle.

For a scalarization-based approach, we simply sum up the normalized results of the objective functions. Assuming an optimal normalization of objective values, each objective has an equal impact on a solution's fitness when evaluated via the scalarization measure. The normalization of objectives is outlined in Section 2.2.3. Formally, we can describe the calculation of the fitness measure F of a layout l as follows:

$$F(l) = \sum_{n=1}^{m} f_n^{Normalized}(\vec{x}_l)$$

where $\vec{x}_l \in X$ is the decision vector of a layout l. After the fitness calculation is completed the algorithm copies the layouts from the sorted solutions list to the archive until it is at capacity. The size of the archive $\mid \overline{P_t} \mid$ is defined in the configuration file (see Section 3.7.2).

Algorithm 3.5: *FixIslands* sub-procedure for repairing production cube islands for a layout l and a floor v

$I \leftarrow GetIslands(l, v)$; /* Sub-procedure for identifying production cube islands for a layout l and a floor v */
$i^{Biggest} \leftarrow \max_{i \in I} | i |$;/* Get biggest island */
$I \leftarrow I \backslash i^{Biggest}$;
$PrevConn \leftarrow GetConnectionsCopy(l)$;/* Sub-procedure for copying previous port connections between cubes. This is later used for restoration. */
$N_l^v \leftarrow i^{Biggest}$;/* Remove all cubes from the layout floor that are not part of the biggest island */
$i^{Pivot} \leftarrow i^{Biggest}$;
if $I = \emptyset$ **then**
    return;/* There was only one island; the floor is considered fixed */
end
while $I \neq \emptyset$ **do**
    /* Sub-procedure to determine closest pair of cubes between the pivot island and the set of remaining islands */
    $c^{Pivot}, i^{Movable}, c^{Movable} \leftarrow GetClosestIsland(i^{Pivot}, I)$;
    /* After the following sub-procedure i^{Pivot} will contain more cubes and $i^{Movable}$ will be empty */
    $SnapMergeIslands(i^{Pivot}, c^{Pivot}, c^{Movable}, PrevConn)$; /* See Algorithm 3.6 */
    $I \leftarrow I \backslash i^{Movable}$;
end

3.7 Implementation details

With the internals of the algorithm discussed, we will now examine the input parameters that comprise a problem instance as well as the variables by which the algorithm was configured. Finally, we will elaborate on the programming environment used to create the algorithm.

3.7.1 Input

The algorithm itself is passed a "scenario" folder path where the inputs and a configuration file are stored. The extensive input of the algorithm is described in Table 3.1. Each of the input CSV files addresses exactly one of the following aspects: The property, the production cubes, the elevators, the material flows or the adjacencies. Because we assume elevators to be square, the input for the elevators only accepts an area property rather

Algorithm 3.6: *SnapMergeIslands* sub-procedure for recursively merging a movable island with a passive pivot island i^{Pivot} while considering previous cube-port connections $PrevConn$ and starting with the closest cube pair of the two islands $(c^{Pivot}, c^{Movable})$

```
   /* Prepare for recursive island merge by snapping the
      closest cube from the movable island to the pivot cube
      from the pivot island                                  */
 1 if ¬SnapAdd(c^Pivot, c^Movable) then
   |  /* If snapping is not possible, add the cube to any cube
   |     of the pivot island                                 */
 2 |  if ¬SnapAdd(i^Pivot, c^Movable) then
 3 |  |  SnapAdd(N_l^v, c^Movable);/* If that did not work, add it to
   |  |     any cube of the layout.  Happens almost never.  */
 4 |  end
 5 end
 6 i^Pivot ← i^Pivot ∪ {c^Movable};
 7 i^Movable ← i^Movable \ c^Movable;
 8 Q ← c^Movable;/* Queue for recursively bringing previously
      connected cubes to the pivot island                    */
 9 while Q ≠ ∅ do
10 |  c^Current ← Q[0];/* Here, we assume an array access operator
   |     to function as a "Dequeue" statement for a typical
   |     Queue data structure                                */
   |  /* p denotes the exact ports the cubes were previously
   |     connected over                                      */
11 |  foreach (c^Partner, p) ∈ PrevConn[c^Current] do
12 |  |  if c^Partner ∈ i^Pivot then
13 |  |  |  continue;/* Cube was already restored, skip it  */
14 |  |  end
15 |  |  SnapAddGracefully(c^Current, c^Partner, p, i^Pivot, v, l); /* See
   |  |     Algorithm 3.1                                    */
16 |  |  i^Pivot ← i^Pivot ∪ {c^Movable};
17 |  |  i^Movable ← i^Movable \ c^Movable;
18 |  |  Q ← Q ∪ c^Movable;/* Here, we assume a set union operator
   |  |     to function as an "Enqueue" statement for a
   |  |     typical Queue data structure                     */
19 |  end
20 end
```

than the exact dimensions. The aforementioned configuration file determines certain behavioral aspects of the application, as discussed in the next section.

Variable Name	**Description**
For Property (single input)	
Length	The x-Dimension of the property in meters
Width	The y-Dimension of the property in meters
Floor Count	The number of floors the facility has
For Production Cubes (set input)	
Name	The name of the production cube
Length	The fixed x-Dimension of the production cube in meters
Width	The fixed y-Dimension of the production cube in meters
Floor Number	The index of the floor this production cube resides in
For Elevators (set input)	
Name	The name of the elevator
Area	The area this elevator needs to occupy in m^2. This is used to determine the side lengths ($\lceil\sqrt{Area}\rceil$).
Floor Span	The number of floors this elevator spans across. This needs to be at least 2.
Floor Start	The index of the floor this elevator starts in
Capacity	The amount of material flow this elevator can handle before it is considered fully occupied
For Material Flows (set input)	
Source	The name of the production cube this material flow originates from
Sink	The name of the production cube this material flow flows to
Intensity	The amount of material transported by this material flow
For Adjacencies (set input)	
PC Name 1	The name of the first production cube of this relationship
PC Name 2	The name of the second production cube of this relationship
Goal State	The goal state of the two cubes. This can either be -1, 0 or 1, corresponding to an "undesired", "indifferent" or "desired" adjacency, respectively.

Table 3.1: The input parameters that define the MFFLP

3.7.2 Configuration

A configuration YAML file controls the algorithm. This file contains information on where to find the input files, where to put the results, and how to act in the individual phases. Specifically, the phase input is structured as follows:

- Archive Size: Specifies how many of the best solutions for each iteration the algorithm keeps in memory. It is also equivalent to the number of layouts of the

final output. This is the only set that persists over several iterations. A higher number means longer runtimes but potentially better results.

- Population Size: For our algorithm, the population size defines the number of new solutions the algorithm generates in the crossover phase in each iteration. After the selection stage, the population is discarded. A higher number means longer runtimes but potentially better results.
- Max. Iterations: Contains the number of iterations, i.e. generational cycles, the algorithm will run for. A higher number means longer runtimes but potentially better results.
- SDE Enabled: Specifies if the algorithm should use shift-based density estimation in the Pareto evaluation sub-procedure.
- Evaluation Mode: Specifies how solutions should be evaluated in each iteration. This parameter can either be set to Pareto or Scalarization.
- Scalarization Objective Range Values File: Holds the location of the file containing extreme values of the individual objectives found in previous runs. This is used when the evaluation mode is set to scalarization. When scalarizing the objective values, the algorithm depends on unbiased normalization. For this, we resorted to normalizing objective values with extreme values that have been observed in previous simulation runs. The values provided by this file can be compared to the previously mentioned ideal and worst points of the (Pareto) search space (see Section 2.2.3).

3.7.3 Environment

The algorithm was written in the C# programming language, with the source files containing more than four thousand lines of code. No inherent parallel programming features of C# and the .NET Framework [39] have been utilized. The application was built as an executable console app, without a user interface. The target framework was the .NET Framework 4.8 [38] and the most important NuGet packages were RhinoCommon [5], YamlDotNet [6] and CsvHelper [10].

3.8 Visualization in Rhinoceros 8 & Grasshopper

To visualize the generated layouts, we used version 8 of the Rhinoceros 3D CAD software as well as the integrated visual programming framework Grasshopper. For this, we built an interactive node-based layout presenter in Grasshopper that can be used to select different layouts, view detailed layout reports and highlight specific floors. There are three major areas present in the file: The layout reader, the post processor, and the group holding the preview components (see Figure 3.3, green left-most, yellow center and red right-most area, respectively). Both the layout reader and the post processor are

