FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

# Assignment of bachelor's thesis

| | |
|---|---|
| **Title:** | Artificial Intelligence Methods for Interior Design and Furnishing |
| **Student:** | Eliška Svobodová |
| **Supervisor:** | Ing. Mgr. Ladislava Smítková Janků, Ph.D. |
| **Study program:** | Informatics |
| **Branch / specialization:** | Web and Software Engineering, specialization Software Engineering |
| **Department:** | Department of Software Engineering |
| **Validity:** | until the end of summer semester 2021/2022 |

## Instructions

Design, implement and test a system for automatic furnishing a specific interior using artificial intelligence methods based on parameters entered by the user.

1. Perform a search and describe the state-of-the-art.
2. Select one or more AI methods applicable to the problem.
3. Design a suitable representation to describe the elements of the interior equipment and to describe the conditions of their location and interrelationships.
4. Design an algorithm for automatic furnishing. The algorithm must take into account the conditions and parameters specified by the user.
5. Implement a user interface for entering the conditions imposed by the user on the proposed interior equipment and an interface for displaying the final design.

---

*Electronically approved by Ing. Michal Valenta, Ph.D. on 26 January 2021 in Prague.*

Bachelor thesis

# ARTIFICIAL INTELLIGENCE METHODS FOR INTERIOR DESIGN AND FURNISHING

**Eliška Svobodová**

Faculty of Information Technology CTU in Prague
Department of Software Engineering
Supervisor: Ing. Mgr. Ladislava Smítková Janků, Ph.D.
May 13, 2021

Czech Technical University in Prague
Faculty of Information Technology
© 2020 Eliška Svobodová. All rights reserved.
*This thesis is school work as defined by Copyright Act of the Czech Republic.It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).*

Citation of this thesis: Eliška Svobodová. *Artificial Intelligence Methods for Interior Design and Furnishing*. Bachelor thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

# Contents


**Acknowledgements** vii

**Declaration** viii

**Abstrakt** ix

**Summary** x

**List of abbreviations** xi

**1 Introduction** 1

**2 Goals** 3
- 2.1 Definition of the problem . . . 3
- 2.2 Goals for this work . . . 3

**3 Related works** 5
- 3.1 Search-based methods . . . 5
- 3.2 Graph-based methods . . . 5
- 3.3 Data-based methods . . . 6
- 3.4 Selection of the method . . . 6
- 3.5 Chapter summary . . . 7

**4 Description of the used methods** 9
- 4.1 Simulated annealing . . . 9
- 4.2 Genetic Algorithm . . . 11
- 4.3 Greedy search . . . 12

**5 Application of methods** 13
- 5.1 Zones arrangement . . . 13
  - 5.1.1 Representation . . . 14
  - 5.1.2 Cost function . . . 14
- 5.2 Objects arrangement . . . 16
  - 5.2.1 Representation . . . 16
  - 5.2.2 Relationships . . . 16
  - 5.2.3 Cost function . . . 18
  - 5.2.4 GA operators . . . 19
- 5.3 Chapter summary . . . 19

**6 Implementation** 21
- 6.1 User interface . . . 22
  - 6.1.1 Requirements . . . 22
  - 6.1.2 Presentation of results . . . 24
- 6.2 Simulated annealing . . . 26
- 6.3 Genetic algorithm . . . 28

6.4 Dataset . . . 31
6.5 Chapter summary . . . 31

**7 Results** **33**
7.1 Test set . . . 33
7.2 Run times . . . 33
7.3 Experiments . . . 35
7.4 Comparison with our paper . . . 36
7.5 Chapter summary . . . 43

**8 Conclusion** **45**

**9 Contents of attached medium** **49**

# List of Figures


5.1 Illustration of overlap penalty calculation . . . 15
5.2 Example of individual's chromosome . . . 16
5.3 Cost calculation of AroundCenter relationship . . . 18
5.4 Illustration of one point crossover . . . 19

6.1 Overview of the system . . . 21
6.2 The screen for specifying the shape of the room . . . 23
6.3 The screen for selecting room functions . . . 23
6.4 The screen for selecting furniture . . . 24
6.5 The screens showing the result of zone planning and furniture arrangement . . . 24
6.6 Overview of ui for taking user requirements . . . 25
6.7 Simulated annealing for zone planning . . . 26
6.8 Plots of SA hyperparameters tuning . . . 28
6.9 Plots of SA restart types tuning . . . 29
6.10 Genetic algorithm for furniture arrangement . . . 30
6.11 GA reproduction rates . . . 31

7.1 Diversity of the test rooms . . . 34
7.2 SA run times . . . 34
7.3 The histogram of scores . . . 35
7.4 Rooms with perfect scores . . . 37
7.5 Rooms with score 8 . . . 38
7.6 Rooms with score 7 . . . 39
7.7 Rooms with score 6 . . . 40
7.8 Rooms with the worst scores . . . 41
7.9 Comparison of the results with original paper . . . 42


# List of Tables


6.1 Simulated annealing initial temperature evaluation . . . 27
6.2 Initial groups size . . . 29
6.3 Population size rates and number of iterations . . . 30

7.1 The usability of designs . . . 36

# List of Algorithms

1 Simulated annealing . . . . . 9
2 Genetic algorithm . . . . . 11
3 Greedy search . . . . . 12
4 Arrangement of objects in a group . . . . . 17

*I want to thank my supervisor, Ing. Mgr. Ladislava Smítková Janků, Ph.D., for guidance and cooperation.*

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60(1) of the Act.

In Prague on May 13. 2021 .....................................

# Abstrakt

Tato práce přispívá k výzkumu automatizovaného interiérového designu návrhem a implementací nového systému. Proces designu je rozdělen na plánování funkčních zón pomocí simulovaného žíháni a na aranžování nábytku genetickým algoritmem. Výsledky obou algoritmů jsou dotaženy gradientním sestupem. Systém dokáže splnit požadavky od uživatele na tvar místnosti, její funkce a použitý nábytek. Experimenty ukazují schopnost systému navrhnout interiér různě tvarovaných místností s různým výběrem kusů nábytku.



**Klíčová slova** automatizovaný interiérový design, metody umělé inteligence, genetický algoritmus, simulované žíhání, knihovna Kivy, Python, uživatelské rozhraní, požadavky od uživatele



# Abstract

This thesis contributes to automated interior design research by designing and implementing a new system. The design process is divided into the planning of the functional zones using simulated annealing and arranging the furniture with a genetic algorithm. The system can fulfill the user's requirements on the room's shape, functions, and used furniture. The experiments show the ability of the system to design an interior of rooms with varying shapes and selection of furniture.



**Keywords** automated interior design, AI methods, genetic algorithm, simulated annealing, Kivy library, Python, user interface, user requirements

# Summary

## Motivation

It is challenging to design a room to be comfortable and functional, and hiring a professional interior designer is expensive. A system that does this work quickly and for free would be helpful. Also, automatically created layouts can be used in games and virtual reality.

The thesis is a part of a larger project that studies automatic interior design, and other works will build on it.

## Goal of thesis

The goal is to design, implement, and test a system that takes user's requirements and outputs an arrangement of furniture that fulfills them. A suitable representation of interior elements is needed with a way how to capture their spatial relationships. The system has to have a user interface for entering requirements and showing the results.

## Solution procedure

I researched the methods used so far and decided to use methods based on state-space search because they allow the incorporation of user requirements. To reduce the size of the search space, I divided the design process into high-level planning and local furniture arrangement.

Planning is done by positioning functional zones with simulated annealing enhanced by restart technique and finalized by gradient descent. A solution is composed of positions of all zones and evaluated based on the proximity to windows and walls, overlapping, shape, and flow in the room.

After that, objects are placed into these zones using a genetic algorithm. Chromosomes consist of groups of objects that procedurally arrange themselves. GA is optimizing the composition of the groups and their positions. The furniture of each zone is positioned separately. The quality of its arrangement is measured based on overlapping, proximity to the wall, and object relationships.

## Results

I tested the system on 36 rooms with varying shapes, zones, and furniture. Ten runs were performed on each of them. The results were manually evaluated and those unusable marked. 75 % of the designs were usable, and 31 out of 36 rooms achieved an average score over 6.

## Conclusion

I developed a system for designing an interior according to user requirements. The thesis builds on an article that I wrote with my supervisor and that was accepted at IEA/AIE 2021 conference. I will continue working on it in the future.

# List of abbreviations

| | |
|---|---|
| GA | Genetic algorithm |
| SA | Simulated annealing |
| SUS | Stochastic universal sampling |

Chapter 1

# Introduction

Our homes are our refuges where we relax, create and meet with family and friends. The interior design ensures that these places are comfortable, eye-pleasing, and functional. However, it is not easy to select and arrange objects to achieve such an environment. A person can struggle with imagining what the room would look like with a different arrangement or how to fit all the new furniture in. That is why professional interior designers exist, but designer's help is often expensive. So, a system that would take a person's requirements for the room and proposed a design would be helpful.

Furthermore, the rise of games and virtual reality brought a need for many indoor layouts that will complement virtual worlds. Creating them all by hand is time-consuming, and their simple procedural generation produces unconvincing (or even unplayable) results. Additionally, the training of agents with artificial intelligence requires even more layouts. These agents have to orient themselves in our environment and understand our commands (like "put the mug on the coffee table"). They need thousands of 3D layouts for training. So, we have to come up with a way how to generate these environments.

The majority of furniture configurations are unconvincing or even physically impossible (for example, they contain overlapping furniture). For example, a wardrobe should not be in the middle of the room, a chair should be aligned at the desk, and a bedside table 2 m away from the bed is useless. For the computer, these are all possible choices. Therefore, the first big challenge is to explain what possible designs are (what rules they have to follow and what features make them plausible). Then comes the part that professional designers do. They study the design guidelines, survey the client and plan the space. The problem for automation is, they often optimize the "feel of the room", which is highly non-programmable.

This thesis follows up work from the Research Summer on Faculty of Information Technology (CTU). I developed the first version of core algorithms (GA, SA) during this program and wrote a paper [14] on it in cooperation with my supervisor, Ing. Mgr. Ladislava Smítková Janků, Ph.D. The paper was accepted at IEA/AIE 2021 conference. This work further improves the methods. It also accommodates a large dataset of furniture (3D-FUTURE [4], circa 10 000 objects) and separates concrete objects from the general properties of their categories. Another contribution of this thesis is the graphical user interface. The thesis is part of a larger project on the Faculty of Information Technology that studies automatic interior design, and more works will build on it.

The structure of this thesis is following. First, I define the automatic interior design problem and goals for this work. Then, I describe the state of the research in the field and evaluate the methods used so far. I explain the decision-making about what algorithms to use and summarize the most influential papers. In the chapter Description of the used methods, I provide a brief explanation of the selected methods. In the chapter Application of the methods, I propose a

representation of interior elements and relationships between them. I describe how I adapted the chosen methods to automatic interior design. The implementation part of the work provides the details of used technologies, the architecture of the system, and the tuning of algorithms. In the chapter Results, I present the evaluation of the system based on experiments. Finally, the Conclusion contains a discussion on this work with ideas and plans for future work.

# Chapter 2
# Goals

## 2.1 Definition of the problem

The output of the proposed automatic interior design system is an arrangement of objects in a room. It can be either entirely autonomous (which is helpful for virtual environment generation) or accept the user's requirements on the design. Requirements include the shape and features of the room (like windows, doors, and sockets), functions of the room, or furniture that has to be included.

To accomplish this task, we have to define a representation of the room and objects inside it. Furniture often needs to be positioned together to serve its function. So, the system needs a definition of relationships between objects. Another challenge is how to define and quantify a good design to construct an evaluation function.

## 2.2 Goals for this work

The main goal of this thesis is to design, develop and test a system that takes user requirements and generates an indoor layout fulfilling them. The thesis is part of the research on automatic interior design. The goal is to show the usability of selected algorithms, not to build a complete software product.

The theoretical part contains a summary of the research in automated interior design. The goal is to evaluate the advantages and disadvantages of different approaches and select appropriate algorithms for this work.

The practical part is focused on the application of the selected algorithms and their tuning. A representation of interior elements (like furniture and a room) with a way how to capture relationships between them will be designed. The architecture of the system will allow switching the algorithms easily. The system will have a UI for taking the user's requirements and for showing the final layout.

The final goal is to test the ability of the system to generate usable interior designs. This thesis aims at developing an experimental version of a system that will be part of the research on automatic interior design.

Chapter 3

# Related works

This chapter summarizes the state-of-the-art in automatic interior design and describes the most influential papers in detail. I divide approaches into three main categories - i) search-based, ii) graph-based, and iii) data-based.

## 3.1 Search-based methods

Search-based methods precisely define the environment and apply optimization or heuristic methods to find the most suitable positions for all objects. In [18] authors extract spatial relationships from a set of human-designed rooms and use them in a cost function. The furniture is first randomly positioned and then optimized by simulated annealing. The method allows great incorporation of user requirements, but it optimizes only a few object relationships. Authors of [6] and [1] divide a room into cells and probabilistically select a furniture into them. Cells significantly narrow the search space of layouts, but there is always a limited number of cells and their positions. [10] uses swarm intelligence to find a Pareto-optimum of a set of cost functions (evaluating overlapping, aisle width, object relationships, ...). Authors of [8] applied a genetic algorithm to search for the optimal layout. The results were promising, especially for kitchens and living rooms, but slow (several seconds). So, they experimented with greedy cost minimization [7]. These two papers were the main inspiration for the original article [14] and greatly impacted this thesis. Therefore they will be described in more detail.

**Advantages:** These methods can easily incorporate all kinds of user requirements - for example, what furniture to use, aisle width, or varying room shapes. Also, they do not need large datasets (they only occasionally use them for space relationships extraction).

**Disadvantages:** They are slow. Generating a layout usually takes a few seconds because it is always starting the state search from scratch.

## 3.2 Graph-based methods

Graph-based methods focus on spatial relationships between objects, like them being next to or opposite each other. They define the layout by creating a graph with objects as nodes and relationships as edges. The paper [11] represents layouts as And-Or-graph. It extracts the relationships from the SUNCG dataset [13], transforms them into And-Or-graph, and then samples from the distribution given by it. The system finalizes the layout using simulated annealing. [15] divides the process into two phases - planning and instantiation. The system iteratively adds nodes to a spatial graph using neural networks. Then it takes it as a constraint for furniture instantiation that is done by neural-guided search. The results of this method prove that

separating high-level planning and low-level instantiation is beneficial. Authors of [19] focus on accurate extraction and modeling of the object relationships. They use tests for complete spatial randomness to learn a spatial graph, which is more relevant than counting co-occurrences of objects. The system from [5] takes a few object categories from the user and tries to complement them to a full layout. First, it extracts co-occurrences of furniture categories in data and creates spatial graphs for each activity. According to them, complementary objects are selected and arranged, imitating similar layouts from the data.