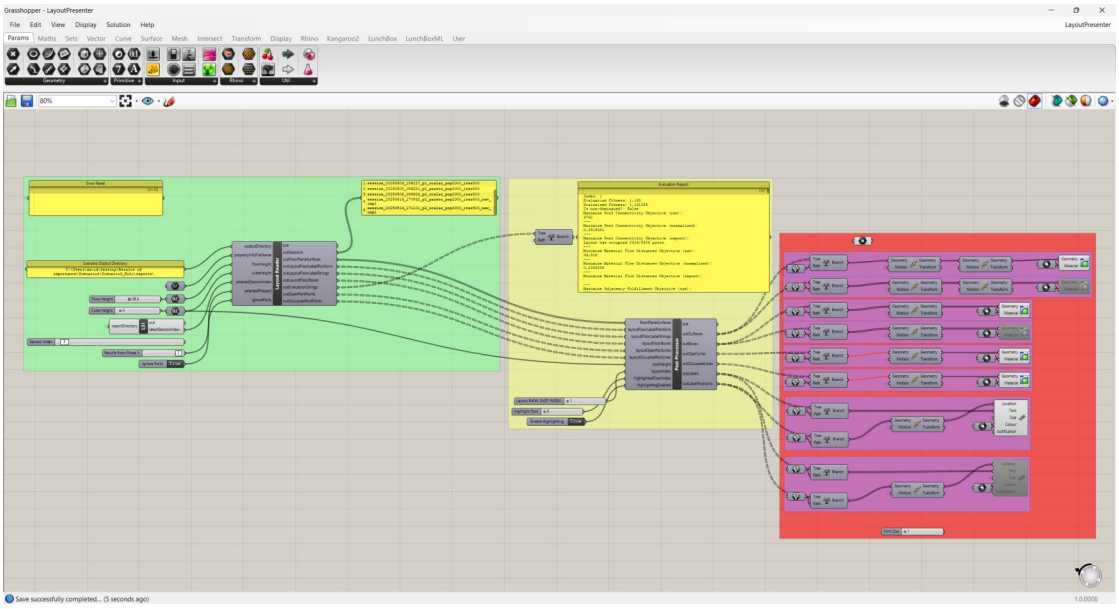
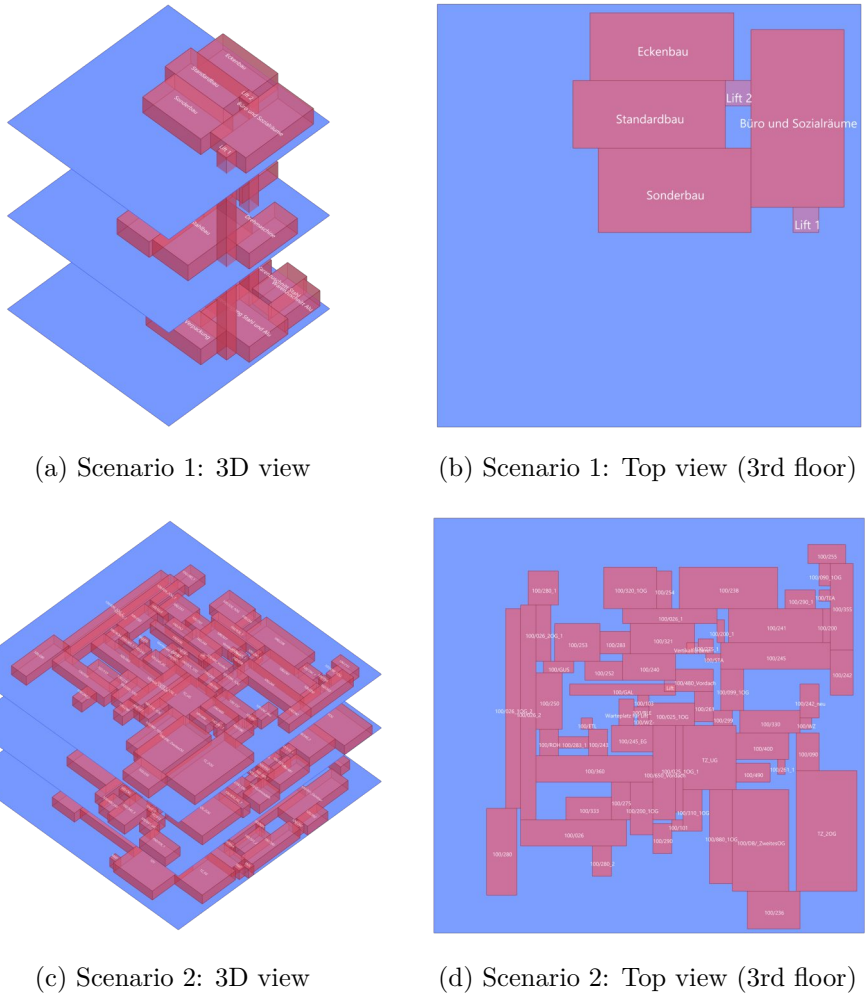


Figure 3.3: The layout presenter in Grasshopper

written in C# and utilize the integrated script component in Grasshopper. The layout reader accepts input parameters that specify the location of the output folder and the property input file. The necessary data is then being read and translated into geometric primitives, labels and summarizing strings. The post processor's role is to interpret and destructure the lists (or "DataTrees") provided by the reader component displaying the currently selected layout and applying floor highlighting if activated. Sliders and toggles provide the capability to control the layout selection and which floors to highlight. Lastly, the preview area uses native Grasshopper components to render a preview of the layout. Figure 3.4 shows example visualizations in Rhinoceros 8 of two solutions for the two scenarios subsequently used in the simulation experiments.



(a) Scenario 1: 3D view

(b) Scenario 1: Top view (3rd floor)

(c) Scenario 2: 3D view

(d) Scenario 2: Top view (3rd floor)

Figure 3.4: Example visualizations in Rhinoceros 8 of two layouts generated via the scenarios used in the simulation experiments (see Section 4.1)

CHAPTER 4

Evaluation

To test the algorithm, we used two distinct problem instances, subsequently referred to as scenarios, provided by civil engineering experts and based on data from ongoing real-world planning projects. Both of these scenarios deal with a three-story property and two full-service bi-directional elevators. What differentiates them is their size: The first scenario comprises 16 production cubes, whereas the second scenario contains 152, or 9.5 times as many, production cubes. Subsequently, the number of material flows and adjacency requirements is also much higher for the second scenario. Table 4.1 shows the complete list of inputs for both instances.

	Scenario 1	Scenario 2
Property Floors	3	3
Property Dimensions	50x50m	114x110m
Per Floor Area	2500m^2	12540m^2
Total Property Area	7500m^2	37620m^2
Production Cubes	16	152
Total Production Cube Area	2059,8m^2	16795,11m^2
Total PC Area / Total Property Area Ratio	0.275	0.446
Elevators	2 (full-service)	2 (full-service)
Material Flows	32	871
Adjacencies	20	714

Table 4.1: The input of the two scenarios. Refer to Section 3.7.1 for a detailed description of the input parameters.

The evaluation chapter can be divided into two main sections: The evaluation of the simulation experiments and the evaluation of the expert study. For the first section, we

ran the algorithm 20 times and recorded a variety of different data to test the utility of our algorithm in two regards: Firstly, we aimed to test the overall performance and the algorithm's ability to generate coherent multi-floor layouts with elevators, fixed-size departments and a novel port-based strategy. Secondly, we wanted to compare the algorithm's performance when using different fitness evaluation methods, specifically scalarization and dominance-based evaluation, in terms of objective satisfaction, runtimes and Pareto-front coverage. Finally, the focus of the second section was to gain supplementary insights on generated solutions via a questionnaire filled out by a professional in the civil engineering field.

4.1 Simulation Experiments

To evaluate the algorithm's performance, both scenarios were run five times for each evaluation measure and each phase, resulting in a total of 20 simulation runs per scenario. The configuration parameters detailed in Table 4.2 were consistently used throughout the simulations.

Parameter	Value
Archive Size	200
Population Size	2000
Max. Iterations	500
SDE Enabled	True

Table 4.2: Configuration Parameters for the Experiments

For each scenario, we wanted to ensure an equal starting point for both fitness evaluation implementations. Thus, we used the overall best layout from the first ten runs, independent of the applied evaluation method, as the seed for the second phase. At the start of phase 2, each run converted this layout by fixing the elevators in place and resolving any collisions with production cubes and the now "solidified" elevators. Of course, as the conversion process is also partly randomized, simulation runs naturally differ in their starting points. Refer to Section 3.1 for an overview of the algorithm design.

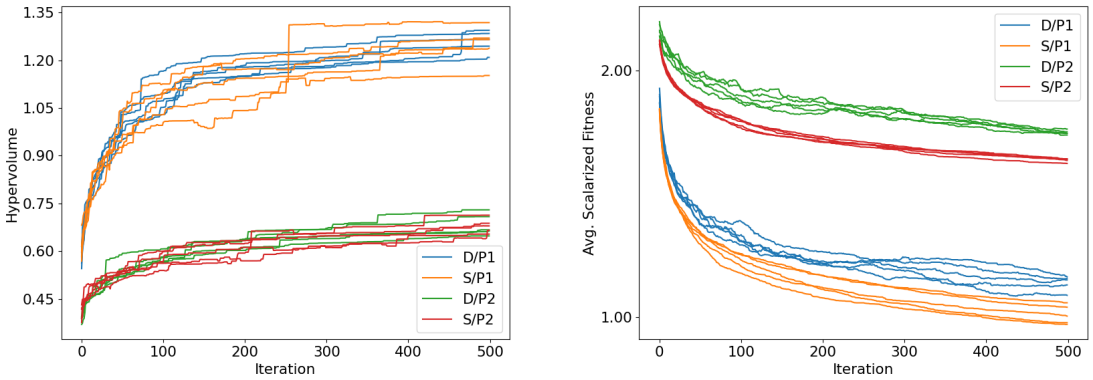
During the optimization processes, a number of different data points were collected. We recorded the average scalarized fitness, the Hypervolume and the objective values of the archive for each iteration. For the arbitrary reference point $r \in \mathbb{R}^m$ of the Hypervolume indicator, we took inspiration from the experimental design by Wang-Sukalia [56] and set it to $(1.1, 1.1, ..., 1.1)$. Furthermore, for the final solution sets, we obtained the Spacing indicator and the objective values. Lastly, the time necessary to complete one iteration was recorded. Refer to Section 2.2.4 and Section 3.4 for a detailed explanation of the mentioned performance measures and objectives. For the experiments, the algorithm was run on a Windows 11 Pro workstation with 256GB of RAM and a 4.2 GHz AMD

Ryzen Threadripper 7960X CPU with 24 cores. Multiple instances of the algorithm were executed in parallel to speed up the collection of results.

The following subsections contain figures that abbreviate the dominance-based and scalarization-based approaches with "D" and "S", respectively, as well as the phases 1 & 2 with "P1" and "P2", respectively.

4.1.1 Scenario 1 Results

Figure 4.1 shows the Hypervolume and average scalarized fitness of the solution archive over time. For both phases, the strategies converged with a similar pace and achieved similar coverage (see Figure 4.1a). The quality of found solutions improves comparably (see Figure 4.1b). Still, scalarization is the superior measure throughout the entire optimization process. However, both approaches yield solutions of similarly good fitness: As shown in Table 4.3, when ranking the final solutions of all runs by their scalarized fitness, both implementations contributed comparably many layouts to the top 10% of solutions.



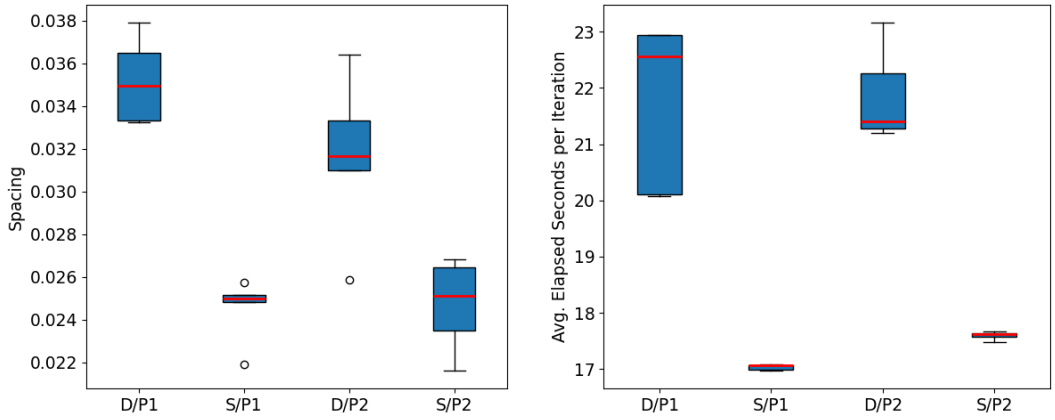
(a) (b)

Figure 4.1: Hypervolume and average scalarized fitness of the solution archive throughout the optimization for the scenario 1 simulation runs. The blue and green lines represent the dominance-based simulation runs, while the orange and red lines denote the scalarization-based simulation runs for the first and second phases, respectively. Each implementation was run 5 times for each phase. A bigger Hypervolume and lower fitness values are better.

	Dominance-based	Scalarization-based
Phase 1	87 (43.5%)	113 (56.5%)
Phase 2	92 (46%)	108 (54%)

Table 4.3: Number of solutions present in the best 10% (n=200) of the unified solution set by evaluation measure and phase for scenario 1

Figure 4.2 shows additional metrics, namely the Spacing of the final solution sets and the average run times. An increased Spacing indicator means that solutions are less uniformly distributed along the Pareto front [7]. This is the case for the dominance-based strategy (see Figure 4.2a) and mirrors the tendency of dominance-based algorithms to explore more niche solution areas. Nevertheless, a higher Spacing value indicates an unevenly distributed solution set and thus an unbalanced (but not necessarily poor) diversity. It is important to note that the Spacing indicator does not provide information about the extent of the covered Pareto front. This means that, although the scalarization-based implementation peformed better from a Spacing perspective, it is unclear whether the extent of the approximated Pareto front is similar to that found by the dominance-based evaluation strategy. However, as mentioned before, the Hypervolume indicates a comparable search space coverage, at least in terms of size.



(a) (b)

Figure 4.2: Spacing of the final solution sets and average elapsed seconds per iteration of the scenario 1 simulation runs, grouped by phases. Lower values are better.

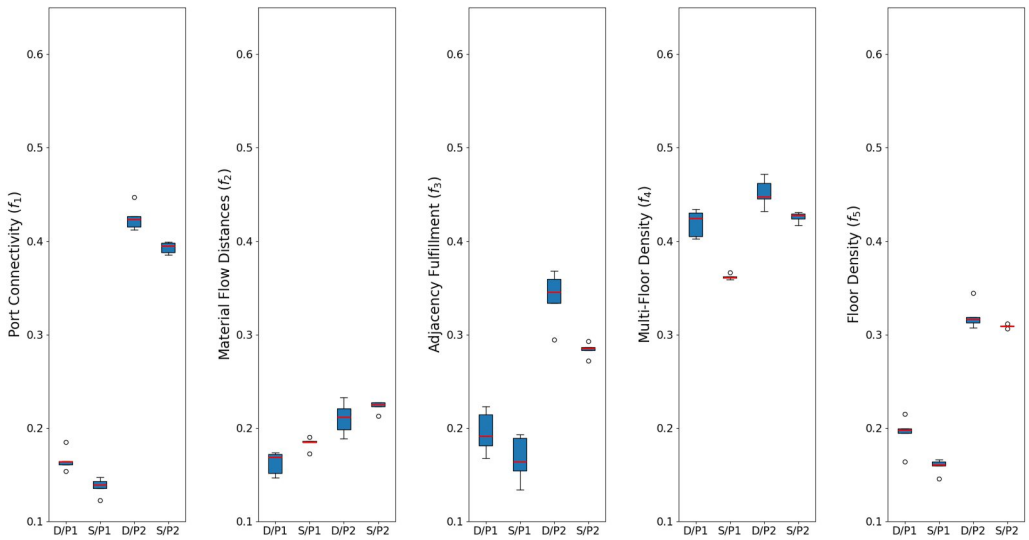
The data collected throughout the simulations clearly indicates that the scalarization strategy is faster for both phases (see Figure 4.2b): In phase 1, an iteration took on average 17 seconds for runs conducted via scalarization evaluation. On the other hand, for dominance-based runs, an iteration took on average more than 4 seconds longer to complete (21.725s). For phase 2, the scalarization- and dominance-based evaluation measures are similarly far apart, with 21.186 and 17.597 seconds, respectively. Noticeably, for both phases, the scalarization runtimes had very little variability ($\sigma = 0.0410s, \sigma = 0.064s$). In contrast, dominance-based evaluation was more unstable regarding iteration runtimes ($\sigma = 1.342s, \sigma = 0.756s$). Generally, the runtime discrepancy can be attributed to the structural differences of the two strategies: Dominance-based evaluation relies on the calculation of strength values and computationally expensive density estimations in each iteration (see Sections 2.2.1 and 2.2.3). Furthermore, for some iterations, additional archive truncation heuristics might also be required (see Section 2.2.2), which is likely the cause for the observed runtime inconsistencies in the dominance-based simulations.

In contrast, the scalarization-based strategy simply sums up the normalized objective values, which is relatively fast and consistent. In total, a full double-phase simulation run with the dominance-based strategy took on average 6 hours and 3 minutes, and with the scalarization evaluation measure, the algorithm ran for 4 hours and 49 minutes.

Some metrics noticeably vary between phases. This can be attributed to the elevators being fixed in the second phase, thus making several objectives more difficult to fulfill. For instance, previously fulfilled adjacencies might not be possible via the same production cube arrangement as in a phase 1 solution. Depending on the positions of the elevators, it is difficult for the algorithm to reach similar objective scores. In the worst case, the elevators were placed in the middle of a large-area production cube. This would likely cause a chain reaction of necessary restructuring steps in the conversion stage of phase 2, steps that likely cause layouts to deteriorate in quality.

Objective Performance

Figure 4.3 shows the values of individual objectives for the final solution sets. The scalarization-based implementation outperforms the dominance-based strategy in port connectivity (f_1), fulfilled adjacency requirements (f_3) and in the two density aspects (f_4, f_5). On the other hand, the data suggest that the dominance-based approach is better at minimizing material handling costs (f_2). Due to the aforementioned reasons, the objective values strongly differ between phases. A fitness evaluation measure that shows better values in phase one, also does so for the second phase.

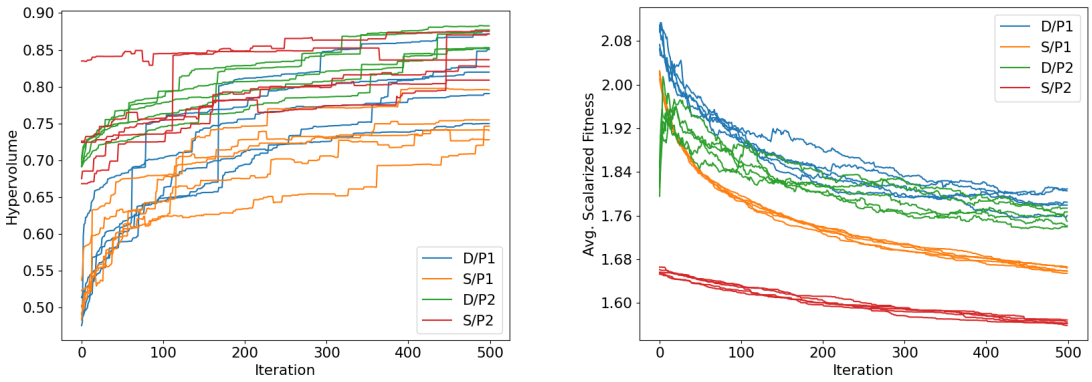


(a) (b) (c) (d) (e)

Figure 4.3: Normalized objective values of the final scenario 1 solution sets, grouped by evaluation strategy and phase. Lower values are better.

4.1.2 Scenario 2 Results

Compared to the first scenario, the Hypervolumes observed for the scenario 2 simulation runs are much more scattered and jittery (see Figure 4.4a). Additionally, for both evaluation strategies, sudden jumps of an increasing Hypervolume appear more frequently. Such jumps are caused by the discovery of new Pareto-optimal solutions, revealing new extremes of objectives. This phenomenon is more likely for bigger instances of our problem, due to the increased search space size. Noticeably, one of the 5 scalarization-based simulation runs for phase 2 starts off with an excellent Hypervolume already. This exceptionally good solution was generated by chance at the beginning of phase 2, where the transient elevators of the seed layout are being converted into solid elevators. With the constraint c_3 now active (see Section 3.3.3), elevators are likely to be considered as invalidly placed by the algorithm due to previously ignored overlaps with the rest of the layout. These constraint violations require collision resolution steps, steps that are partly randomized and normally worsen the layouts; however, not always. Indeed, we confirmed our suspicion of an exceptional solution discovery by chance by examining the archive of the first iteration of said simulation run: The algorithm found exactly one outlier solution with an excellent scalarized fitness of 0.67 and better values for all objectives. In comparison, in all other simulation runs for phase 2, the best three solutions of the first iteration had an average scalarized fitness of around 1.3. Interestingly, this layout was the best solution of all the phase 2 runs for this scenario.