**Advantages:** They are capable of capturing the (sometimes complicated) relationships between objects.

**Disadvantages:** A spatial graph itself does not uniquely define a layout, so it needs an additional method to finalize the design.

## 3.3 Data-based methods

Data-based methods use a large dataset (like SUNCG [13]) of indoor layouts to train deep neural networks to imitate them. In the paper, [17] authors broke down the design process into positioning activity zones and arranging furniture into them. For the first step, conditional generative adversarial networks are used with an empty room as a condition. Then, a deep network takes the dimensions and orientations of the generated zones and decides what furniture should be in them. A separate network is trained for each activity. The paper [12] suggest a pipeline from deep convolutional generative models. It iteratively adds objects to a given scene. In each iteration, a representation of the input image is extracted and run through a set of networks. They determine the existence, dimensions, and orientation of the new object. Authors of [9] experimented with putting together deep generative models with spatial graphs. They transform the room into a hierarchy (a tree graph), and an encoder is trained to encode it by levels into a vector gradually. The Gaussian distribution approximates these vectors, so when we give a random vector from this distribution to a decoder, it generates a new design. The paper [3] applies deep reinforcement learning to position objects into a room iteratively. A room with possibly other objects is taken as an environment, and the agent moves the new piece of furniture around to find the optimal position. The agent is trying to imitate the given designs through Q-learning.

**Advantages:** They have to be trained only once, and then they can produce authentic results very fast. A user sees a result in a couple of milliseconds.

**Disadvantages:** Because the graph-based methods imitate layouts, they mostly cannot fulfill specific user requirements like functions of the room, concrete furniture, or aisle width. Also, they are highly dependent on the given dataset (a sufficiently large and quality dataset is hard to find, SUNCG that was used in the past is no longer available).

## 3.4 Selection of the method

I divide the design process into phases and use the methods where they are suited the best. First, the room is planned into functional zones, and then, the furniture is placed into them.

I decided to use search-based methods because they allow greater freedom in incorporating user requirements. The cost function of zone arrangements is noisy and with many local minimums, which is the specialty of simulated annealing. I chose a genetic algorithm for furniture placements because the interior design can be described as the composition of individual objects. GA samples the search space and combines parts of good solutions.

### Automated Interior Design Using a Genetic Algorithm

This paper [8] uses a genetic algorithm enhanced with an island model to find the best layout and then optimizes material selection for the final result.

During initialization, furniture pieces are selected from the database based on the importance of the specific room type and randomly placed and rotated. The authors use four islands with 50 individuals each. Then the search starts, in each generation are individuals evaluated with a cost function. 70 % of the individuals continue to the next population, and a crossover creates the rest. Finally, the mutation is applied to 50 % of the new population.

Objects have properties characterizing their ideal position - the probability of standing against a wall, the required space on the sides, or possible parents (other objects they have to be aligned against). A cost function is a weighted sum of terms representing design guidelines - circulation, group relationships, or functional needs.

Crossover is done by interchanging half of the objects between selected individuals. Any individual with overlapping objects is rejected. Mutation provides moves that help individuals achieve believable designs - align with the closest object, snap the object to the closest wall or add objects to a parent.

The authors conducted a perceptual study when testers had to choose between an automatically created layout and a layout designed by an artist. It showed that the system could produce livable interior designs. Kitchen and living room automatic designs were even preferred over the artist's ones.

### Automatic Furniture Arrangement Using Greedy Cost Minimization

This paper [7] follows [8]. It takes the mutation moves from the previous paper and uses them for a greedy algorithm.

Furniture is again randomly positioned in the initialization phase. Then, in each iteration, the layout is evaluated with the cost function, and a set of moves is applied. If the resulting layout has a lower cost, it is accepted as a new solution.

Additionally, to make the design more believable, several small objects are procedurally added as decorations - for example, a lamp is positioned in an empty corner, a coffee mug on a table, or a shelf on an empty wall.

The authors repeated a perceptual study to compare the greedy algorithm with the genetic algorithm. The greedy search was preferred in all types of rooms apart from kitchens and living rooms (types with which the genetic algorithm was the most successful in the previous paper). They assumed that it is given by the nature of the interior design problem. It is unnecessary to search for the global minimum exhaustively. The search space contains multiple local minimums that represent possible layouts. So the authors decided that it is better to present the user with multiple good designs quickly.

## 3.5 Chapter summary

The state-of-the-art research was summarized and divided into three main approaches. The search-based methods encode interiors as states and use a heuristic or an optimization algorithm to find the optimum. They are generally better at incorporating user requirements. Their disadvantage is that they are slow. Graph-based methods build from objects a spatial graph capturing their relationships. They have to be used in conjunction with other methods that find the concrete layout satisfying the given spatial graph. Data-based methods are based on a large dataset of interior designs on which they train a model that then generates more designs. However, they are just imitating the dataset and cannot incorporate user requirements easily.

I divided the design process into high-level planning of the functional zones and low-level furniture arrangement of the furniture inside them. I chose to use simulated annealing and genetic algorithm. Both use a cost function to which I can include penalties for user requirements.

Chapter 4

# Description of the used methods

## 4.1 Simulated annealing

Simulated annealing (SA) is a random-search technique that approximates the global optimum of noisy functions. It got its name according to the annealing process in metallurgy, where a material is heated and then slowly cooled down to achieve better properties. It originates in 1983 in a paper "Optimization by Simulated Annealing" by Kirkpatrick, Gelatt, and Vecchi, who used it for solving the traveling salesman problem. I studied the algorithm in Busetti's overview [2].

**Algorithm 1** Simulated annealing

```
solution ← generate_initial_solution()
T ← initial_temperature()
while not terminate_condition() do
    for i ← 0, num_it do
        new_solution ← neighbor(solution, T)
        if cost(new_solution) < cost(solution) then
            solution ← new_solution
        else if random(0, 1) < e^(−(cost(new_solution)−cost(solution))/T) then
            solution ← new_solution
        end if
    end for
    T ← cool_down(T)
end while
```

The search is done in iterations by generating a neighbor of a current solution and either accepting it as a new solution or rejecting it. The annealing process is simulated with the temperature variable $T$ that controls the probability of accepting a worse solution. The search starts at a high initial temperature and gradually cools down. The algorithm does *num_it* iterations for every temperature value. A better solution is always accepted (with 100 % probability), a worse solution has a P probability of being accepted:

$$P = e^{-\delta f / T}$$

where $\delta f$ is the increase of the cost between the current solution and the new one. It is also dependent on the temperature. In the beginning, the temperature is high, and the algorithm is exploring the search space. Then it is gradually decreased, so better solutions are mostly accepted, which causes exploitation of the best area. This process prevents being stuck in a

local optimum. Busetti, in his overview [2] states that 40% of the run is usually spent on exploitation. The probability is calculated using the difference between a new and a current solution. Therefore, only a little bit worse solutions have a bigger chance of being accepted than the significantly worse ones.

The technique is robust and applicable to many problems. However, because it is so general, one must specify the number of parameters to use it:

- representation of solutions and definition of a neighboring solution
- initial temperature
- energy function
- annealing schedule
- terminating condition

The search space is defined by the representation of solutions and their neighbors. The neighbor() function should introduce small random changes into the given solution. All solutions must be reachable with a sufficiently short path between any two of them - the diameter of the graph should be small.

Initial temperature significantly influences the efficiency of the algorithm. If it is too high, almost no solutions are rejected, and the algorithm "wanders around". If it is too low, the algorithm does not have a chance to explore, and it is likely to get stuck in the local optimum. A suitable initial temperature is the one that accepts the worse solution with an 80 % chance. However, it is often hard to guess such a temperature. A common approach is to generate several random solutions and their neighbors and initialize the temperature according to them.

Solutions are evaluated by the energy function. It has to be computationally fast because it is called in every iteration. The algorithm is independent of its implementation and takes it as a black box.

The annealing schedule describes the behavior of the temperature during the search. When T tends to zero, the probability P has to tend to zero too. There are many cooling schedules, following linear, quadratic, or exponential curves. The one originally used by Kirkpatrick, Gelatt, and Vecchi is the exponential one:

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

where $\alpha$ is a parameter close to 1 (the original paper used 0.9). The choice of the cooling schedule depends heavily on the problem we are trying to solve.

We decrease the temperature every L iterations, which is equal to generating Markov chains of length L. The efficiency of this approach is given by the rate $\Delta T/L$. The length of the Markov chain can be defined by the number of iterations or the number of acceptances, or whichever comes first.

The terminating condition can be the number of temperature decreases or the number of solutions to be generated. Another option is to stop when there is no improvement in an entire Markov chain at one temperature, and the acceptance ratio is below some fixed point.

SA handles constraints in two ways. The first approach is to reject the generated solutions that violate them. It is easy to implement, but if there are too many constraints or they create disjoint search space, we cannot use it. In that case, we have to transform constraints into penalties. The cost of violating constraints increases with the decreasing temperature.

Simulated annealing is based on slowly cooling down the temperature, which means that the longer we wait, the better solution we get. From Busetti [2]: "It has been proved that by carefully controlling the rate of cooling of the temperature, SA can find the global optimum. However, this requires infinite time." That is why special adaptations of the algorithm have been developed.

Sometimes, the algorithm can come across a promising solution, but it bounces away because randomness is part of its decision process. So, the restart technique remembers the best solution so far, and if the algorithm seems stuck, it begins from it. The restart can be random or be triggered by the difference between the current and the best solution, or it can be performed every $n$ iterations if there is no progress.

Fast (FSA) and very fast (VFSA) simulated annealing algorithms were developed to speed up convergence. The FSA proposes different distributions to be used while generating a new solution that allows using a faster cooling schedule. The VFSA (or adaptive simulated annealing) adds an individual approach to all dimensions of the solutions with their respective temperatures. That supports the solution's encoding with each variable from a different distribution.

## 4.2 Genetic Algorithm

A genetic algorithm (GA) is an evolutionary algorithm inspired by natural selection. Solutions are encoded into chromosome-like data structures (individuals) and selectively combined to create a better set of solutions (population). The source of information about GA was [16] for me.

To apply GA to a problem, we have to define encoding for solutions and a cost function evaluating them. The algorithm begins with generating the initial set of solutions - the initial population. That can be done randomly or with some heuristic. Then, in every iteration, individuals are selected based on their fitness[1] into an intermediate population. Individuals with higher fitness have a higher chance of being selected. Solutions in the intermediate population are combined with a crossover operator. Finally, the mutation is applied to a small percent of the new population. This process repeats until a termination condition is met - total number of generations, minimal criteria for solution quality, or maximal computation time.

**Algorithm 2** Genetic algorithm

```
population ← generate_initial_population()
while not terminate_condition() do
    intermediate_pop ← select(population)
    new_pop ← crossover(intermediate_pop)
    population ← mutate(new_pop)
end while
```

The original, canonical genetic algorithm introduced by John Holland works with solutions encoded as fixed-length binary strings. However, since that, other encodings and operators for them were explored. The parts of the chromosome should be as independent of each other as possible because GA is especially good at finding their best combination. Nevertheless, there are techniques how to overcome this limitation.

The selection operator is responsible for selecting individuals for the intermediate population. One individual can be chosen multiple times. There are several selection operators. Tournament selection iteratively chooses $k$ individuals and selects the best one of them. Roulette wheel selection creates a "wheel" with sectors with proportionate size of how good the individual is. Then it takes a pointer, "spins the wheel" and lets it randomly choose a parent. An important feature of all selection operators is selective pressure. If it is high, it means that only the very best individuals are chosen to reproduce - the algorithm is exploiting a promising area. However, that also means that the population is losing its diversity. So, if the selective pressure is already high at the beginning of GA, it leads to premature convergence. If the selective pressure is low, worse individuals can be selected and pass on their genes that can create a new valuable combination.

[1]solutions are evaluated with cost function (a lower value is better), this cost is then transformed into fitness value (higher is better)

A crossover operator is given two individuals (parents) and recombines their genes into new individuals (children). The classical ones randomly choose one or several crossover points in the chromosomes and swap these parts between parents. Although the crossover operator is technically problem-independent, its implementation can significantly impact the efficiency of the algorithm. That is why special operators are developed for problems with parts of the chromosome dependant on each other.

Mutation should introduce small random changes into the solutions to help explore (and prevent premature convergence). For example, flip a bit in the binary chromosome or add a random value to a part of a chromosome.

The advantages of GA include robustness against noisy functions and adaptability to many different kinds of problems. Its primary disadvantage is the compute time. Especially the cost function is called for every individual in every iteration, so it has to be fast. Additionally, its strength is finding the area with a global optimum, but it is inefficient in discovering the precise optimal point.

To adapt GA to varying kinds of problems and improve its performance, several techniques were developed. Elitism always allows the best individuals to carry on to the next population unchanged to ensure that we will not lose the best solutions that we found so far. The island model uses more separate populations that are evolved separately for most of the time, but after each $x$ iterations, migrations occur. During migration is the best individuals from every island moved to another island. That allows exploitation of different areas on different islands and prevents premature convergence. A similar model is the cellular GA that arranges individuals in a grid. During selection, they can select a mate only from their neighborhood. This results in the appearance of several “islands” with different trends in the genome that compete with each other and gradually form larger and larger islands.

## 4.3 Greedy search

Greedy search or Hill climbing is a simple optimization algorithm for finding the exact point of local optimum. It iteratively examines neighbors of the current solution and picks the best one to continue with. The algorithm stops when there is no better solution in the neighborhood.

The significant disadvantage of this algorithm is that it performs only a local search and easily gets stuck on the local optimum. It looks only on the direct neighbors and does not accept a worse solution even when it would lead it to a much better optimum. That is why it is mostly used with another global-search algorithm (for example, simulated annealing or genetic algorithm).

**Algorithm 3** Greedy search

```
solution ← get_initial_solution()
neighbors ← get_neighbors(solution)
best_neighbor ← neighbor_with_lowest_cost(neighbors)
while cost(solution) > cost(best_neighbor) do
    solution ← best_neighbor
    neighbors ← get_neighbors(solution)
    best_neighbor ← neighbor_with_lowest_cost(neighbors)
end while
```

# Chapter 5
# Application of methods

This chapter explains how were methods from chapter 4 applied to the automatic interior design problem. It also contains a description of representation of objects and their relationships.

The goal is to position all selected furniture into the given room so that it does not violate any physical laws (like overlapping of the objects) and that the arrangement is functional (objects are arranged according to their relationships). As well as professional designers, I divided the design process into two parts: high-level functional zones planning and low-level object arrangement. For a human designer, it is useful to first think of what parts of the room will serve to what function. They focus on how the person will move through space (the "flow") and how are the features of the room utilized (for example, light from the window). Then they arrange furniture to support these functions.

This division is especially convenient for the AI designer because it allows it to reduce the search space significantly. Without zones, the search space is composed of all combinations of furniture positions, so it is enormous. On top of that, most of its solutions are unacceptable - objects overlap, or their arrangement towards each other does not make sense. With zones, AI can first plan the space and find the best positions for all functional zones (there are not many of them and have simpler placement requirements). Then it can arrange the furniture only within its zone and mostly ignore the rest of the room.