(a) (b)

Figure 4.4: Hypervolume and average scalarized fitness of the solution archive throughout the optimization for the scenario 2 simulation runs. The blue and green lines denote the dominance-based, the orange and red lines show the scalarization-based simulation runs for the first and second phase, respectively. A bigger Hypervolume and lower fitness values are better.

The average scalarized fitness of the archive in both implementations converges very similarly for phase 1, although the scalarization-based method stays better (see Figure 4.4b). Unsurprisingly, the scalarization-based simulation runs perform a lot better when

measured by the scalarized fitness, especially in the second phase: Because the criterion for selecting the best seed solution for phase 2 was the scalarized fitness, the first iteration of the scalarization-based simulation runs considered the converted archive to be an exceptionally good starting point, even if most solutions might have been dominated. In contrast, the dominance-based implementation discarded many of the converted seed layouts for newly generated solutions in the first iteration, as the just converted archive presumably showed poor strength and diversity values, even though the solutions might have had acceptable scalarized fitness. That is why we observe such a strong discrepancy between the two implementations in the average scalarized fitness of the solution archives in the second phase. It is important to note that this situation does not occur for the simulation runs of the first scenario (see Figure 4.1b). We assume that for smaller search spaces, like in scenario 1, the correlation between the two evaluation measures is more intense: There are simply fewer severe errors to be made in a layout that contains fewer cubes, so even if the two modi operandi did not agree in their decisions at the beginning of the second phase, their scalarized fitness stayed on similar levels. The best 10% of the unified solution set contained 52.5% of dominance-based and 47.5% from scalarization-based runs for phase 1. In the second phase, the shares of solutions diverge notably stronger, with 61.5% stemming from the scalarization-based implementation and only 38.5% originating from dominance-based simulations (see Table 4.4).

	Dominance-based	Scalarization-based
Phase 1	105 (52.5%)	95 (47.5%)
Phase 2	77 (38.5%)	123 (61.5%)

Table 4.4: Number of solutions present in the best 10% (n=200) of the unified solution set by evaluation measure for scenario 2

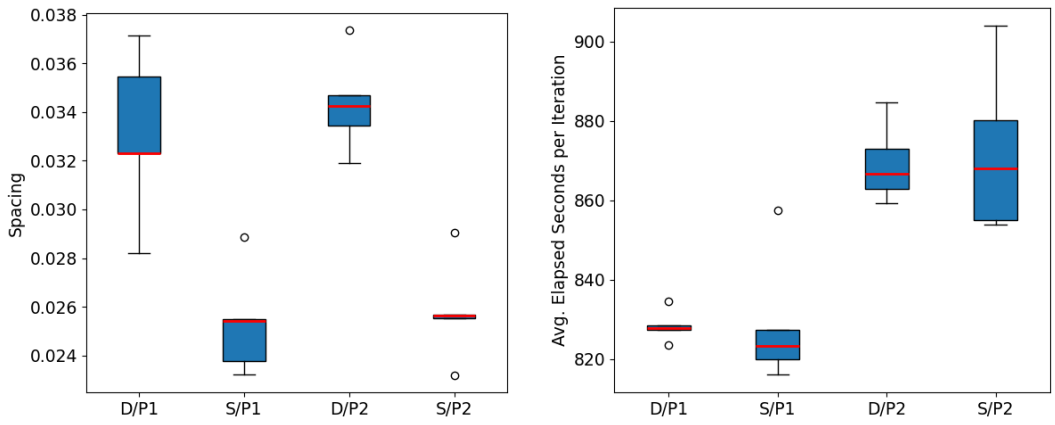
Figure 4.5 shows the Spacing of the final solution sets as well as the time required for an iteration to finish, grouped by evaluation measure and phase. Similarly to the first scenario, for the dominance-based implementation, a higher Spacing value can be observed, indicating that this implementation tends to be more exploratory. The extreme number of participating cubes, material flows and adjacency requirements is reflected in the extreme runtimes of this scenario. For the first phase, one iteration took on average 13 minutes and 48 seconds, and for the second phase on average 14 and a half minutes. Between the two evaluation strategies, a significant runtime discrepancy could no longer be observed. Notably, we observed relatively high standard deviations for the runtimes in the second phase, for both the dominance-based and scalarization-based implementations ($\sigma = 8.95s$, $\sigma = 18.6s$). Especially the fluctuations of the latter strategy were unexpected, compared to the first scenario (see Figure 4.2b), with σ being 2-4% of the average iteration runtime (compared to only 0.3% in scenario 1). Either way, the total runtimes for this scenario can safely be labeled as extensive: Phase 1 and 2 took on average 115 hours and 121 hours, respectively, ultimately resulting in 236 hours or 9-10 days for a double-phase run to complete.

Objective Performance

Figure 4.6 shows the objective values of the final solution sets. For the second scenario, the fulfillment of objectives is more ambivalent when comparing the two evaluation strategies. Generally, there is less discrepancy between the phases in terms of objective satisfaction. For scenario 1, we argued that the conversion of layouts and the accompanying activation of the collision constraint c_3 for the elevators make some objectives more difficult to fulfill. Although this still holds, the share of elevators in the set of all cubes is now much smaller. This means that the potential performance penalty by initial repair procedures is less significant because most of a layout's components will be unaffected by said operations, and thus contribute normally to a solution's fitness. Consequently, we observed fewer differences in the objective performance between the individual phases. For the most part, a strategy that performs better in an objective in phase one also does so in the other. Again, scalarization clearly outperforms the dominance-based implementation in the two density objectives f_4 and f_5 (see Figures 4.6e, 4.6d) as well as in port connectivity (see Figure 4.6a). Solutions returned by the dominance-based algorithm show better scores for material flow distances and fulfill more of the adjacency requirements (see Figure 4.6b, 4.6c). Notably, for exactly these objectives, the scalarization-based simulation runs consistently stagnate very early in the optimization process.

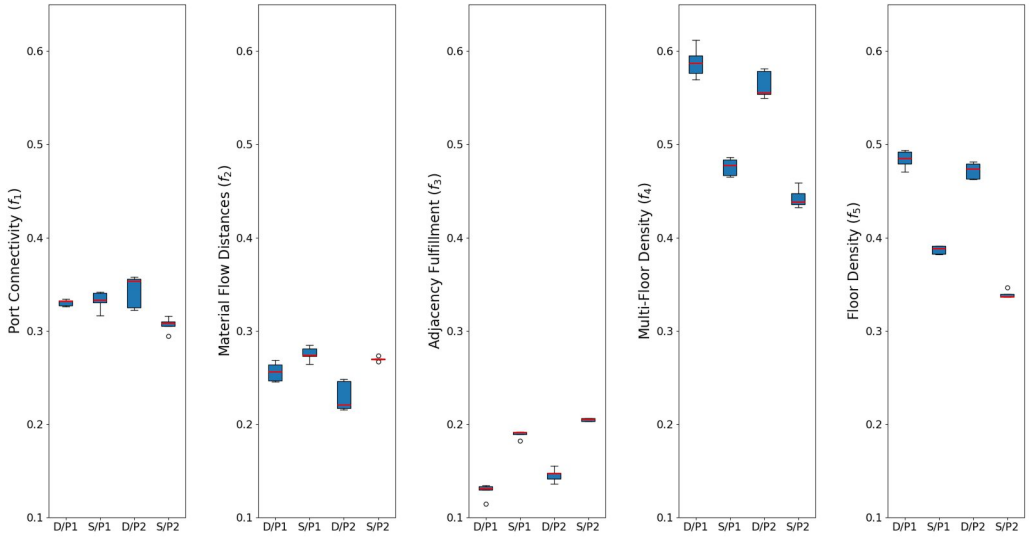
4.2 Expert Study

Apart from the collection of the extensive numerical data, we were interested in obtaining an additional viewpoint from a qualified professional on some of the layouts generated. Thus, we designed an expert study, in which a civil engineering expert evaluated a total of 8 layouts, 4 for each scenario. Each of these subsets was divided into 2 separate



(a) (b)

Figure 4.5: Spacing of the final solution sets and average elapsed seconds per iteration of the scenario 2 simulation runs, grouped by phases. Lower values are better.



(a) (b) (c) (d) (e)

Figure 4.6: Normalized objective values of the final scenario 2 solution sets, grouped by evaluation strategy and phase. Lower values are better.

folders. Every folder contained one solution originating from a scalarization-based and one stemming from a dominance-based simulation run. The expert was unaware which algorithm variant was used to obtain which layout. However, he was given the inputs as well as the final normalized objective values for each design. Furthermore, the questionnaire contained a small introduction explaining the purpose of this thesis and high-level descriptions of the individual objective functions. For each solution, we asked several questions about key aspects, namely material handling costs, clustering, elevator positions and layout density (see Appendix A: Expert Study Questions). Additionally, for each of the sub-folders, we asked the expert to select one of the two layouts that he prefers over the other and provide an explanation for his decision. The questionnaire was answered digitally via Microsoft Word.

4.2.1 Scenario 1 Results

For the first scenario, the expert highlighted that the material handling costs "looked efficient" in most of the layouts, for both of the implementations. Similarly, clustering, i.e. the closeness of related production cubes, was considered very good for all layouts (see an example in Figure 4.7).

His judgments regarding the elevator utilization were more ambivalent. For the calculation of this metric, the expert used a simple greedy routing algorithm, similar to how our procedure decided to assign elevators to material flows (see Section 3.4.2). Some elevators were "significantly less utilized" than others, something that "could be further optimized through a different positioning" (see Figure 4.8). The goal of a balanced elevator utilization

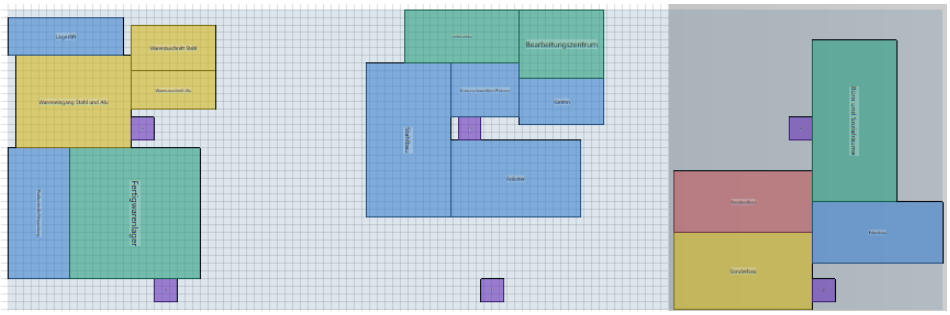
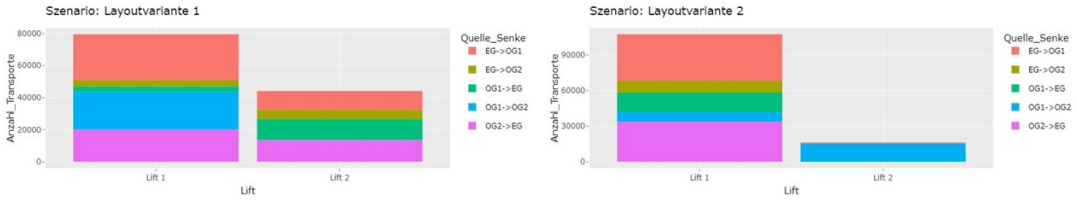


Figure 4.7: The ground, first and second floor of a scenario 1 layout with "very good" clustering, generated via the scalarization-based implementation. Each color corresponds to an individual cluster. It was noted that the position of one elevator (purple, close to lower edge) is too far away from the rest of the cubes. Screenshot with colored departments provided by expert.

was not explicitly modeled in the algorithm; thus, the layouts vary more strongly in this regard. The layout's density was also judged more ambivalently, although, apart from individual production cubes that were seen as misplaced, most of the layouts were considered sufficiently dense ("Position Büro differently to save building space", "Except Lift 1 [the layout density is] very good").



(a) (b)

Figure 4.8: Elevator utilizations of two scenario 1 layouts. On the left-hand side, the layout is balancing the transport load reasonably well between the two elevators. On the right-hand side, the layout hardly utilizes the second elevator. The colored sections denote different combinations of source and target floors. Figures provided by expert.

4.2.2 Scenario 2 Results

In the eyes of the expert, the layouts of the second scenario were more flawed compared to the results of the first input set. Clustering was considered to be poor for most of the layouts ("Cubes of the same type should be closer to each other", "Usually areas of the same type would be more clustered together"). Figures 4.9 and 4.10 show scenario 2 layouts with good and poor clustering, respectively, according to the expert.

It is important to note that the algorithm was unable to group clusters efficiently because the adjacency fulfillment objective (f_3) was designed to fulfill one-to-one adjacency

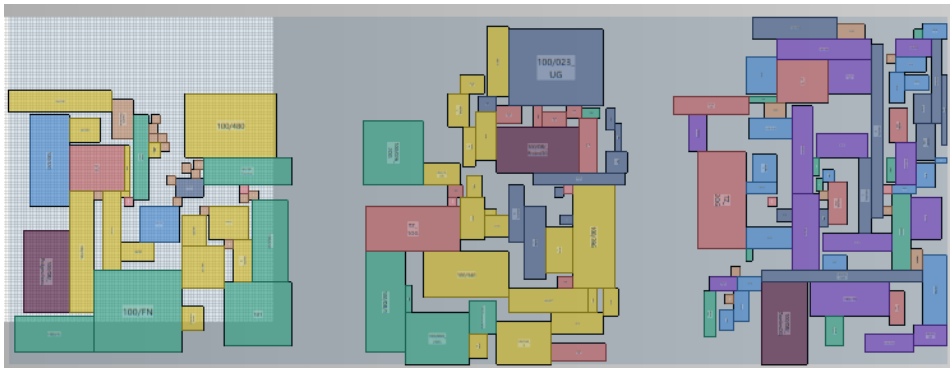


Figure 4.9: The ground, first and second floor of a scenario 2 layout with "good" clustering, generated via the dominance-based implementation. Each color corresponds to an individual cluster. Screenshot with colored departments provided by expert.

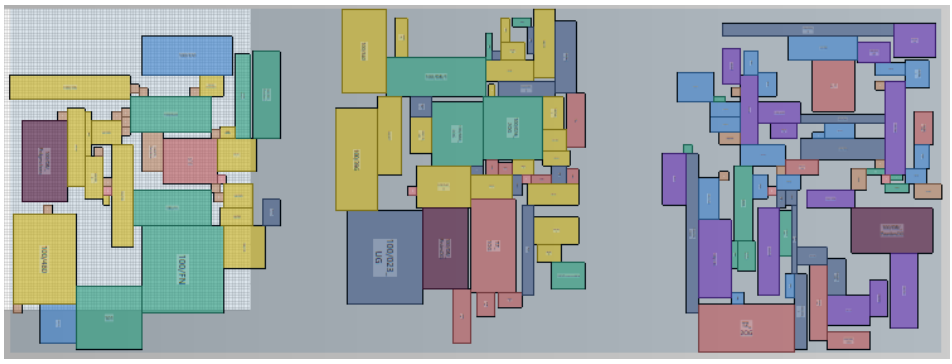


Figure 4.10: The ground, first and second floor of a scenario 2 layout with poor clustering in the second floor (right-hand side), according to the expert. The layout was generated via the scalarization-based implementation. Each color corresponds to an individual cluster. Screenshot with colored departments provided by expert.

requirements. As a cluster of n production cubes was converted into n^2 adjacency requirements in the input pre-processing, the algorithm was not optimizing intra-cluster distances, but binary adjacency relationships. In other words, the objective did not model the (goal of) intra-cluster proximity, which was lacking in many of the layouts, in a suitable manner. Especially for the second scenario, this design flaw resulted in layouts containing many small islands that satisfied some one-to-one adjacency requirements, however, they did not form coherent clusters. This flaw was less of a problem for the first input set, as it contained a much smaller number of such binary requirements. To promote more consistent success rates for intra-cluster proximity, the adjacency objective would need to be reformulated, possibly via a distance-based approach, instead of relying on strict port connectivity (i.e. touches). Most of the layouts showed good density, according to the expert ("The algorithm performed well in densely packing the layout", "[The layout shows] hardly any unnecessary space"). Still, it was noted that some layouts

were "cluttered" (see Figure 4.10, second floor) and some contained unnecessary gaps ("[There are] a lot of empty spaces in the middle of the layout"). Notably, material flows were rated better for designs stemming from dominance-based runs. This observation matches with our collected objective data of the final solution sets of both scenarios (see Sections 4.1.1 and 4.1.2).

4.2.3 Scenario-independent Remarks

For each scenario, the expert favored a solution originating from each implementation exactly once. Thus, the expert study did not reveal any underlying preference for a specific fitness evaluation strategy. The selections were justified with respect to material handling costs, elevator positioning/utilization and density being better. It was stated that the solutions presented were "good starting points to generate more detailed layouts, e.g. with transport paths". Lastly, the expert highlighted that production cubes spanning across multiple floors would be an interesting extension to the procedure.

CHAPTER 5

Discussion

As outlined in Section 1.2, the first research question this thesis aimed to answer was about finding a method for optimizing multi-floor production layouts with fixed department dimensions and dynamically positioned elevators, in the context of multiple objectives. We addressed this question by proposing a specialized algorithm architecture that is based on two phases and distinct positioning anchors, which we referred to as ports. The two-phase design was primarily motivated by the fact that most of the related research did not address elevators in collision constraints. Two main reasons inspired the the port-based solution model. Firstly, discrete relationships of cubes formulated via the exact location of their points of contact allowed for a very efficient "snapping" algorithm: Because the algorithm kept track of both the world coordinates and the state of ports (i.e. open/occupied), it wasted little time on examining impossible cube constellations. Secondly, because we refrained from using an Unequal Area (UA) problem formulation, we needed a consistent method for precisely swapping and copying cubes, for instance, during the crossover phase. Initial input by civil engineering experts indicated that such formulations were too rigid and inflexible. Our reasoning for avoiding the mentioned formulation was based on the fact that most of the related work dealing with UA problem instances generally neglected collisions between elevators and other layout elements.

5.1 Fitness, Objectives & Normalization

The second question we set out to answer was about how a dominance-based evaluation measure influenced the quality of our algorithm's results compared to a (unweighted) scalarization-based approach. Unsurprisingly, the scalarization-based evaluation measure delivered better results when measured by the scalarized fitness. After all, when an implementation uses the criterion that is ultimately used to judge the layouts of all evaluation measures to guide its search, it is expected to claim superiority. Therefore, we need to take a closer look at the results presented in Chapter 4 and interpret them in terms

of the general objective satisfaction. Apart from slightly more efficient material flows, the said fitness evaluation strategy also showed no observable advantageous influence on the objectives. In the second scenario, the differences between the strategies are more profound. Using a dominance-based evaluation resulted in solutions with more fulfilled adjacency requirements and superior material flow distances. On the other hand, scalarization seems to be better at "densely packing the layout", in the words of the expert. Ultimately, both methods returned acceptable results for both input sets. The preferences of the expert lead to a similar conclusion: For each fitness evaluation strategy, he selected two layouts as the superior solutions.

One of the major issues with both evaluation measures was the task of normalizing objective values during the optimization procedure. After all, to succeed in many-objective optimization, it is essential to normalize goals in a way that gives each of them an equal share of influence in the overall evaluation of a solution. However, due to their different ranges, this task is non-trivial. Furthermore, on the surface, dominance-based evaluation seems to promise alleviation of said task, due to its inherent design. Unfortunately, SPEA2 and many related algorithms rely on density estimators that also require some form of objective normalization to provide unbiased results. For dominance-based simulation runs, we followed two approaches proposed by He et al. [21] to calculate ideal and nadir values for every iteration, which were subsequently used for the normalization-reliant density estimations (see Section 2.2.3). As the authors of the said paper rightfully point out, these strategies are rather "naive", as the normalization is based on the algorithm's current front approximation. Naturally, this approximation improves with time, and with it the normalization mechanism becomes more accurate. Still, He et al. [21] showed that there are superior, more informed, normalization strategies for MOEAs. Future work could utilize these more elaborate normalization mechanisms to further improve the overall performance of dominance-based algorithms, such as those used in the domain of MFFLPEs.