## 5.1 Zones arrangement

I am using a classic simulated annealing algorithm with an exponential annealing schedule and step decrease enhanced by a restart technique. Details are explained in chapter 6.

The simulated annealing continues until the solution is not sufficiently improving between temperature decreases and an average acceptance ratio is below 30 %. After that, the solution is finalized by a greedy search. All direct neighbors are generated in its every iteration, and the best one is picked as the new solution. A direct neighbor solution has only one of its zones changed, and that zone solution is altered by step of size 1.

A functional zone is a part of the room that serves one of the functions: sleep, work, relaxation, dining, conversation, or storage. It contains the furniture that supports this function. For example, the sleep zone will likely contain a bed and a nightstand.

A zone has a rectangular shape, a list of objects and a function. The function defines the placement requirements of the zone. For the time being, I distinguish only the probability of the zone being by the window. In the future, the algorithm could also consider sockets, pipes, or the proximity of other zones.

### 5.1.1 Representation

In simulated annealing, one solution encodes the arrangement of all zones in the room. It consists of zone solutions that specify the position of individual zones. Zone's position and shape are given by coordinates of its rectangular bounding box inside a room.

To further limit the search space, I am distinguishing between storage zones and other zones. Each storage zone contains only one piece of furniture - the storage object and they have fixed size. They are taken differently because the study of professional designs shows that they are mainly positioned independently from other furniture. Also, they are always positioned at the wall, facing into the room. On the contrary, other zones have variable sizes, can be positioned anywhere in the room, and do not have any orientation.

The next thing to specify is the neighborhood of a solution. A random neighbor is obtained by first randomly choosing zone solutions to change (from one to all of them). Those zone solutions generate their neighbor, which is put to the neighboring solution along with unchanged non-chosen zone solutions. The storage zone's neighbor is created by moving the storage object along the wall in a random direction and distance. Other zone solutions are either moved in a random direction or resized. The current size of the step limits all these changes.

### 5.1.2 Cost function

A cost function measures the quality of an arrangement. The zone is penalized for every violation of physical constraints or design guidelines. The components of the cost function:

**By window penalty** Some zones need more light (for example, work zone) or benefit from the view from the window (like conversation zone). These zones have non-zero probability $p$ of being by the window. The penalty is calculated from the distance of the zone from its nearest window.

$$windows = \sum_{z \in zones} p * distance(z, window) * 2$$

**By wall penalty** Zones are mostly positioned by the walls because then they leave a natural aisle in the middle of the room that allows the best flow. Also, some furniture in the zone requires being placed by a wall. Distances of all zone corners from their nearest walls are measured, and the three smallest ones *d1, d2, d3* are used for the penalty. The third-smallest distance is divided by two, so it brings a slight advantage to the zones in the corners.

$$wall = \sum_{z \in zones} d1 + d2 + (d3/2)$$

However, zones were often positioned from wall to wall (across the whole room) and blocked the pathway. So, I added a condition that if the third smallest distance from the wall is shorter than the aisle width, the zone gets an additional penalty of 100.

**Overlap penalty** Zones cannot overlap because the object arrangement algorithm depends on it. It performs a search only for the furniture in its zone, and if zones overlap, their furniture will likely create a physically impossible design. As shown in the image 5.1 from the original article [14], every zone is expanded by the size of an aisle, and the overlapping area of this shape with other zones is added to the penalty. This motivates zones to leave an aisle between each other and allows the user to move through the room. Storage zones are extended only on their front side because they can be positioned directly next to each other. The non-storage zone ensures the aisle between a storage zone and a non-storage zone.

$$overlapping = sqrt(\sum_{z \in zones} \sum_{other \in (zones \setminus z)} overlap_area(expanded(z), other))$$

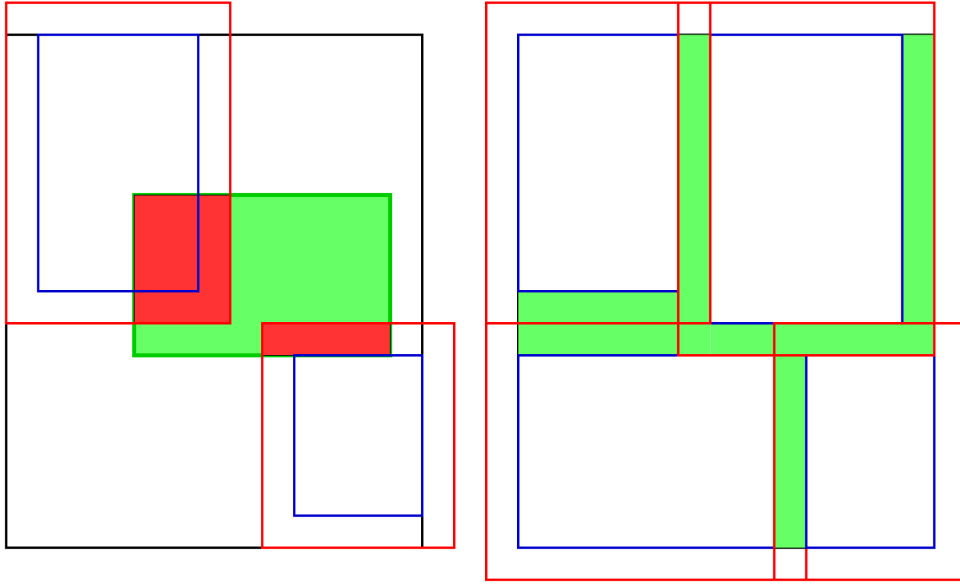


**Figure 5.1** Illustration of overlap penalty calculation. Each zone is expanded by the aisle size and an overlapping area is measured. Overlapping of aisles isn't penalized. This approach ensures free aisles between zones.

**Zone shape** The rectangular shape of the zone should not exceed some ratio of the sides. The algorithm needs to avoid awkwardly long rectangles because it would be likely impossible to fit some of the furniture or arrange it correctly. The penalty is added if one dimension is at least 3-times larger than the other.

$$shape = min\{\sum_{z \in zones} max_dimension(z) - 3 * min_dimension(z), 0\}/4$$

**Free space penalty** This penalty evaluates the usage of the available space. The room area that is not covered by a zone or an aisle is added to the penalty.

$$free_space = sqrt(unused_area(room))$$

**Door space penalty** Doors have to stay accessible. I measure the overlapping of the area in front of the door with the zones.

$$doors = \sum_{z \in zones} overlap_area(door_area, z)$$

**Space ratios** The available space in the room should be divided in such way, that zones with more or larger furniture take up more space. The perfect ratio *perfect* of the zone's furniture areas is calculated as the area of each zone divided by the sum of areas of all zones. Then it is compared with the ratio of the current solution.

$$space_ratio = 100 * abs(perfect - current)$$

With that many penalties, the cost function is very noisy and has many local minimums. Even a slight change in the arrangement of the zones can have a massive impact on the cost of the solution.

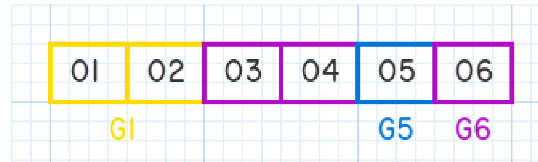



**Figure 5.2** An example of how the individual's chromosome can look like. The chromosome is as long as its number of objects with multiple references to the same group.

## 5.2 Objects arrangement

After functional zones are positioned, furniture is arranged inside them using a genetic algorithm. Each zone is optimized separately as one population. Storage zones are excluded because they contain just one piece of furniture, and their placement was decided during the zone optimization phase.

### 5.2.1 Representation

A fundamental element is an object. It has its category, shape, and orientation. Its category defines its probability of standing against a wall, a space around the object that has to be free and spatial relationships. There are three kinds of relationships - the object can be next to another object, it can be opposite to another object, or several objects can be around a central object.

Objects are put together in groups. When a new group is created, its objects are arranged in relation to each other to fulfill their spatial relationships. The group is then moving as one object, preserving the arrangement of objects inside.

Each individual has its chromosome that represents an arrangement of all furniture in one zone. A chromosome is composed of object groups and has a length same as the number of objects inside the zone. At the beginning of the algorithm, groups are formed randomly. Then, object groups can be joined or divided by crossover or mutation. That means that multiple references to one group are held, one on the position of each of its inner objects. Groups are distinguished by its indexes.

### 5.2.2 Relationships

Every `Object` has an instance of `Category`. The category contains a list of relationships that it can participate in. Every relationship has a list of category names that can be used as the other side of the relationship. A parent is the object that owns the relationship and a child is any object with suitable category.

Each relationship has its arrangement function that places the parent object and its children objects to satisfy the relationship. It also has a cost function that evaluates how the relationship is fulfilled for the parent object.

When a new object group is created, it arranges its objects as is shown in 4. The method `object_with_most_relationships(rest, [type of relationship])` takes a list of objects that weren't yet positioned and a type of relationship. It counts possible children of every object for this relationship and returns the one with the most of the children. That one is taken as a parent. The rest of the objects that have suitable categories are arranged to it as its children. The objects that couldn't be arranged are returned.

**Algorithm 4** Arrangement of objects in a group

$rest \leftarrow all_inner_objects$
$parent, num_rel \leftarrow$ object_with_most_relationships($rest, AroundCenter$)
**if** $num_rel > 0$ **then**
  $rest \leftarrow$ arrange_around_center($rest, parent$)
**end if**
$parent, num_rel \leftarrow$ object_with_most_relationships($rest, OppositeTo$)
**while** $num_rel > 0$ **do**
  $rest \leftarrow$ arrange_opposite_to($rest, parent$)
  $parent, num_rel \leftarrow$ object_with_most_relationships($rest, OppositeTo$)
**end while**
$parent, num_rel \leftarrow$ object_with_most_relationships($rest, NextTo$)
**while** $num_rel > 0$ **do**
  $rest \leftarrow$ arrange_next_to($rest, parent$)
  $parent, num_rel \leftarrow$ object_with_most_relationships($rest, NextTo$)
**end while**

## Around center

The around relationship is asymmetrical. Categories that can be in the center has `AroundCenter` relationship and the ones that can be arranged around some center have `Around` relationship.

The `Around` relationship is the simpler one. The object with `Around` relationship calculates its cost by finding all its possible parents and measuring its distance and orientation to them. The penalty for wrong orientation (from the parent) gives penalty of 100. The smallest sum of distance and penalty for orientation is returned as a cost. `Around`'s arrangement function does nothing, all work is done in `AroundCenter` relationship.

To calculate the cost of `AroundCenter`, the area of central object is divided to four sectors as is shown in 5.3. The sum of intersecting area of all possible children is calculated for every sector. Then the difference of opposite sectors are calculated and the penalty is:

$$penalty = sqrt(min(diff_up_down, diff_left_right))$$

This motivates a visually balanced arrangement.

When a central object's arrangement function is called, all possible children are divided into four groups - north, south, east, west. The opposite groups are trying to have similar areas. The groups on the shorter side of the central object has a smaller chance at assigning some furniture to them. These groups are then positioned on the corresponding sides of the central object with regular spaces, oriented towards its center and a center of the group aligned with the object's center.

## Opposite to

The cost of `OppositeTo` relationship consists of penalty for direction, distance and alignment. The penalty for wrong (not opposite) orientation is 100. Alignment is measured as the difference of `x` or `y` coordinates of centers.

The arrangement of children towards the parent object starts by rotating children to have opposite orientation to the parent. Then they are positioned in a line in front of the parent. The distance between a parent and its child is given by their free spaces on their front side.

## Next to

`NextTo` has two possible alignments - side and center. The side alignment means that back sides of the objects are on one line. A typical example is a nightstand with a bed. The center

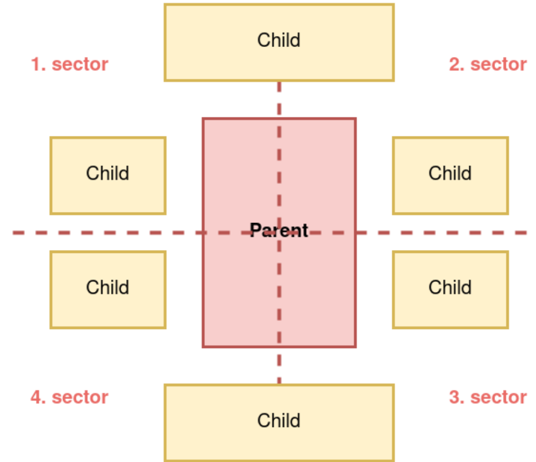



**Figure 5.3** Illustration of the sectors for calculation of the cost of AroundCenter relationship.

alignment arranges object centers next to each other.

The cost is composed of penalties for direction, distance and alignment. Orientations of objects must be the same, otherwise the `NextTo` relationship gets penalty of 100. As the alignment cost is taken the absolute difference between corresponding coordinates of objects.

### 5.2.3 Cost function

Individuals are evaluated with a cost function. It does not take groups into account, only positions of objects. It consists of the following penalties:

**Overhanging room** A solution is penalized for any area of its objects outside of the room.

$$\sum_{o \in all_objects} area(o) - area(intersection(room, o))$$

**Overhanging zone** Objects should not overhang their zone (otherwise, they could overlap with other objects from neighboring zones). They are penalized for their area outside the zone plus for the distance from the zone, if they are completely out.

$$\sum_{o \in all_objects} overhanging_area(o, zone) + distance(o, zone)^2/10$$

**Overlapping** Objects cannot overlap. Also, most objects have some space around them that is supposed to be free too - for example, the space in front of the sofa. So, I expand all objects by their free spaces and penalize any overlapping.

$$\sum_{i \in \{0..len(objects)-2\}} \sum_{j \in \{i+1..len(objects)-1\}} overlap_area(objects[i], objects[j])$$

**By wall** Some categories of objects are more likely to stand against a wall. They can be by it with its back or with one of its sides (the category also defines that). Costs are calculated for all "by wall" sides, and the smallest one is added as a penalty. The cost of one side is given as a sum of distances of side's corners to their nearest wall.

$$\sum_{o \in all_objects} min(\{side_cost | side_cost = distance(corner1, wall) + distance(corner2, wall)\})$$

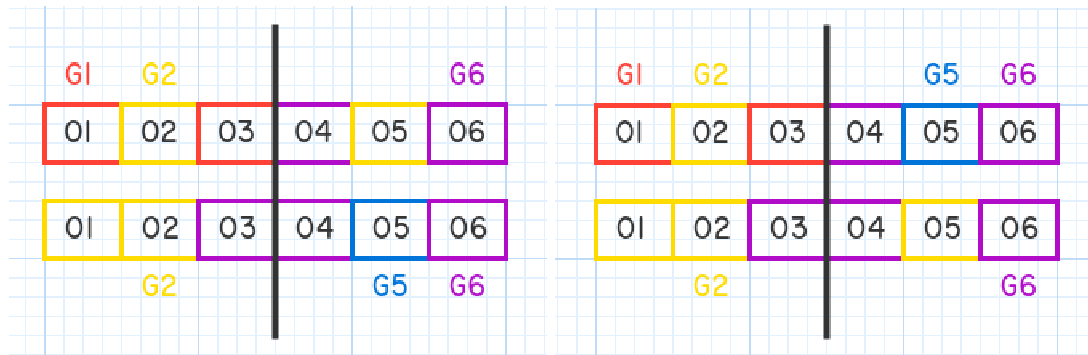



**Figure 5.4** Illustration of one point crossover with chromosomes containing groups of objects. A random point is chosen and parts after it are switched.