More accurate normalization of objectives typically yields high-quality results, especially in scalarization-based optimization. After all, depending on the upper and lower bounds of any given objective, the algorithm "thinks" that there is still much room for improvement, or that it has already found solutions of satisfactory quality. After conducting an in-depth analysis of the collected data, we found that the normalization of some objectives was poorly executed. For both scenarios, the normalization of the material flow distances objective (f_2) resulted in values that were twice as low (i.e. "good") early on in the optimization process compared to the other goals. For the second scenario, the same was true for the adjacency fulfillment goal (f_3). This indicated that the upper bounds of the manually provided ranges for normalizing said goals were far higher than the actual values that the algorithm would typically encounter. This led to the objectives having less impact on the overall fitness, and equally there was lower pressure for the scalarization-based implementation to improve in those aspects. Indeed, Sections 4.1.1 & 4.1.2 show that the scalarization-based procedure underperformed in exactly those poorly normalized objectives, compared to the dominance-based implementation. Considering the number

of repetitions in our experiments, we believe this is unlikely to be a coincidence. Thus, we conducted a small number of test simulations with the input set of the smaller scenario to investigate our assumption further. For these runs, we manually adjusted the problematic normalization ranges to better align with the ranges of the other objectives. The results of the simulations suggested that the algorithm can output more balanced solutions with reasonable fitness, which ultimately supports our view of the negative correlation discussed above.

Due to these reasons, an unbiased comparison between the two evaluation measures based on the data presented seems impractical. The main advantage of dominance-based approaches is their capability to find solutions in non-convex areas of the Pareto front, areas that are usually inaccessible by scalarization-based strategies. However, there are hybrid scalarization-based measures, like the cascaded weighted-sum proposed by Jakob and Blume [26], that are capable of imitating said capability with significantly smaller computation times. Assuming proper normalization, we believe that such fitness evaluation strategies pose considerable potential to effectively optimize MFFLPEs. Still, given the discussed normalization issues, we believe the need for special heuristics or approximation formulas to calculate realistic objective ranges is apparent. Only with such sub-procedures can one guarantee a high degree of stability and consistency in the algorithm's output and performance when using scalarization-based evaluation in the context of varying input sets. That being said, the data presented in the Evaluation chapter (see Figures 4.1b and 4.4b) shows that the scalarization-based simulation runs returned overall fitter solutions for both scenarios. Thus, our results support Hypothesis 1.

5.2 Runtimes

Any optimization procedure, including the problem of generating layouts with satisfactory objective values, has to be evaluated in the context of the time it requires to terminate. After all, as Meller and Bozer [36] put it, "runtimes are required to be reasonable". Compared to related literature, runtimes for the proposed algorithm can be seen as extensive [36], [18], [15]. This is likely caused by a number of reasons, one of which is the high-level formulation of our problem. Table 2.1 in the Literature Review chapter indicates that the most prevalent way of formulating MFFLPs seems to be Mixed Integer Programming (MIP). These formulations tend to be extremely efficient. In contrast, for each cube, our formulation kept track of the ports that were occupied by adjacent cubes. Thus, we implicitly modeled a solution via an undirected graph with cubes making up the nodes and connections signifying a touch via certain ports. Considering this, acceleration techniques previously discovered by researchers, like the ones presented by Goetschalckx and Irohara [18], were not applicable to our implementation. Also, one key reason for the lengthy execution times is the high-level C# programming language we utilized. This programming language has many abstractions, like automated garbage collection [37], which create a certain runtime overhead. For computationally expensive problems like ours, a low-level programming language like C is usually a better choice, as it interacts

more efficiently with the CPU. Furthermore, the program is highly memory-intensive, as each generation is based on the creation of thousands of internal objects. Thus, more granular control over memory, a trait of many low-level programming languages, could potentially be highly beneficial for algorithm runtimes [19]. Lastly, as briefly pointed out in Section 3.7.3, our implementation did not utilize any parallelization features available in the C# programming language. Re-implementing certain sub-procedures to take full advantage of a machine's computing power could thus result in considerable speedups. This is especially true for sub-procedures that test the feasibility of many different cube arrangements, like Algorithm 3.1, for instance. For the smaller scenario, we observed that scalarization-based simulation runs took about 20% less computation time. For the second scenario, there were no considerable differences in average runtimes between the two measures. This suggests that the interrelation between the computation time of the genetic sub-procedures and the size of an input set is likely to be superlinear. In contrast, sub-procedures responsible for evaluating solutions seem less sensitive to the size of scenarios, as they are mostly comprised of simple queries about certain layout properties. Thus, we assume that for the second scenario, runtime differences caused by the choice of the fitness evaluation algorithm were overshadowed by the computational complexity of other algorithm sub-steps. Due to the reasons mentioned above, we reject Hypothesis 2, of a scalarization-based implementation being quicker than a SPEA2+SDE approach.

5.3 Hypervolumes

The observed Hypervolumes of the final solution sets in both scenarios of the two evaluation strategies are very similar (see Chapter 4). For the first scenario, the average Hypervolume of the Pareto front approximations is virtually identical, with very small standard deviations. A similar pattern can be observed for the solution sets of the bigger scenario's second phase. Although runs that utilized the dominance-based evaluation strategy returned front approximations with slightly better Hypervolume values, the difference between the two strategies is considerably subtle. Thus, we reject Hypothesis 3 of a dominance-based SPEA2+SDE approach resulting in solution sets with better Hypervolume. We believe that both evaluation measures show similar Hypervolume values because the objective functions we used were not inherently conflicting, which made the exploration of potential trade-off solutions less of a priority.

5.4 Limitations & Future Work

The limitations of this thesis can broadly be categorized into unaddressed objectives, algorithm assumptions, experiment design decisions and usability concerns. We especially focus on shortcomings identified by the expert, and reflect on assumptions that we believe had the most significant impact on algorithm performance. Additionally, we point out limitations in the input sets we used for the experiments. Lastly, we discuss further steps necessary to improve the procedure's usability.

5.4.1 Unaddressed Objectives

As previously laid out in Section 4.2, one of the main flaws the expert identified with the majority of solutions presented was the unequal utilization of elevators. This issue can be viewed as part of a larger sub-problem in MFFLPEs that addresses the optimal number and floor assignment of elevators with varying capacity. In our case, both of these aspects were fixed variables in the input sets. Intuitively, the balanced utilization of elevators could be modeled as an additional objective. Such an objective could promote layouts with elevators that diverge minimally from a predetermined flow value. This flow value would need to incorporate the material flow intensity for each pair of floors, as well as the available elevators and their respective floor ranges. Lastly, one could introduce more complex sub-procedures that either pre-assign vertical partitions of material flows to elevators with respect to a balanced elevator utilization or calculate elevator assignments online during the optimization process. Either way, this task would be in itself a combinatorial optimization problem that would require special algorithms. Matsuzaki et al. [35], for instance, address said task via an intermediary genetic sub-algorithm in the context of solving UA-MFFLPs.

Similar to the poor elevator utilization, layouts performed poorly in terms of clustering production cubes meaningfully. As stated in Section 4.2.2, due to the design of the adjacency fulfillment objective, the algorithm was given insufficient hints of what makes up a well-clustered solution. In related research, the adjacency goal is not uncommon and is, like in our formulation, frequently based on an "all-or-nothing" philosophy: Either departments are touching or they are not [33, 44]. Notably, we were unable to find related research that explicitly considered the clustering aspect of departments, i.e. the proximity of production cubes belonging to the same group. Arguably, the goals of coherent production cube clusters and fulfilled binary adjacency requirements search for entirely different layout traits, and should thus be formulated separately. There are various ways in which a reasonable clustering of production cubes could be promoted. The most obvious choice is the introduction of another objective that rewards layouts with same-cluster production cubes positioned near each other. Another, possibly more sophisticated, strategy would be to consider clusters of production cubes as their own layouting sub-problem, and at the same time, as production cubes of higher order. Either way, the parallel use of two separate objectives, one for improving layout clustering and another one for promoting the satisfaction of binary adjacency requirements, seems highly feasible.

Besides the two issues discussed above, many more objectives could be considered, as the proposed procedure only deals with the highly abstracted Multi-Floor Facility Layout Problem. In reality, there are many additional factors that must be considered. Prominent examples include the structural properties of buildings, like supporting pillars or load limits, aisles for humans, non-rectangular property shapes, etc. [43, 32]. Future work needs to further investigate more holistic frameworks that propose efficient ways of merging the objectives and requirements posed by various disciplines that comprise the field of multi-floor facility layout design.

5.4.2 Algorithm & Experiment Design

Two of the most impactful constraints were the use of ports and assuming the department dimensions to be fixed. Both of these design choices likely hide good trade-off solutions. The former constraint limited the algorithm's capability to handle more granular department dimensions, regardless of whether these dimensions were set by decision-makers or the algorithm. The latter constraint intensifies the first one and is especially limiting, as decision-makers often do not know the exact department dimensions beforehand. On the contrary, considering the number of papers that work with UA formulations (see Table 2.1), designers often provide ranges of acceptable sizes or areas for departments, which naturally benefits procedural flexibility. Still, our results indicate that even with these assumptions in place, the proposed evolutionary algorithm can achieve reasonable search space coverage and generate layouts with satisfactory objective values.

Another limitation of this thesis was that the scenarios used in the simulations both contained exactly two full-service elevators (see Table 4.1). Thus, we have little information about the algorithm's performance in relation to different elevator properties. For instance, a high number of elevators might negatively influence the algorithm's ability to place them reasonably. Future work should further analyze different specialized heuristics for placing elevators reasonably. For example, Huang et al. [23] use the "center of gravity" of material flows to approximate efficient positions for elevators post-hoc. However, with such an approach, elevators would likely collide with the rest of the layout, a simplification that many related papers made use of (see Table 2.1, "Elevator Has Area?"). Like the cardinality of elevators, the number of floors also remained unchanged between the scenarios. Hence, it is difficult to make statements about the reliability of such a procedure for buildings with a higher floor count. Lastly, additional analysis is necessary to examine how the algorithm's runtime increases in regard to the problem size, to further test the procedure's robustness against large input sets.

5.4.3 Usability

As mentioned in Section 3.7.3, the application was built as a console app, executable via a terminal. Intuitively, a next step for future work is to create a user interface for such an application, possibly tested through user studies, to enable stakeholders to manage different scenarios, algorithm configurations and results more easily. Considering the trend towards cloud-based (optimization) services, a web-based user interface that utilizes, for instance, WebGL [40] for layout visualizations seems highly feasible.

Lastly, none of the implementation variations allowed for specific weights to be set, reflecting the objective preferences of decision-makers. Introducing weights for the scalarization-based evaluation strategy could easily be achieved via a weighted-sum approach. For the SPEA2+SDE implementation, incorporating external preferences is less straightforward, as it is based on dominance relations and not scalar values. One could follow the approach proposed by Friedrich et al. [16], who incorporate weights in the density estimator of the SPEA2. Future work needs to investigate the effectiveness

of different sets of weights to potentially overcome the observed shortcomings of the evaluation measures in the context of MFFLPEs studied in this thesis.

CHAPTER 6

Conclusion

This thesis proposed a novel port-based evolutionary algorithm for solving a multi-floor facility layout problem with elevators, considering five objectives. Our work presented the individual genetic stages and showcased the applied crossover and mutation operators in great detail. We tested two distinct implementations of the algorithm, with one version utilizing a dominance-based approach and the other relying on a scalarization-based evaluation. Furthermore, we consulted a civil engineering expert in an expert study that included generated layouts to gather further insights into the algorithm's ability to satisfy key performance indicators typical of production layouts. According to the collected numerical data and the opinions expressed by the expert, both evaluation strategies generated reasonable layouts. Although our analysis of the final solution sets suggests notable differences in achieved objective values between the different implementations, an inaccurate normalization mechanism chosen for the scalarization-based strategy likely induced an unbalanced optimization of goals. Therefore, we conclude that said measure is highly sensitive to the quality of the utilized objective normalization mechanism and that no evaluation strategy is inherently more suited for the task at hand. As both measures have their own advantages, we see potential in hybrid scalarization-based approaches, provided that they are based on reliable normalization heuristics. For both implementations, runtimes are generally extensive due to a lack of code optimizations and the high-level formulation of the problem. Still, for smaller problem instances, scalarization-based runs terminate noticeably faster. To further investigate the robustness of such a procedure, future work should focus on evaluating algorithms for MFFLPEs with a variety of different multi-floor scenarios. Additionally, we encourage future approaches that include assignable weights for the individual implementations to better align the direction of optimization with the preferences of decision-makers. Lastly, feedback provided by the civil engineering expert revealed that the aspects of balanced elevator utilization as well as coherent production cube clustering have yet to be investigated as standalone objectives in the context of MFFLPEs.

Overview of Generative AI Tools Used

GitHub Copilot (GPT-4o)[1]

- Bug fixing assistance
- Creation of Excel macros
- Creation of Python utilities for the aggregation & analysis of collected data

OpenAI ChatGPT (GPT-4o)[2]

- Assistance for creating correctly annotated pseudo code & mathematical formulas
- Assistance in related work analysis
- Assistance for correct LaTeX formatting

[1] https://docs.github.com/en/copilot/, accessed on 25.08.2025
[2] https://openai.com/index/hello-gpt-4o/, accessed on 25.08.2025

Übersicht verwendeter Hilfsmittel

GitHub Copilot (GPT-4o)[1]

- Hilfe beim Beheben von Bugs
- Erstellung von Excel-Makros
- Erstellung von Python Hilfsprogrammen zur Aggregation und Analyse von gesammelten Daten

OpenAI ChatGPT (GPT-4o)[2]

- Hilfe bei der Erstellung von korrekt annotiertem Pseudeocode und mathematischen Formeln
- Hilfe bei der Analyse verwandter Literatur
- Hilfe bei der korrekten Formatierung von LaTeX

[1] https://docs.github.com/en/copilot/, accessed on 25.08.2025
[2] https://openai.com/index/hello-gpt-4o/, accessed on 25.08.2025

List of Figures

List of Tables

List of Algorithms

Appendix A: Expert Study Questions

A.1 For each of the layouts, please answer the following questions:

1. What could be improved in the layout?
2. What do you think of the Material Handling Costs?
3. How is the cube clustering? Are cubes that would normally be placed closely together near each other?
4. What do you think of the elevator positions?
5. What do you think of the layout density?
6. Please specify the following metrics for each layout:
    - 6.1 Material Handling Cost (in km/year)
    - 6.2 Transport Grid Utilization ("Transportnetzauslastung")
    - 6.3 Land Area Balance ("Flächenbilanz")

A.2 For each pair of layouts A+B in a folder Scenario X/Folder Y, e.g. Scenario 1/Folder 1, pick the layout that you generally preferred. Please explain your preference.

A.3 What is your overall judgment of the generated layouts? Do you think they are usable for the layout design process?

A.4 Do you have any other remarks? (optional)

Bibliography

[1] Abbas Ahmadi and Mohammad Reza Akbari Jokar. An efficient multiple-stage mathematical programming method for advanced single and multi-floor facility layout problems. *Applied Mathematical Modelling*, 40(9):5605–5620, 2016. ISSN 0307-904X. doi: https://doi.org/10.1016/j.apm.2016.01.014. URL https://www.sciencedirect.com/science/article/pii/S0307904X16300026.

[2] G. Aiello, M. Enea, and G. Galante. A multi-objective approach to facility layout problem by genetic search algorithm and electre method. *Robotics and Computer-Integrated Manufacturing*, 22(5):447–455, 2006. ISSN 0736-5845. doi: https://doi.org/10.1016/j.rcim.2005.11.002. URL https://www.sciencedirect.com/science/article/pii/S0736584506000573. 15th International Conference on Flexible Automation and Intelligent Manufacturing.

[3] Gordon C. Armour and Elwood S. Buffa. A heuristic algorithm and simulation approach to relative location of facilities. *Management Science*, 9(2):294–309, 1963. ISSN 00251909, 15265501.

[4] Robert McNeel Associates. Rhinoceros 3d, . URL https://www.rhino3d.com/. Accessed on: 10.02.2025.

[5] Robert McNeel Associates. Nuget: Rhinocommon, . URL https://www.nuget.org/packages/RhinoCommon/. Accessed on: 11.06.2025.

[6] Antoine Aubry. Nuget: Yamldotnet. URL https://www.nuget.org/packages/YamlDotNet/. Accessed on: 11.06.2025.

[7] Charles Audet, Jean Bigeon, Dominique Cartier, Sébastien Le Digabel, and Ludovic Salomon. Performance indicators in multiobjective optimization. *European Journal of Operational Research*, 292(2):397–422, 2021. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2020.11.016. URL https://www.sciencedirect.com/science/article/pii/S0377221720309620.

[8] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella, and Walter J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, Jun 2009. ISSN 1572-9796. doi: 10.1007/s11047-008-9098-4. URL https://doi.org/10.1007/s11047-008-9098-4.

[9] Yongtao Cao, Byran J. Smucker, and Timothy J. Robinson. On using the hypervolume indicator to compare pareto fronts: Applications to multi-criteria optimal experimental design. *Journal of Statistical Planning and Inference*, 160:60–74, 2015. ISSN 0378-3758. doi: https://doi.org/10.1016/j.jspi.2014.12.004.

[10] Josh Close. Nuget: Csvhelper. URL `https://www.nuget.org/packages/CsvHelper/`. Accessed on: 11.06.2025.

[11] Scott Davidson. Grasshopper. URL `https://www.grasshopper3d.com/`. Accessed on: 10.02.2025.

[12] Kalyan Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6:182–197, May 2002. doi: 10.1109/4235.996017.

[13] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. ieee trans syst man cybernetics - part b. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 26:29–41, 02 1996. doi: 10.1109/3477.484436.

[14] Gunter Dueck and Tobias Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90(1):161–175, 1990. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(90)90201-B. URL `https://www.sciencedirect.com/science/article/pii/002199919090201B`.

[15] Forough Enayaty Ahangar, Behrooz Karimi, Negin Ahangar, and Alireza Sheikh-Zadeh. An optimisation approach for multi-floor facility layout design using flexible bays. *International Journal of Industrial and Systems Engineering*, 45:244–270, January 2023. doi: 10.1504/IJISE.2023.134356.

[16] Tobias Friedrich, Trent Kroeger, and Frank Neumann. Weighted preferences in evolutionary multi-objective optimization. *International Journal of Machine Learning and Cybernetics*, 4(2):139–148, March 2012. ISSN 1868-808X. doi: 10.1007/s13042-012-0083-y.

[17] Vincent J.L. Gan, C.L. Wong, K.T. Tse, Jack C.P. Cheng, Irene M.C. Lo, and C.M. Chan. Parametric modelling and evolutionary optimization for cost-optimal and low-carbon design of high-rise reinforced concrete buildings. *Advanced Engineering Informatics*, 42:100962, 2019. ISSN 1474-0346. doi: https://doi.org/10.1016/j.aei.2019.100962.

[18] Marc Goetschalckx and Takashi Irohara. Efficient formulations for the multi-floor facility layout problem with elevators. January 2007.

[19] hanabi1224. C vs c benchmarks. URL https://programming-language-benchmarks.vercel.app/csharp-vs-c/. Accessed on: 06.07.2025.

[20] Jonathan Hathhorn, Esra Sisikoglu, and Mustafa Y. Sir and. A multi-objective mixed-integer programming model for a multi-floor facility layout. *International Journal of Production Research*, 51(14):4223–4239, 2013. doi: 10.1080/00207543.2012.753486. URL https://doi.org/10.1080/00207543.2012.753486.