**Objects relationships** Every category has defined the most relevant relationships, and every object is trying to fulfill at least one of its relationships. Relationships that cannot be satisfied because there are no objects with suitable categories are skipped.

$$\sum_{oinall_objects} min(\{relationship_costs(o)\})$$

### 5.2.4 GA operators

As a selection operator, I chose stochastic universal sampling (SUS). Individuals are evaluated by the cost function and assigned a fitness according to it. SUS is an analogy to having a roulette wheel with parts proportionally large to the fitness of individuals. We are selecting with $N$ equally spaced pointers, where $N$ is the number of children we want to get.

Crossover is a variation on a classical one-point crossover. A point inside a chromosome is randomly chosen, and objects after it are switched between the chromosomes and form two new children chromosomes. That might divide or join some of the groups, as is shown in figure 5.4. Each part of the chromosome knows to group with which id it belongs. New groups will rearrange themselves according to the relationships inside them.

Mutation is applied to a random part of the population to explore the search space. To ensure that the most promising solutions are not lost, ten best individuals are always copied into the next generation without a change. There are five types of mutation:

- A random group is moved randomly (the shift is chosen, so the group does not leave the zone area).
- A random group is rotated to the left or the right around its center.
- A random group is snapped to the closest wall.
- Two random groups are joined together.
- A random group is divided.

Mutations are designed to apply small changes to the given chromosome that improve the arrangement of the furniture.

## 5.3 Chapter summary

Functional zones mark the parts of a room that serves a specific function (sleep, dinning, storage, conversation, . . . ). Zones are planned with simulated annealing. A solution consists of zone

solutions that encode the positions and shapes of zones in a room. Storage zones are treated differently because they contain only one storage object positioned by a wall facing inside a room. Cost function evaluates zones' distances to windows and walls, their overlapping, and the available space usage.

Furniture is positioned inside zones with a genetic algorithm. Objects inside zones are combined into groups that procedurally arrange themselves. The GA is optimizing the composition and positions of these groups. Furniture has categories that define spatial constraints and relationships with other objects (around, opposite, and next to). Solutions are evaluated based on their overlapping, distance to a wall, and satisfaction of their relationships. I use stochastic universal sampling to select the parents, one-point crossover to combine them, and mutations designed to improve the designs.

# Chapter 6
# Implementation

This chapter contains implementation details of the system and used technologies. Tuning of the genetic algorithm and simulated annealing is described.

The whole program is written in Python. I chose the Kivy library for the user interface because it is multi-platform, GPU accelerated, and provides a user-friendly, easily extensible interface. For geometric computing, I decided to use the Shapely package.

The code is organized into four packages:

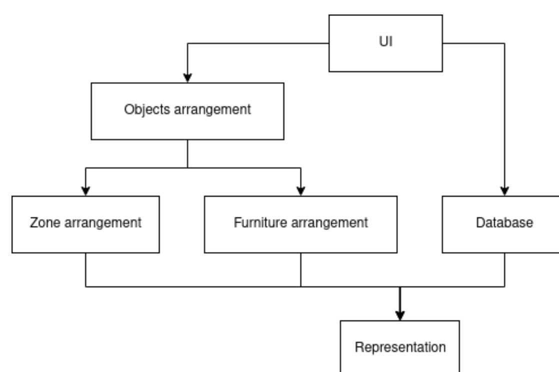
**representations** contain all common classes that represent the room, the furniture, or features of these objects. All other packages use them.

**ui** package consists of kv and Python files that define the user interface.

**arrangement** is the core package containing algorithms for space planning and furniture arrangement.

**database** package provides access to databases of furniture categories, room functions, and furniture.

**Figure 6.1** Overview of the system.




GA's and SA's hyperparameters were tuned on a set of test rooms that consists of 14 L-shaped rooms, 12 rectangular rooms, and 10 rooms of other varying shapes. Rooms contain different combinations of functions and furniture. I measured the cost of the final solution and, for SA, a number of iterations. Because every room is different, I divided the costs (and iterations) by

their sums for that room, grouped data by the parameters that I was testing, and took their mean.

This thesis aims at implementing a system that is a proof-of-concept. It was not developed as a commercial software application. Concretely, that simulated annealing with a genetic algorithm can be used to generate livable interior designs. So, the system as a software product isn't thoroughly tested and doesn't contain a proper input validation. That could be a subject of future work.

## 6.1 User interface

I implemented the user interface with the Kivy library. Kivy divides its code into Python and kivy language (in .kv files). Kivy provides elementary UI elements - widgets, like `Button`, `Label`, or `TextInput`. They are arranged together in layouts. The appearance and hierarchies of widgets are defined in kivy files. Their behavior is added in Python code. Every widget has a canvas, an area where the graphics elements can be drawn (like rectangles, ellipses, ...).

The entry point of the program is the `DesignerApp` class. It extends kivy's `App` class that provides two important methods - `build()` and `run()`. In `build()`, all screens are initialized and given to kivy's `ScreenManager` that switches between them.

There are three screens for taking user's requirements - the shape and features of the room, its functions, and furniture that is supposed to be arranged inside zones. Then comes the design process, which is divided into the planning of the zones and furniture arrangement. First, a screen with optimized zones is shown, and if it is satisfactory, the user can continue on the screen with the arrangement of the furniture.

Every screen contains one top layout that extends the `MainLayout` class that defines the common placement of the UI elements - heading of the screen, the previous and the next button, and a horizontal `BoxLayout` for content.

### 6.1.1 Requirements

The requirements are entered on `RoomScreen`, `FunctionsScreen` and `FurnitureScreen`.

To allow the user to input requirements in arbitrary order and freely switch between requirements screens, I created a central class `RoomBuilder`. It gathers all features that the user enters and notifies other parts of the requirements UI about the changes (the Observer pattern). It has its inner instance of the `Room` and provides an interface for the UI elements to modify it.

#### Room shape and features

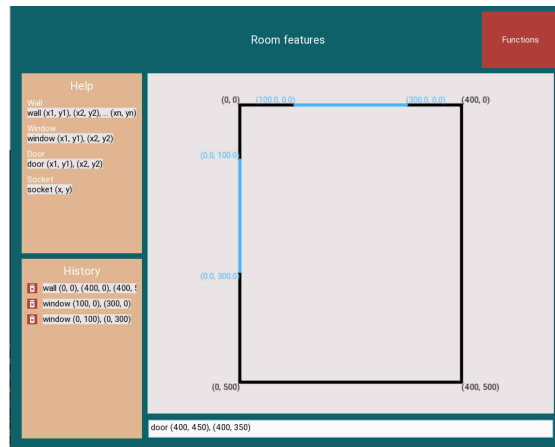
The user specifies the shape and features of the room with a command line. The progress is shown on the canvas above the command line. The screen for defining room shape and features contains a help box with all available commands, a history box with commands executed so far, a canvas with visualization of the room and the command line.

The state is held in `CanvasState` class that remembers the command history, draws on canvas (executes commands), and notifies about changes (the Observer pattern). It keeps track of the center of the drawing, so the visualization of the room is always in the center of the canvas. When the user writes a command in the command line and hits enter, `CommandLine` creates the new command with `CommandFactory` and adds it to the `CanvasState`. `CanvasState` recalculates the center, updates its history, redraws the room, and notifies its subscribers about a change. `CanvasState` saves the inputted shape to `RoomBuilder` when the screen is switched to the next one.

Because this is the first version of the system, there is no input validation yet. It needs to be checked whether the walls aren't skewed (arrangement of the storage zones assumes that) and

room features are placed in possible places.

**Figure 6.2** The screen for specifying the shape of the room and its features (like windows, doors and sockets).

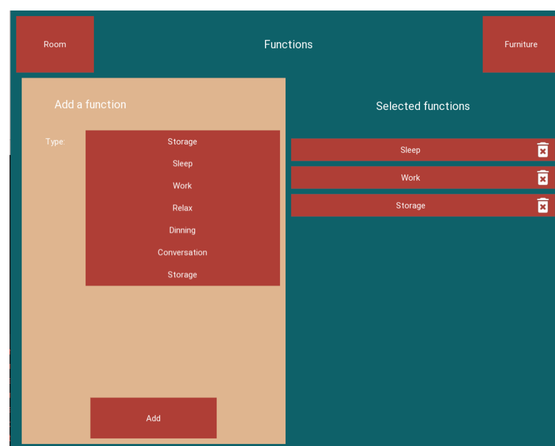



## Room functions

The functions screen consists of two columns - the left one for selecting the new function and the right one showing the selected functions. The user chooses from a drop-down containing all available functions that are obtained from the `FunctionsDatabase`.

When the “Add” button is clicked, the function is handed over to the `RoomBuilder` that transforms it into a new zone and adds it to the room. Then the `RoomBuilder` notifies its subscribers about the change. The layout showing the selected functions is among them and adds the new function to its list.

**Figure 6.3** The screen for selecting room functions.

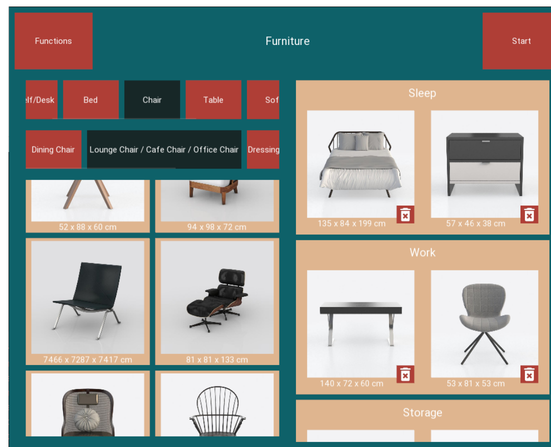



## Furniture

In the furniture screen, the user can choose a super-category and then one of its categories. The list of available furniture from this category appears. When one of them is clicked, an image of it appears by the cursor. The user can put it to any of the zones on the right. That is done

by `SelectEvent`. When a piece of furniture is clicked, it sets a time event that moves with the image and stores the furniture piece. When `ZoneBox` is clicked, it cancels the event and saves the object to the corresponding zone.

**Figure 6.4** The screen for selecting furniture that will be arranges inside zones.




The furniture comes from the 3D-FUTURE database [4] which contains nearly 10 000 models. So, I use `RecycleView` for the list of available furniture that prepares only those furniture cards that are currently visible. That prevents performance degradation and long downloading of the furniture from the remote server. Images are also cached to reduce network load.

## 6.1.2 Presentation of results

After requirements are entered, the user can move to the arranging phase by clicking "Start" button on the furniture screen. The resulting room is got from the `RoomBuilder` and, with the help of `ScreenManager`, given to the zone planning screen. Entering the zones screen triggers the zones arrangement algorithm. When it finishes, the plan of the space is shown. By clicking "Continue", the plan is given (again with the `ScreenManager`) to the objects screen, and the arrangement of the furniture begins. When it is done, the result is drawn to the canvas.

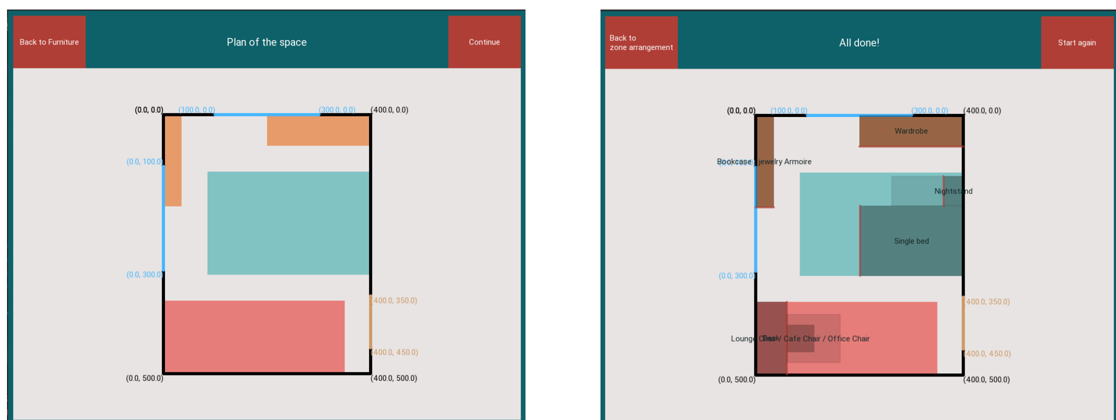



**Figure 6.5** The screens showing the result of zone planning and furniture arrangement.

**Figure 6.6** Overview of ui for taking user requirements.

DesignerApp(App)
+ build()

RoomScreen(Screen)
+ room_builder: RoomBuilder

FunctionsScreen(Screen)
+ room_builder: RoomBuilder

FurnitureScreen(Screen)
+ room_builder: RoomBuilder

RoomEditorLayout(MainLayout)
+ room_builder: RoomBuilder
+ canvas_state: CanvasState

FunctionsSelectLayout(MainLayout)

FurnitureSelectLayout(MainLayout)
+ room_builder: RoomBuilder

HelpHistoryBox

CommandLine(TextInput)
+ canvas_state: CanvasState
+ execute():None

AddFunction
+ function: str
+ room_builder: RoomBuilder
+ add_function(): None

SelectedFunctions
+ room_builder: RoomBuilder
+ method(type): type

FurnitureCollectionLayout

SelectedFurnitureLayout
+ select_event: SelectEvent
+ room_builder: RoomBuilder

AvailableFurniture(RecycleView)

CommandFactory
+ available_commands: List[Command]
+ create(str): Command

<<Interface>>
CanvasChangeSubscriber
+ add(Command): None
+ remove(Command): None

FunctionsDatabase
+ get_all()

FurnitureCard
+ furniture: Furniture
+ select_event: SelectEvent
+ on_furniture_button_pressed(Button): None

ZoneBox
+ select_event: SelectEvent
+ on_press(): none

<<Interface>>
Command
+ execute(Canvas): None
+ save(RoomBuilder): None
+ command_name(): str
+ get_help(): str
+ points(): List[Tuple[int, int]]

<<Interface>>
CanvasChangePublisher
+ attach(CanvasChangeSubscriber): None
+ detach(CanvasChangeSubscriber):None
+ notify_add(Command): None
+ notify_remove(Command): None

RoomBuilder
+ room: Room
+ get_result(): Room
+ reset(): None
+ reset_room_features(): None
+ add_function(str): None
+ remove_function(Zone): None

FurnitureDatabase
+ get_super_categories()
+ get_categories(SuperCategoryName)
+ get_furniture_from_category(str)

SelectEvent
- event: ClockEvent
- image: Image
- furniture: Furniture
+ set_event(Furniture): None
+ cancel_event(): None

CanvasState
+ commands: List[Command]
+ center: Tuple[int, int]
+ add_command(Command):None
+ remove_command(Command): None
+ save(RoomBuilder)

<<Interface>>
RoomZoneChangePublisher
+ attach(RoomZoneChangeSubscriber): None
+ detach(RoomZoneChangeSubscriber): None
+ notify_add(Zone): None
+ notify_remove(Zone): None

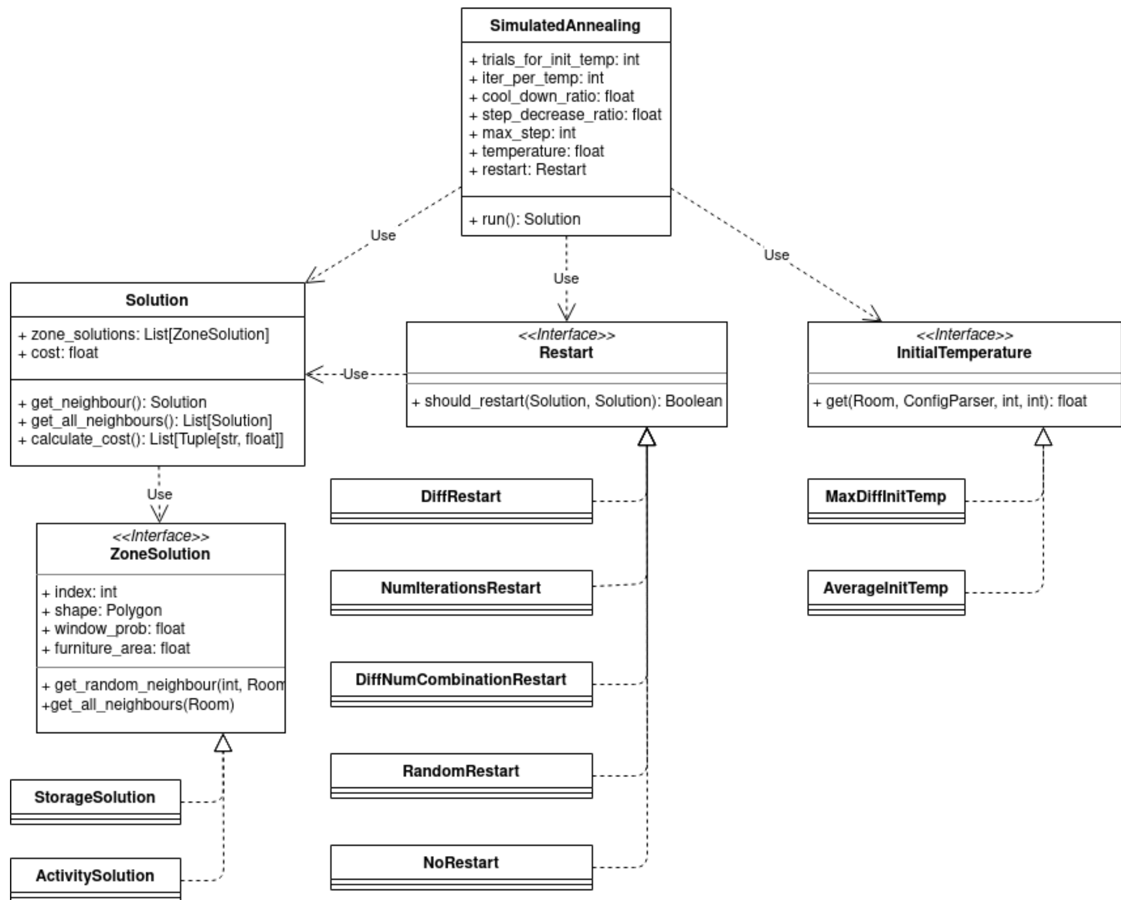
<<Interface>>
RoomZoneChangeSubscriber
+ add(Zone): None
+ remove(Zone): None

## 6.2 Simulated annealing

Simulated annealing is a general technique, so we have to specify many parameters to adapt it to the problem. Their choice has an enormous influence on the efficiency and result quality of the algorithm.

The simulated annealing algorithm is implemented as a class that has to be instantiated for a given room. Then, the `run()` method has to be called to start the computation. The constructor of `SimulatedAnnealing` class takes `restart` and `init_temp` parameters. Those are interfaces (in Python implemented as abstract classes) that allow easy switch of algorithm components.

**Figure 6.7** Simulated annealing for zone planning




## Initial temperature

The first parameter is the initial temperature. The ideal initial temperature is the one that averagely accepts a worse solution with 80 % probability. The problem is that zone arrangement is always performed on a different room with different zones. So, the average value of the cost function in individual cases varies extremely.

I implemented two variants of the initial temperature calculation. Both take 200 random solutions, generate their neighbors with the initial max step, and measure the absolute cost difference between them. Then, the first one `MaxDiff` finds the largest difference between them

**Table 6.1** Simulated annealing initial temperature evaluation

| Init type | Parameter | Number of iterations | Final cost | Score |
|---|---|---|---|---|
| MaxDiff | 0.8 | 0.227414 | 0.262219 | 0.489633 |
| MaxDiff | 1 | 0.249747 | 0.244990 | 0.494737 |
| Average | - | 0.255041 | 0.246064 | 0.501105 |
| MaxDiff | 1.2 | 0.267798 | 0.246727 | 0.514525 |

*max_diff* and returns:

$$T = c * max_diff$$

where $c$ is a constant. I experimented with $c = \{0.8, 1, 1.2\}$

The second one `Average` calculates average difference from all trials *avg_diff* and returns the new initial temperature:

$$T = -\frac{avg_diff}{ln(0.8)}$$

Every method was run five times for each room. The results are shown in table 6.1, the score is calculated as the sum of number of iterations and final cost columns. Even though `MaxDiff` with parameter 0.8 achieved the best score, it had the worst final cost. So I use the second-best result - `MaxDiff` with parameter 1.

## Hyperparameters

To achieve the best results possible, I ran the simulated annealing algorithm with different combinations of its hyperparameters, collected performance data, and evaluated them. Simulated annealing relies on the following hyperparameters:

**Iterations per temperature** For every temperature, $p$ iterations of generating and evaluating a neighbor are run. They give the algorithm time to explore and find the best area before decreasing the temperature and reducing the space that is examined.

$$p = 30, 100, 200$$

**Cool down ratio** The temperature is decreased every time after a fixed number of iterations according to an exponential annealing schedule: $T_{k+1} = p * T_k$.

$$p = 0.8, 0.9, 0.95$$

**Step decrease ratio** The size of step is limiting how different a solution's neighbour might be. I set the initial step to $min(room_dimensions)/2$. Then is step decrease the same as at the temperature: $S_{k+1} = p * S_k$.

$$p = 0.8, 0.9, 0.95$$

Every combination of the hyperparameters was run on all rooms. I took only results with final cost better than was its median. In figure 6.8 are results sorted by the final cost (left) and by the sum of cost and number of iterations. The final cost is almost the same, but the number of iterations (and with it, the run-time of the algorithm) varies. Therefore, it is more advantageous to take the parameters with the lowest number of iterations which is 50 iterations per temperature, 0.9 cool down rate, and 0.95 step decrease ratio.

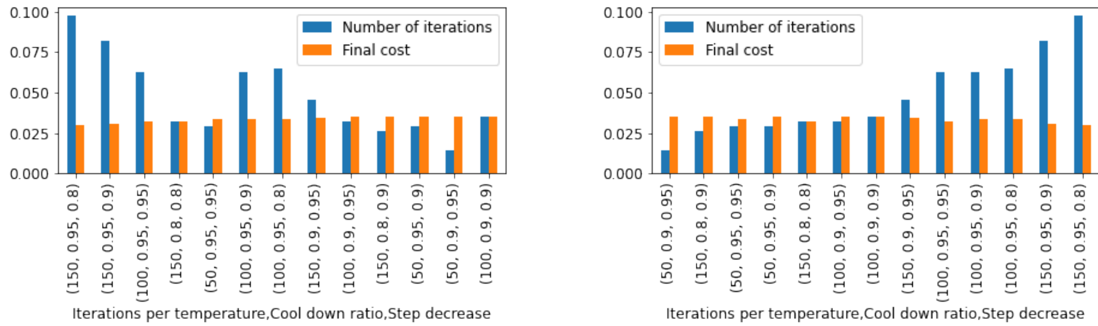



**Figure 6.8** Plots showing the results of SA hyperparameters tuning - iterations per temperature, cool down ratio and step decrease. The final cost is almost the same but the number of iterations grows significantly.

### Restarts

The restart technique is supposed to speed up the algorithm and improve the quality of its results. Its main function is to unstuck a solution from the local optimum by returning it to a more promising starting point. I experimented with four types of restarts:

**Random** There is a small chance $p$ that restart will happen in each iteration.

$$p = 0.005, 0.01, 0.05$$

**Number of iterations** If the algorithm cannot find a better solution for $p$ iterations, restart is performed.

$$p = 5, 10, 15$$

**Difference** Restart is triggered when the cost of the current solution is *1 - p* % greater than the cost of the best solution.

$$p = 0.4, 0.5, 0.6$$

**Combination of difference and number of iterations** Algorithm performs a restart when at least one of the conditions is satisfied. Either the current solution is much worse than the best one, or there was no improvement for a number of iterations.

$$p = (0.3, 10), (0.4, 10), (0.5, 10), (0.3, 20), (0.4, 20), (0.5, 20)$$

I tested multiple values of parameters for all restarts and a no restart for comparison. In figure 6.9 are plotted results sorted by type (left) and by final cost (right). Not all restarts are beneficial, and different types have different effects. The `RandomRestart` improves final cost but extends computation time significantly. `NumiterationsRestart` is very similar to not using the restart technique at all. `DiffRestart` is just increasing the number of iterations with no cost improvement. I decided to use `DiffNumCombinationRestart` with a 0.3 diff rate and 20 iterations without change because it achieved the lowest cost without significantly prolonging the run time.

## 6.3 Genetic algorithm

The furniture is arranged inside zones with GA. As the SA class, it has to be instantiated for every room and started with the `run()` method. The constructor of `GeneticAlgorithm` class

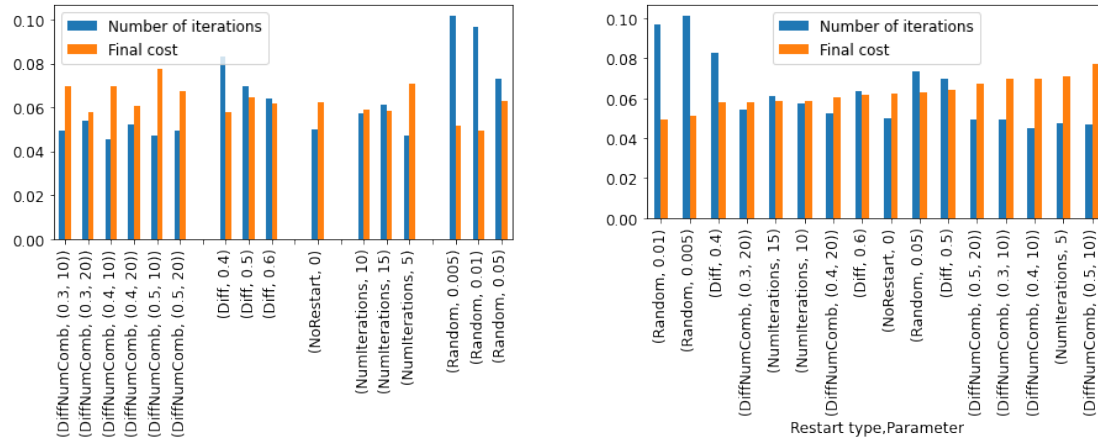



**Figure 6.9** Plots showing the results of SA restart types testing.

**Table 6.2** Initial groups size

| Type of groups | Final cost |
|---|---|
| Random groups | 0.473741 |
| Small groups | 0.526259 |

takes (in addition to others) an `InitializationOperator` and a list of `MutationTypes`. The GA optimizes several populations (zones) at once because I plan to implement cooperation between them (for example, objects overlapping).

In contrast with the original paper [14], I had to devise a new representation of furniture arrangement that better supports groups of furniture. I experimented with how to generate initial solutions. I tried creating chromosomes with one object in each group so that the algorithm can put together the best groups. As another option, I implemented the initialization operator that creates groups randomly.

I ran both initialization operators three times on all test rooms and averaged final costs for each room. Randomly created groups achieved better results, probably because they sample the search space better.

Because the number of furniture in the zones varies, I calculate the population size as the population size rate times the number of furniture in the zone. For the time being, the GA runs for a fixed number of iterations. So, I tested it with the population size rate to find the best combination.

As the best option proved both max iterations and population size rate equal 50.

Classic GA uses a high crossover rate and a low mutation rate. That ensures the convergence of the algorithm. However, the authors of the article [8] applied mutation on 50 % of the new population. I also experimented with high mutation rates, and they proved successful. Because I use a fixed number of iterations, I don't need the GA to converge fully. Figure 6.11 visualizes the final costs of runs with different reproduction rates. Crossover rate 75 % and mutation rate 75 % achieved the best results. Compared to that, classic GA with a low mutation rate and high crossover rate had the highest cost.