[21] Linjun He, Hisao Ishibuchi, Anupam Trivedi, Handing Wang, Yang Nan, and Dipti Srinivasan. A survey of normalization methods in multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 25(6):1028–1048, 2021. doi: 10.1109/TEVC.2021.3076514.

[22] Hasan Hosseini-Nasab, Sepideh Fereidouni, Seyyed Mohammad Taghi Fatemi Ghomi, and Mohammad Bagher Fakhrzad. Classification of facility layout problems: a review study. *The International Journal of Advanced Manufacturing Technology*, 94:957–977, 2018.

[23] Hsiang-Hsi Huang, Ming-Der May, Hsiang-Ming Huang, and Yu-Wei Huang. Multiple-floor facilities layout design. In *Proceedings of 2010 IEEE International Conference on Service Operations and Logistics, and Informatics*, pages 165–170, 2010. doi: 10.1109/SOLI.2010.5551588.

[24] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. Modified distance calculation in generational distance and inverted generational distance. In António Gaspar-Cunha, Carlos Henggeler Antunes, and Carlos Coello Coello, editors, *Evolutionary Multi-Criterion Optimization*, pages 110–125, Cham, 2015. Springer International Publishing. ISBN 978-3-319-15892-1.

[25] Niloufar Izadinia, Kourosh Eshghi, and Mohammad Hassan Salmani. A robust model for multi-floor layout problem. *Computers Industrial Engineering*, 78:127–134, 2014. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2014.09.023.

[26] Wilfried Jakob and Christian Blume. Pareto optimization or cascaded weighted sum: A comparison of concepts. *Algorithms*, 7(1):166–185, March 2014. ISSN 1999-4893. doi: 10.3390/a7010166. URL http://dx.doi.org/10.3390/a7010166.

[27] Ibolya Jankovits, Chaomin Luo, Miguel F. Anjos, and Anthony Vannelli. A convex optimisation framework for the unequal-areas facility layout problem. *European Journal of Operational Research*, 214(2):199–215, 2011. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2011.04.013. URL https://www.sciencedirect.com/science/article/pii/S0377221711003560.

[28] Dan Ji, Zeqiang Zhang, Minjie Zhao, Zongxing He, and Zihan Guo. A multi-objective adaptive memetic algorithm and engineering application for a double-floor layout problem with separate human and vehicle transport elevators. *Engineering*

Applications of Artificial Intelligence, 155:111010, 2025. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2025.111010. URL https://www.sciencedirect.com/science/article/pii/S0952197625010103.

[29] Hüseyin Karateke, Ramazan Şahin, and Sadegh Niroomand. A hybrid dantzig-wolfe decomposition algorithm for the multi-floor facility layout problem. *Expert Systems with Applications*, 206:117845, 2022. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2022.117845. URL https://www.sciencedirect.com/science/article/pii/S0957417422011010.

[30] James Kennedy. *Particle Swarm Optimization*, pages 760–766. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_630.

[31] J. Kephart. A biologically inspired immune system for computers. *Artificial Life*, 1994.

[32] Abdullah Konak, Sadan Kulturel-Konak, Bryan A. Norman, and Alice E. Smith. A new mixed integer programming formulation for facility layout design using flexible bays. *Operations Research Letters*, 34(6):660–672, 2006. ISSN 0167-6377. doi: https://doi.org/10.1016/j.orl.2005.09.009.

[33] Kyu-Yeul Lee, Myung-Il Roh, and Hyuk-Su Jeong. An improved genetic algorithm for multi-floor facility layout problems having inner structure walls and passages. *Computers Operations Research*, 32(4):879–899, 2005. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2003.09.004. URL https://www.sciencedirect.com/science/article/pii/S0305054803002752.

[34] Miqing Li and Xiaohui Liu. Shift-based density estimation for pareto-based algorithms in many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 18:348–365, June 2014. doi: 10.1109/TEVC.2013.2262178.

[35] Kenichiro Matsuzaki, Takashi Irohara, and Kazuho Yoshimoto. Heuristic algorithm to solve the multi-floor layout problem with the consideration of elevator utilization. *Computers Industrial Engineering*, 36(2):487–502, 1999. ISSN 0360-8352. doi: https://doi.org/10.1016/S0360-8352(99)00144-8. URL https://www.sciencedirect.com/science/article/pii/S0360835299001448.

[36] Russell D. Meller and Yavuz A. Bozer. Alternative approaches to solve the multi-floor facility layout problem. *Journal of Manufacturing Systems*, 16(3):192–203, 1997. ISSN 0278-6125. doi: https://doi.org/10.1016/S0278-6125(97)88887-5.

[37] Microsoft. Fundamentals of garbage collection, . URL https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals. Accessed on: 06.07.2025.

[38] Microsoft. .net framework, . URL https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview/. Accessed on: 11.06.2025.

[39] Microsoft. .net parallel programming, . URL https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/.

[40] Mozilla. Mdn webgl. URL https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API. Accessed on: 27.07.2025.

[41] Julia Reisinger, Stefan Kugler, Iva Kovacic, and Maximilian Knoll. Parametric optimization and decision support model framework for life cycle cost analysis and life cycle assessment of flexible industrial building structures integrating production planning. *Buildings*, 12, February 2022. doi: 10.3390/buildings12020162.

[42] Julia Reisinger, Xi Wang-Sukalia, Peter Kán, Iva Kovacic, Hannes Kaufmann, Maximilian Knoll, and Maria Antonia Zahlbruckner. Framework for integrated multi-objective optimization of production and industrial building design. In *Proceedings of the 2022 European Conference on Computing in Construction*, volume 3 of *Computing in Construction*, Rhodes, Greece, July 2022. European Council on Computing in Construction. ISBN 978-8-875902-26-1. doi: 10.35490/EC3.2022.223.

[43] Julia Reisinger, Maria Antonia Zahlbruckner, Iva Kovacic, Peter Kán, Xi Wang-Sukalia, and Hannes Kaufmann. Integrated multi-objective evolutionary optimization of production layout scenarios for parametric structural design of flexible industrial buildings. *Journal of Building Engineering*, 46:103766, 2022. ISSN 2352-7102. doi: https://doi.org/10.1016/j.jobe.2021.103766.

[44] Kazi Shah Nawaz Ripon, Kyrre Glette, Kashif Nizam Khan, Mats Hovin, and Jim Torresen. Adaptive variable neighborhood search for solving multi-objective facility layout problems with unequal area facilities. *Swarm and Evolutionary Computation*, 8:1–12, 2013. ISSN 2210-6502. doi: https://doi.org/10.1016/j.swevo.2012.07.003. URL https://www.sciencedirect.com/science/article/pii/S2210650212000442.

[45] Sancho Salcedo-Sanz, Javier Del Ser, Itziar Landa-Torres, Sergio Gil-Lopez, and Antonio Portilla-Figueras. The coral reefs optimization algorithm: A novel metaheuristic for efficiently solving optimization problems. *TheScientificWorldJournal*, 2014:739768, July 2014. doi: 10.1155/2014/739768.

[46] Sancho Salcedo-Sanz, C. Camacho-Gómez, Daniel Molina, and Francisco Herrera. A coral reefs optimization algorithm with substrate layers and local search for large scale global optimization. pages 3574–3581, July 2016. doi: 10.1109/CEC.2016.7744242.

[47] Hanif D. Sherali, Barbara M. P. Fraticelli, and Russell D. Meller. Enhanced model formulations for optimal facility layout. *Operations Research*, 51(4):629–644, 2003. ISSN 0030364X, 15265463.

[48] Surya Prakash Singh and R. Sharma. A review of different approaches to the facility layout problems. *International Journal of Advanced Manufacturing Technology*, 30: 425–433, September 2006. doi: 10.1007/s00170-005-0087-9.

[49] Adam Slowik and Halina Kwasnicka. Evolutionary algorithms and their applications to engineering problems. *Neural Computing and Applications*, 32(16):12363–12379, Aug 2020. ISSN 1433-3058. doi: 10.1007/s00521-020-04832-8. URL https://doi.org/10.1007/s00521-020-04832-8.

[50] Opossum Support. Food4rhino: Opossum. URL https://www.food4rhino.com/en/app/opossum-optimization-solver-surrogate-models. Accessed on: 18.05.2025.

[51] Thomas Tait. Galapagos tutorial. URL https://hopific.com/galapagos-grasshopper-tutorial/. Accessed on: 18.05.2025.

[52] David M. Tate and Alice E. Smith. A genetic approach to the quadratic assignment problem. *Computers Operations Research*, 22(1):73–83, 1995. ISSN 0305-0548. doi: https://doi.org/10.1016/0305-0548(93)E0020-T. Genetic Algorithms.

[53] Berna Ulutas. A modified flexible bay and slicing structure for unequal area facilities. *IFAC Proceedings Volumes*, 45(6):1635–1640, 2012. ISSN 1474-6670. doi: https://doi.org/10.3182/20120523-3-RO-2023.00362. 14th IFAC Symposium on Information Control Problems in Manufacturing.

[54] Robert Vierlinger. Food4rhino: Octopus. URL https://www.food4rhino.com/en/app/octopus. Accessed on: 18.05.2025.

[55] Wallacei. Food4rhino: Wallacei. URL https://www.food4rhino.com/en/app/wallacei. Accessed on: 18.05.2025.

[56] Xi Wang-Sukalia. *Many-objective optimization for maximum flexibility in industrial building design*. Wien, 2022.

[57] Kuan Yew Wong and Komarudin Komarudin. Solving facility layout problems using flexible bay structure representation and ant system algorithm. *Expert Syst. Appl.*, 37:5523–5527, July 2010. doi: 10.1016/j.eswa.2009.12.080.

[58] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999. doi: 10.1109/4235.797969.

[59] Eckart Zitzler. *Evolutionary algorithms for multiobjective optimization: Methods and applications*, volume 63. Shaker Ithaca, 1999.

[60] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. 2001.

[61] Eckart Zitzler, Marco Laumanns, and Stefan Bleuler. A tutorial on evolutionary multiobjective optimization. In Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen, and Vincent T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, pages 3–37, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-642-17144-4.