**Figure 6.10** Genetic algorithm for furniture arrangement

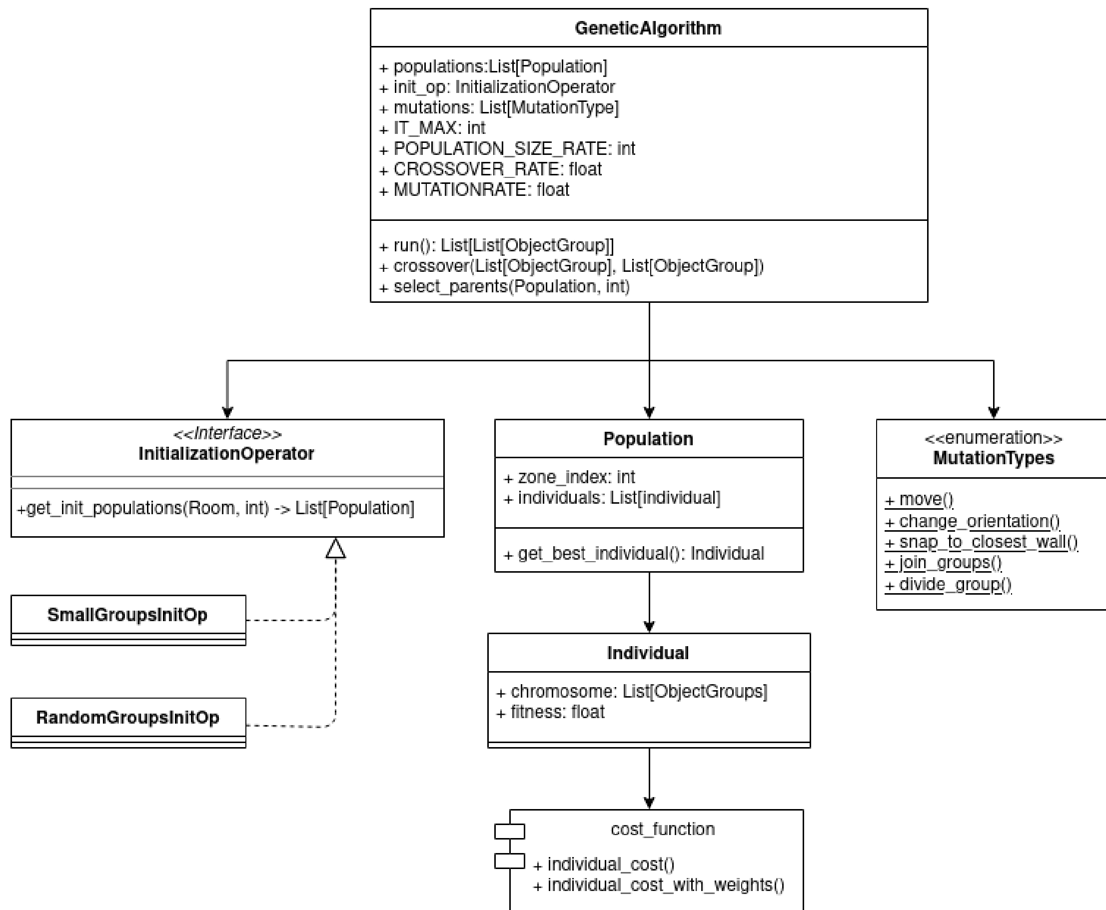



**Table 6.3** Population size rates and number of iterations

| Max iterations | Population size rate | Final cost |
|---|---|---|
| 50 | 50 | 0.091860 |
| 50 | 75 | 0.097976 |
| 75 | 75 | 0.098001 |
| 25 | 75 | 0.099138 |
| 25 | 50 | 0.103068 |
| 75 | 50 | 0.111065 |
| 75 | 25 | 0.125846 |
| 50 | 25 | 0.130175 |
| 25 | 25 | 0.142871 |

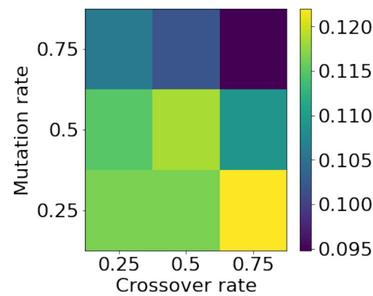



**Figure 6.11** Visualization of results for different combinations of mutation and crossover rates. The GA achieved lowest cost with both crossover and mutation rate 75 %.

## 6.4 Dataset

One of the contributions of his thesis is the incorporation of a large dataset of furniture. The 3D-FUTURE: 3D FUrniture shape with TextURE contains almost 10 000 models of furniture. An image and a obj model are provided for every furniture piece. Each model has a unique id, super-category, category, style, theme, and material.

I also added a size attribute calculated from the dimensions of the model. However, not all models have the same size units, most are in meters, but some are measured in centimeters. Because I translate all sizes to cm, some models have unrealistic dimensions. I solved this problem by checking if the model has some dimension larger than 10 m and if so, I divide it by 100.

The dataset isn't perfectly organized. Some objects are included in the wrong categories. This would have to be corrected by hand, but that is beyond the scope of this thesis.

Data are stored on the faculty server in a folder hierarchy (super-category/category/models). This is a temporary solution, and a database should be created in the future.

## 6.5 Chapter summary

The system is implemented in Python and is using Kivy library for user interface and Shapely package for geometric calculations. The program provides UI for entering the shape and features of the room, the functions of the room, and selecting furniture. The presentation of the results is divided into showing the positions of the zones and visualization of the final design.

The efficiency and the quality of results of simulated annealing were improved by tuning the method of initial temperature calculation, restart techniques, and SA's hyperparameters (iterations per temperature, cool down ratio, and step decrease ratio). To enhance the GA, I experimented with mutation and crossover rates, random and small initial group sizes, and reproduction rates. Unconventionally, I decided on a 75 % mutation rate because it achieved the best results in the experiments. I can use it because I run the GA for a fixed number of iterations.

I incorporated the 3D-FUTURE dataset into my work. It contains almost 10 000 models of categorized furniture. However, it is not perfectly organized.

# Chapter 7

# Results

This chapter presents results of experiments on automatic interior design. By “testing of the system” I mean testing the selected algorithms whether they can produce usable interior designs. The goal of this thesis isn’t to create a perfect software product but to experiment with the algorithms.

## 7.1 Test set

The proposed system was tested on a set of rooms that consists of 14 L-shaped rooms, 12 rectangular rooms, and 10 rooms of other varying shapes. Rooms contain different combinations of functions and furniture. Figure 7.1 demonstrates the diversity of test data.

The shape of the room is drawn with black lines and coordinates in the corners. Windows are marked with blue, doors are marked with light brown. The zones are visualized as colorful rectangles. Each color corresponds to a zone function:

| Function | Color |
|---|---|
| conversation | yellow |
| sleep | blue |
| work | red |
| relaxation | green |
| dinning | purple |
| storage | brown |

Grey rectangles indicate positions of objects with the category written on them. The actual area of the object is drawn in darker grey, and it has a red line on the front side. The lighter grey visualizes the space around the object that is supposed to stay free.

## 7.2 Run times

I ran the design process ten times for each test room and collected data about their run times. Figure 7.2 shows the dependency of a run time of simulated annealing on a number of zones in the room. Even though there is some randomness involved, the run times follow an exponential curve. The most time-consuming operation in SA is the random neighbor generation and the calculation of the cost function. The cost function is executed in every iteration, and the more zones there are, the more complicated the computation is. When generating a neighbor, a solution (containing positions of all zones) changes a random number of its zones. That causes a larger variance of the run times.

**Figure 7.1** On the left are counts of room features and storage and other zones. The right part shows histograms of room areas and areas of furniture in zones.

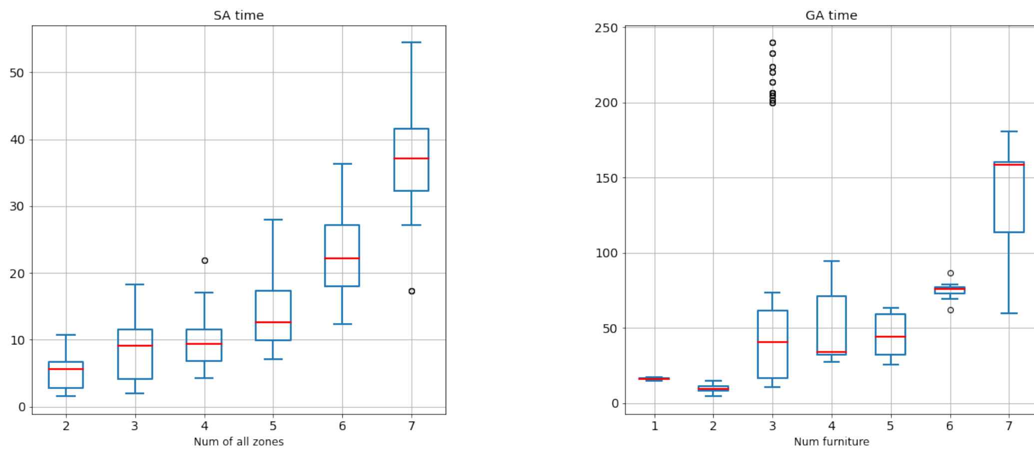



**Figure 7.2** Run time of SA in dependence on number of zones (left). Run time of GA in dependence on number of furniture (right).

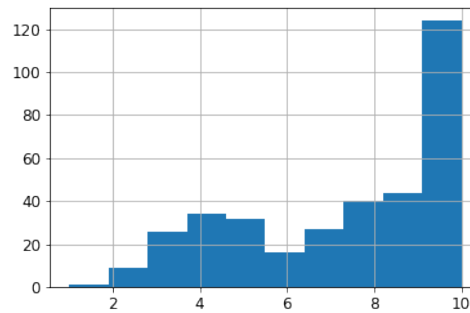



**Figure 7.3** The histogram of scores

The results of GA aren't that even. The run time grows with the growing number of objects, but it also depends on the composition of furniture in the zone. Some categories of furniture have more relationships than the others. That influences computation speed because, during a cost function calculation, the cost of every object's relationship is calculated.

## 7.3 Experiments

Even in real life, it isn't easy to measure the quality of an interior design. The evaluation will always be subjective. Authors of related papers used perceptual studies or comparisons with other similar works to assess the quality of their methods. This is beyond the scope of this thesis and can be a subject for the following works.

To decide about the system's usability, I ran the design process ten times for each test room and created a subjective evaluation of the results. I assigned each layout the following attributes:

**Label** I marked the best and the worst design from all trials on the specific room.

**Room block** If there is an inaccessible part of the room (blocked by furniture).

**Usability** The design is usable if there are no inaccessible parts of the room and no objects overlap.

**Relationships** How many objects are correctly arranged towards another object.

**Score** Subjective impression of the room based on arrangement of the furniture. Unusable designs cannot achieve a score higher than 5.

74.79 % of the designs are usable. Figure 7.3 shows the histogram of scores. Most of the designs that got a score of 4 or 5 had one group of objects blocking a part of the room, making the design unusable, but the rest of the layout was good.

Table 7.1 shows the distribution of usable and unusable designs. Out of 353 layouts, 263 were usable. Most of the unusable arrangements were caused by blocking a part of the room. That should be prevented by the combination of overlap penalty for zone arrangement and the penalty for overhanging the zone during the furniture positioning. However, because zones are in some cases too small to accommodate its furniture, the objects have to partly stand out of the zone.

As was described in chapter 3, both simulated annealing and genetic algorithms were already used for automatic furniture arrangement and achieved promising results. The novelty of the system is in using them together and for different parts of the design process. The simulated annealing wasn't, to my knowledge, applied to the planning of the functional zones. The genetic

**Table 7.1** The usability of designs

| | Overlapping objects | No overlapping objects |
|---|---|---|
| Part of the room inaccessible | 12 | 65 |
| All parts of the room accessible | 13 | 263 |

algorithm was used in [8] to arrange furniture, but I am narrowing its search space by giving it one zone at a time.

The figures 7.4, 7.5, 7.6, 7.7, 7.8 present the examples of the generated designs. They are grouped according to their average scores from all test runs. The important features that make the rooms from figure 7.4 easily optimizable are enough space in the room and the simplicity of their relationships. On the other hand, the rooms in figure 7.8 have an average score below 5. Those rooms are extremely crowded or contain furniture groups that have complicated relationships.

The results show the ability of a system to design rooms with varying shapes and features. The zone arrangement needs an improvement because most unsatisfying results were caused by a small area or dimensions of the zones. It will be a subject of future research.

## 7.4 Comparison with our paper

My Bachelor thesis builds on work from a Research Summer program at the Faculty of Information Technology. During this program, I developed the first versions of simulated annealing and genetic algorithm. In cooperation with my supervisor Ing. Mgr. Ladislava Smítková Janků, Ph.D., I wrote a paper about it that was accepted at IEA/AIE 2021 conference and will be published in July.

The contributions of this thesis are following:

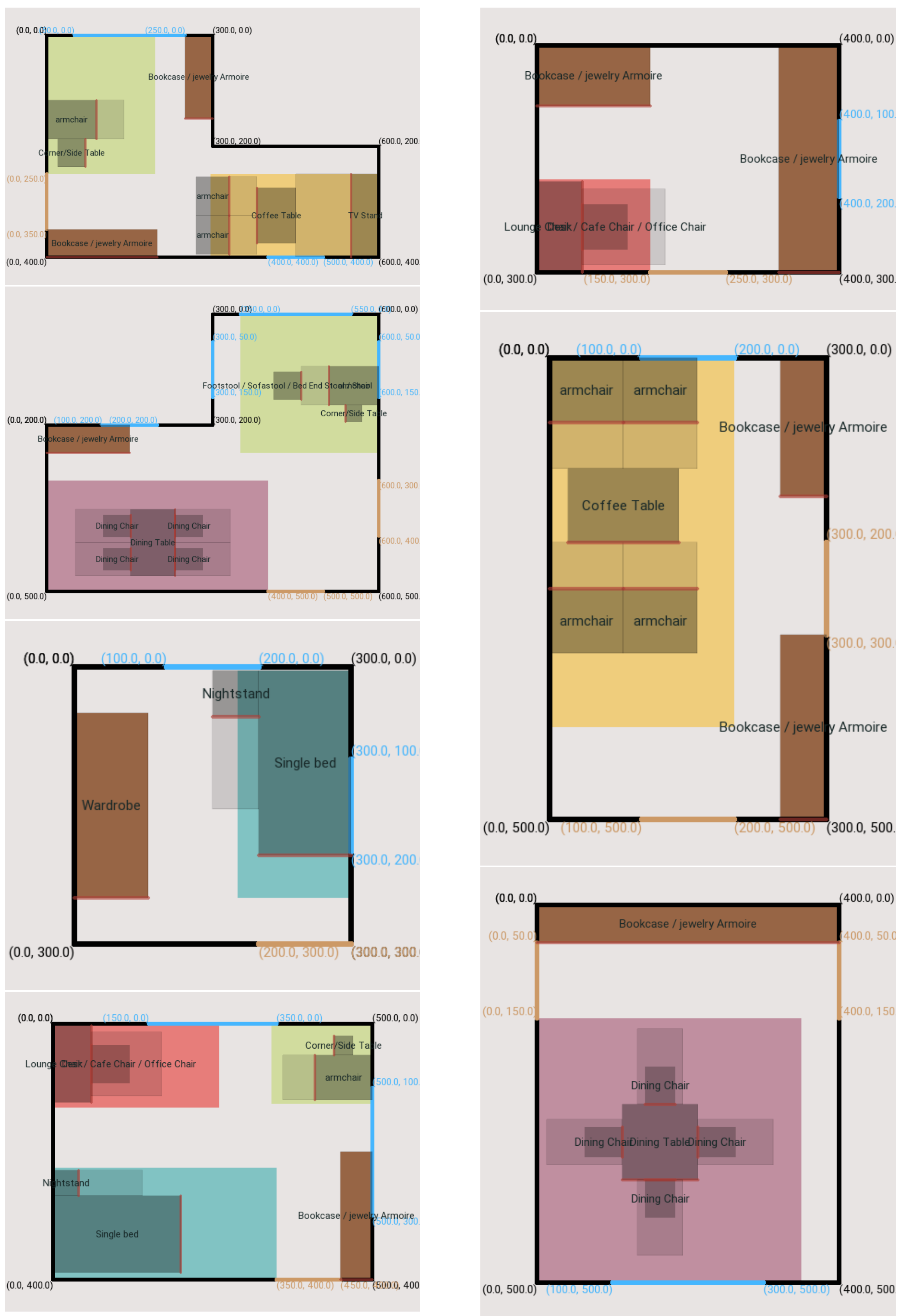
**Object relationships** The paper works with manually created input data where relationships are defined on the level of furniture. The relationships form a spatial graph that should be satisfied. The current system incorporates a large dataset of categorized furniture with multiple possible relationships defined for every category. The separation of relationships from concrete objects allows greater extensibility.

**Object groups in GA** The GA in our paper optimizes the positions of objects. The thesis introduces groups of objects in an individual's chromosome. That makes the arrangement of furniture towards each other easier.

**User interface** The original algorithms took input from XML and configuration files and outputted an image of the design. Now, the system has a simple UI for taking user requirements and presenting the result.

**Improvement of SA and GA** I added a restart technique to enhance SA and experimented with different combinations of hyperparameters. GA had to be modified to work with object groups.

Figure 7.9 shows the same (or similar) rooms optimized with the original algorithms and the the new system.




■ **Figure 7.4** The rooms that achieved the average score from all runs higher than 9.

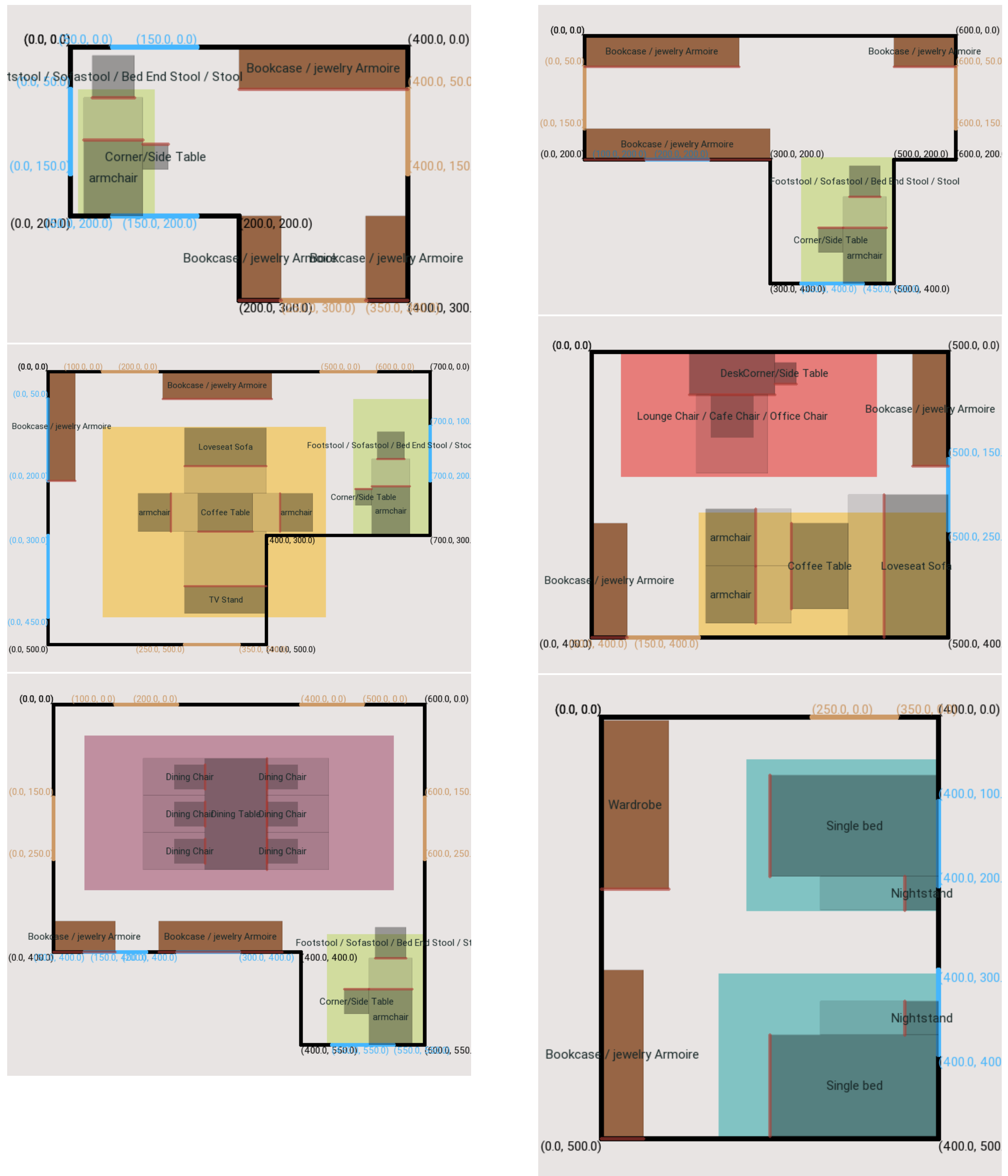
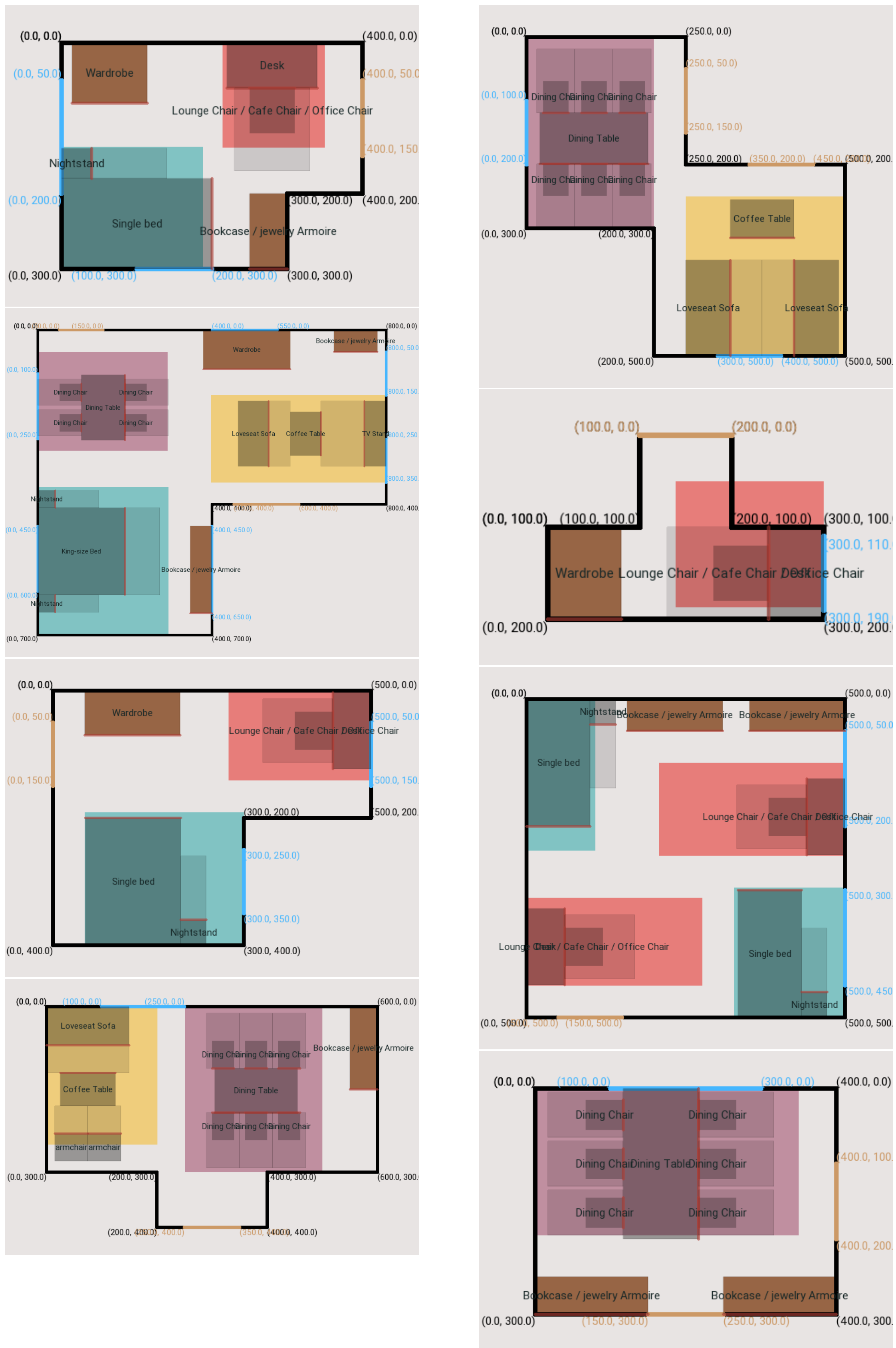



**Figure 7.5** Examples of rooms with average score between 8 and 9.




■ **Figure 7.6** Examples of rooms with average score between 7 and 8.

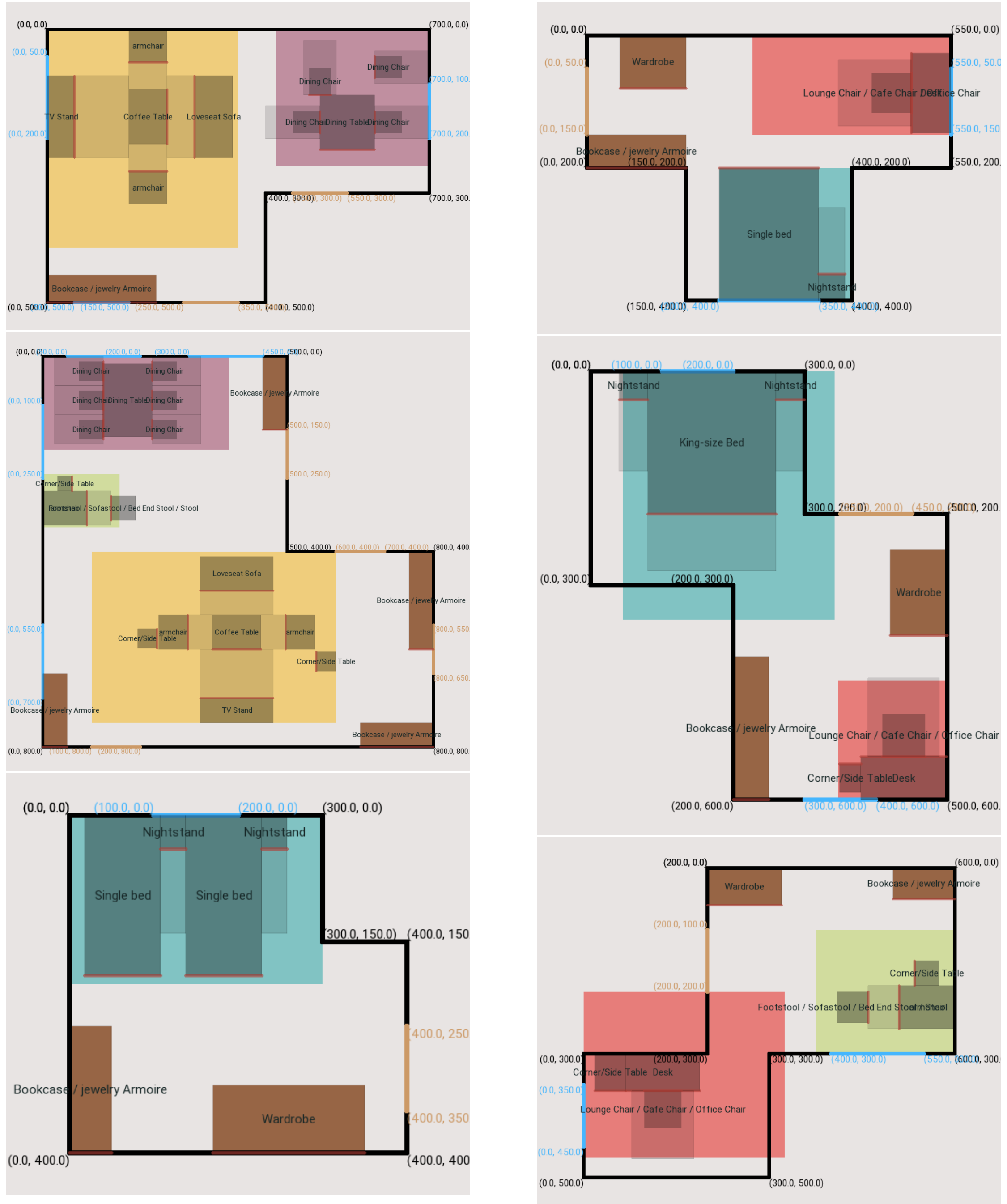



**Figure 7.7** Examples of rooms with average score between 6 and 7.

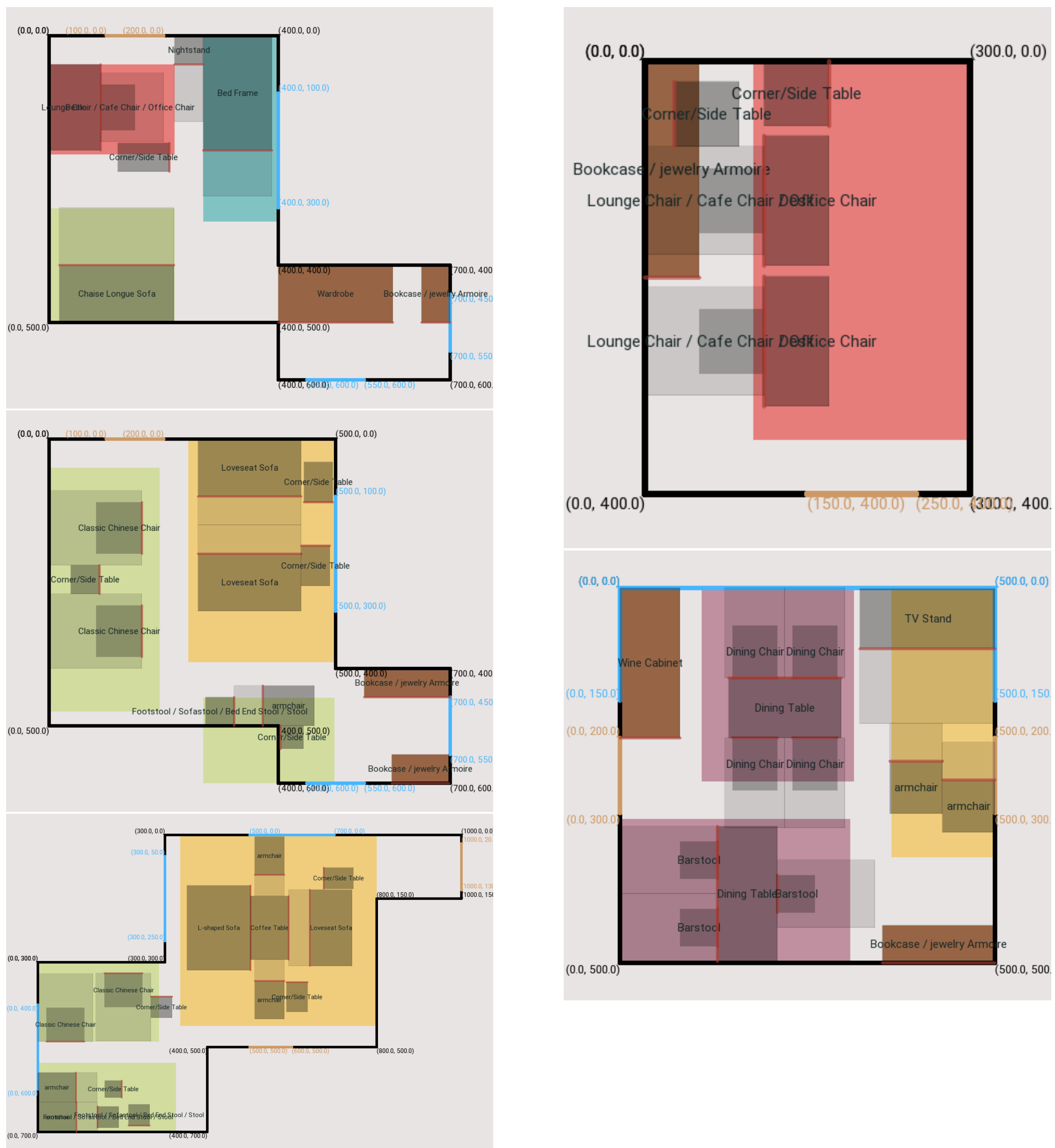
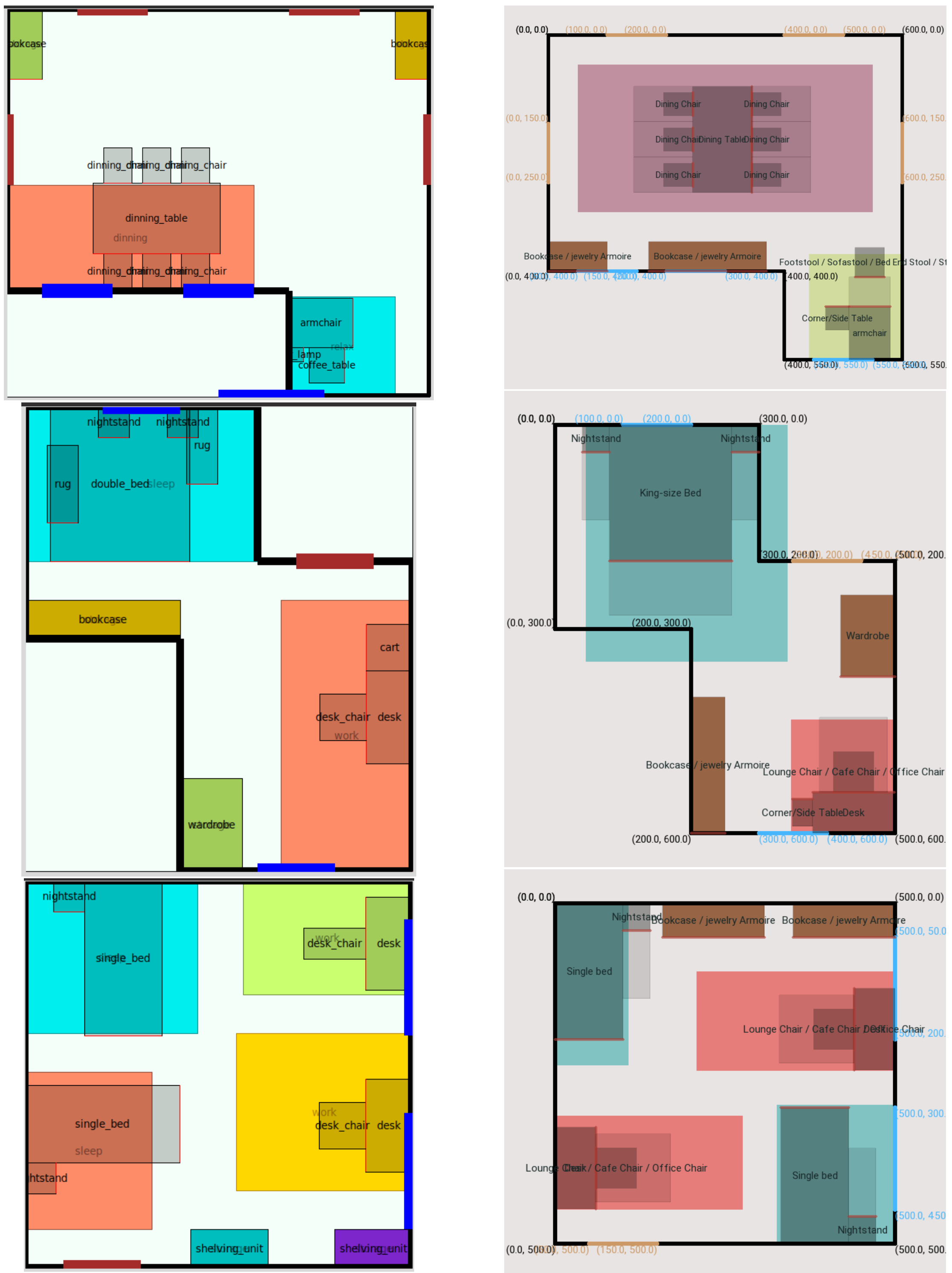



**Figure 7.8** The rooms that algorithms weren't able to sufficiently optimize (average score below 5). They contain a lot of furniture or have complicated relationships.




■ **Figure 7.9** On the left are results of the algorithms presented in our paper. On the right are rooms optimized with the system presented in this thesis.

## 7.5 Chapter summary

The proposed system was tested on a diverse set of 36 rooms. The design process was run ten times on each room, and 75 % of the results were usable. The rest was unusable mainly because one group of objects blocked a part of the room. The examples demonstrate the usability of the system for interior design.

The system was compared with the first implementation of core algorithms that will be published in a paper at IEA/AIE 2021. The contributions of this thesis include the separation of relationship definitions from concrete objects, incorporation of a large dataset of categorized furniture, implementation of the user interface, and improvement of algorithms for zone planning and furniture arrangement.

# Chapter 8
# Conclusion

This bachelor thesis aimed to create an algorithm for automatic furnishing of a given interior using artificial intelligence methods. The algorithm has to be implemented as a part of an experimental system that accepts requirements from the user and have a graphical user interface.

First, I summarized the research on automatic interior design and categorized it into three groups - search-based, graph-based and data-based. Search-based methods can incorporate a wide variety of user requirements, but they are slow (one layout takes seconds or minutes to generate). Graph-based methods are capable of capturing complicated relationships between objects. Their disadvantage is that the spatial graph itself does not uniquely define a layout. So they have to be used with another method that finds a design satisfying the spatial graph. The most significant advantage of data-based methods is their speed (one layout is generated in a couple of milliseconds). However, they rely on neural networks that are trained only to imitate human-created designs, so they are less able to incorporate user requirements.

The design process proposed in this thesis is divided into high-level planning of functional zones and low-level furniture arrangement inside these zones. I chose simulated annealing for the planning of the functional zones because the cost function that evaluates them is very noisy. SA can overcome the many local minimums. Furniture is arranged by a genetic algorithm. This algorithm searches for the best composition of groups of furniture and their positions.

I implemented SA with an exponential annealing schedule and decreasing step size. A solution is composed of positions of all zones and evaluated based on overlapping, shapes of its zones, and their distances from walls, windows, and doors. The performance of SA is enhanced by a restart technique that returns to the best solution found so far. I experimented with different methods to decide the initial temperature, restart techniques, and combinations of hyperparameters to improve the efficiency and quality of results of SA.

The genetic algorithm evolves several populations at once - one per functional zone. Chromosomes consist of groups of objects where each of them is arranged together procedurally. The compositions of the groups and their positions are optimized with a crossover operator and mutations. A new population consists of ten best individuals from the previous generation and individuals created by crossover and mutation. 75 % individuals are selected as parents with stochastic universal sampling and produce children. The rest is made of the best individuals. A mutation is applied to 75 % of these individuals. The GA runs for 50 iterations. The reproduction rates were selected based on experiments with their combinations.

Simulated annealing and a genetic algorithm were already applied on automatic interior design and achieved promising results. The novelty of the proposed method lies in combining them together and using them for different parts of the design process.

I tested the arranging process on 36 different rooms. I ran the design process ten times for every room and assigned a score to each of the results. 75 % of the resulting layouts were usable

(with no parts of the room blocked and no overlapping of the furniture). The most mistakes in the designs were caused by zones that were too small to accommodate its furniture. The zone planning will be the subject of future work. The chapter 7 provides examples of interior designs created by the system. It was able to furnish rooms of varying shapes, number of zones and number of furniture.

This thesis builds on a work from the Research Summer on Faculty of Information Technology (CTU) where I developed the first version of core algorithms (GA, SA) and wrote a paper [14] on it in cooperation with my supervisor, Ing. Mgr. Ladislava Smítková Janků, Ph.D. The paper was accepted at IEA/AIE 2021 conference. The contributions of this thesis are incorporation of a 3D-FUTURE dataset of categorized furniture, separating concrete objects from general properties of their categories, and implementation of user interface. The core algorithms were also improved - the SA algorithm was enhanced by restart technique and tuned, and the GA now operates on self-arranging groups of objects.

The proposed method is part of the more extensible research on usage of artificial intelligence methods for automatic interior design. It will be utilized into a larger system.

I will continue to work on the system and improve the zone planning. Additionally, I want to focus on combining furniture into hierarchies and their arrangement inside them. Also, further study of the evaluation of designs is necessary. The system as a software product provide many possibilities for future works. For example, the user interface lacks input data validation and a drawing interface would be more user-friendly than a command line for specifying the shape of the room.

# Bibliography

[1] Sherif Abdelmohsen, Ayman Assem, Sherif Tarabishy, and Ahmed Ibrahim. A heuristic approach for the automated generation of furniture layout schemes in residential spaces. In John. S Gero, editor, *Design Computing and Cognition '16*, pages 459–475, Cham, 2017. Springer International Publishing.

[2] Franco Busetti. Simulated annealing overview, 05 2001. [2021-04-25]. URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.5018&rep=rep1&type=pdf`.

[3] Xinhan Di and Pengqian Yu. Deep reinforcement learning for producing furniture layout in indoor scenes. *ArXiv*, abs/2101.07462, 01 2021.

[4] Huan Fu, Rongfei Jia, Lin Gao, Mingming Gong, Binqiang Zhao, Steve Maybank, and Dacheng Tao. 3d-future: 3d furniture shape with texture. *arXiv preprint arXiv:2009.09633*, 2020.

[5] Qiang Fu, Xiaowu Chen, Xiaotian Wang, Sijia Wen, Bin Zhou, and Hongbo Fu. Adaptive synthesis of indoor scenes via activity-associated object relation graphs. *ACM Trans. Graph.*, 36(6), November 2017. `doi:10.1145/3130800.3130805`.

[6] Paul Henderson, Kartic Subr, and Vittorio Ferrari. Automatic generation of constrained furniture layouts, 2019. `arXiv:1711.10939`.

[7] Peter Kan and Hannes Kaufmann. Automatic furniture arrangement using greedy cost minimization. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 491–498, 03 2018. `doi:10.1109/VR.2018.8448291`.

[8] Peter Kán and Hannes Kaufmann. Automated interior design using a genetic algorithm. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*, pages 1–10, 11 2017. `doi:10.1145/3139131.3139135`.

[9] Manyi Li, Akshay Patil, Kai Xu, Siddhartha Chaudhuri, Owais Khan, Ariel Shamir, Changhe Tu, Baoquan Chen, and Daniel Cohen-Or. Grains: Generative recursive autoencoders for indoor scenes. *ACM Transactions on Graphics*, 38:1–16, 02 2019. `doi:10.1145/3303766`.

[10] Yuerong Li, Xingce Wang, Zhongke Wu, Shaolong Liu, and Mingquan Zhou. Flexible indoor scene synthesis via a multi-object particle swarm intelligence optimization algorithm and user intentions. In *2019 International Conference on Cyberworlds (CW)*, pages 29–36, 2019. `doi:10.1109/CW.2019.00014`.

[11] Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. Human-centric indoor scene synthesis using stochastic grammar. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5899–5908, 2018. doi:10.1109/CVPR.2018.00618.

[12] Daniel Ritchie, Kai Wang, and Yu-An Lin. Fast and flexible indoor scene synthesis via deep convolutional generative models. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6175–6183, 06 2019. doi:10.1109/CVPR.2019.00634.

[13] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[14] Eliška Svobodová and Ladislava Smítková. Method for automatic furniture placement based on simulated annealing and genetic algorithm. In *The 34th International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems (IEA/AIE 2021)*, 07 2021. [will be published].

[15] Kai Wang, Yu-An Lin, Ben Weissmann, Manolis Savva, Angel Chang, and Daniel Ritchie. Planit: planning and instantiating indoor scenes with relation graph and spatial prior networks. *ACM Transactions on Graphics*, 38:1–15, 07 2019. doi:10.1145/3306346.3322941.

[16] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4, 10 1998. doi:10.1007/BF00175354.

[17] Bailin Yang, Liuliu Li, Chao Song, Zhaoyi Jiang, and Yun Ling. Automatic furniture layout based on functional area division. In *2019 International Conference on Cyberworlds (CW)*, pages 109–116, 2019. doi:10.1109/CW.2019.00026.

[18] Lap-Fai Yu, Sai Yeung, Chi-Keung Tang, Demetri Terzopoulos, Tony Chan, and Stanley Osher. Make it home: Automatic optimization of furniture arrangement. *ACM Trans. Graph.*, 30:86, 07 2011. doi:10.1145/2010324.1964981.

[19] Song-Hai Zhang, Shao-Kui Zhang, Wei-Yu Xie, Cheng-Yang Luo, and Hong-Bo Fu. Fast 3d indoor scene synthesis with discrete and exact layout pattern extraction, 02 2020.

Chapter 9

# Contents of attached medium

```
readme.txt ........................ a brief description of medium contents
src
├── impl ............................ source codes of the implementation
├── experiments ..................... inputs and results of experiments
└── thesis .......................... source of the thesis in LaTeX
text ................................ text of the thesis
└── thesis.pdf ...................... text of the thesis in PDF
```