AI-Driven Generative Design: Evolutionary Optimisation of Residential Floor Plans

Zuxing Wu, a1816653

Supervisor: Adel Nikfarjam

Master of Computer Science

June 16, 2025

Contents

1 Introduction

1.1 Aim

Residential floor plan design is a complex and challenging task that requires careful consideration of factors such as room dimensions, spatial adjacency, privacy, convenience, and orientation. Traditional approaches to floor plan design are often time-consuming and labour-intensive, frequently resulting in suboptimal outcomes. Evolutionary algorithms have emerged as a promising alternative for optimising floor plans, as they can efficiently explore the design space and generate high-quality solutions. This project aims to develop an improved method for optimising residential floor plans through the application of evolutionary algorithms, addressing several limitations identified in previous research. The proposed approach utilises a novel and straightforward representation scheme, with the fitness of each floor plan evaluated according to multiple criteria, including privacy, comfort, practicality, and convenience. The findings of this research are expected to advance the field of residential floor plan design and offer valuable insights into the application of evolutionary algorithms to architectural design challenges.

1.2 Motivation

Residential floor plan design is a fundamental component of architectural practice, involving the spatial arrangement of rooms, windows, doors, corridors, and ancillary spaces within a dwelling. The configuration of a floor plan significantly influences the functionality, convenience, comfort, ventilation, and energy efficiency of a residential building. Traditional approaches to floor plan design typically rely on manual drafting or computer-aided design (CAD) tools, which are often both time-consuming and labour-intensive. Furthermore, such methods may yield suboptimal outcomes, as they are heavily dependent on the subjective judgment and experience of individual designers.

In contrast, evolutionary algorithms provide a systematic and effective means for optimising floor plans by thoroughly exploring the design space and generating high-quality solutions. These algorithms are inspired by the principles of natural evolution, operating on a population of candidate solutions and utilising mechanisms such as crossover and mutation to iteratively improve design quality across generations. The performance of each candidate solution is rigorously evaluated using fitness functions, which quantitatively assess the extent to which the design satisfies the specified objectives of the floor plan optimisation problem.

2 Literature Review

Previous research has explored the application of evolutionary algorithms to optimise residential floor plans. Brintrup et al. [1] compared three interactive genetic algorithms (i.e., sequential IGA, multi-objective IGA, and parallel

IGA) on a multi-objective floor planning task, finding that the multi-objective IGA provides more diverse results and achieves faster convergence for floor plan optimisation. They developed interactive evolutionary algorithms that allow designers to incorporate their preferences and constraints into the optimisation process. This method has shown promising results in generating floor plans that meet both functional and aesthetic requirements.

It was found that proportional roulette wheel selection is the most effective parent selection method for the mating pool, and k-point crossover is the most effective for improving fitness during evolution [3]. Combining evolutionary algorithms with greedy-like algorithms can help find near-optimal solutions in Automated Floor Plan Generation (AFPG), although this approach simplifies multi-objective optimisation by linearly combining partial evaluation functions [8]. Subramanian et al. [12] used a genetic algorithm with KD tree models in a web application to generate floor plans accessible even to non-expert users.

Wang and Duan [14] attempted to optimise floor plan design by focusing on energy consumption and consumer satisfaction, demonstrating that the preferences of different types of consumers vary significantly. Therefore, different evaluation criteria are needed to satisfy different family types. The quality and efficiency of residential floor plan design can be improved by combining the Monte Carlo Tree Search (MCTS) algorithm and Particle Swarm Optimisation (PSO) [15]. The MCTS algorithm incorporates human experience, which helps compress the search space and improve search efficiency [15]. The PSO algorithm can handle continuous variables and is suitable for optimising room sizes due to its parallel processing capability [15].

Energy consumption, probable uniformity (PU), and spatial useful daylight illuminance (sUDI) are three objectives considered in Chaichi and Andaji Garmaroodi's optimisation process [2] using the NSGA-II algorithm. The results show that NSGA-II can provide a set of more sustainable floor plans that require less computational power and time. Reliable metrics for evaluating the amount and uniformity of light are tested in the optimisation process, which can help avoid unwanted convergence by imposing mutual restrictions [2].

Shi et al. [10] chose the (1+1) Evolutionary Algorithm (EA) to optimise macro placement by randomly selecting two macros and exchanging their coordinates, maintaining only one solution and iteratively improving it. Laignel et al. [7] proposed a novel constraint programming-genetic algorithm for automatic apartment layout generation, demonstrating its ability to produce architecturally valid floor plans within a minute by discretising space into a grid and solving layout assignments under complex constraints.

Zou et al. [17] proposed a memory-based simulated annealing algorithm for fixed-outline floor planning with soft blocks, using a memory pool and geometric auxiliary functions to escape local optima and efficiently meet design constraints, showing superior performance on benchmarks. Both Zou [17] and Singha [11] used B*-tree structures to represent the floor plan, which is a hierarchical structure that allows for efficient storage and manipulation of floor plan data. Kang and Kim [5] presented a novel method that analyses mobile app logs and Google server data to identify user behaviour patterns, using a genetic algorithm to generate optimal indoor floor plans that minimise living costs and enhance spatial analysis. Zawidzki and Szklarski [16] developed a multi-objective optimisation framework for single-story house floor plans, balancing functionality, sunlight, view, and noise reduction through gradient-based methods and user-defined preferences, as demonstrated in a case study.

Furthermore, the integration of machine learning techniques with evolutionary algorithms has gained attention in recent decades [4]. For example, Wang et al. [13] proposed a hybrid approach that combines a genetic algorithm with a neural network to predict the fitness of candidate solutions. This method significantly reduces the computational cost of evaluating large populations and accelerates the optimisation process. Ploennigs and Berger [9] proposed using diffusion-based generative AI to automate architectural floor plan design, improving the validity of generated layouts from 6% to 90% and highlighting both its potential and challenges in the design automation process.

Overall, the literature indicates that evolutionary algorithms, particularly when combined with other optimisation techniques and machine learning methods, offer a powerful tool for optimising residential floor plans. These approaches not only improve the efficiency and quality of the designs but also provide flexibility in accommodating various design preferences and constraints.

3 Methodology

The methodology of this project involves the application of evolutionary algorithms to optimise residential floor plans. The process will be divided into several key stages: 1. Representation of the floor plan, 2. Solution evaluation, 3. Initialisation of the population, 4. Design of the evolutionary optimisation framework. These stages are described in detail below.

3.1 Representation of Floor Plans

To represent a concrete representation of the floor plan, we will use a hierarchical structure (shown in Figure 1) to represent the floor plan data. The top-level structure will be a `House` class that contains a boundary and a list of rooms. Each room will be represented by a `Room` class, which includes its dimensions (width and depth), position (x, y coordinates), and lists for doors and windows. The doors and windows will be represented as tuples containing their width, the wall they are located on, and their position on that wall.

To display the floor plan, we used Python's `matplotlib` library to create and save a visual representation of the floor plan. We defined a `save_snapshots` function that takes a layout as one of the parameters, and generates a visual representation of the floor plan. The function iterates through each room in the layout, draws rectangles for the rooms, and adds doors and windows as lines on the walls. The boundary is drawn in black. The resulting image is saved as a PNG file.

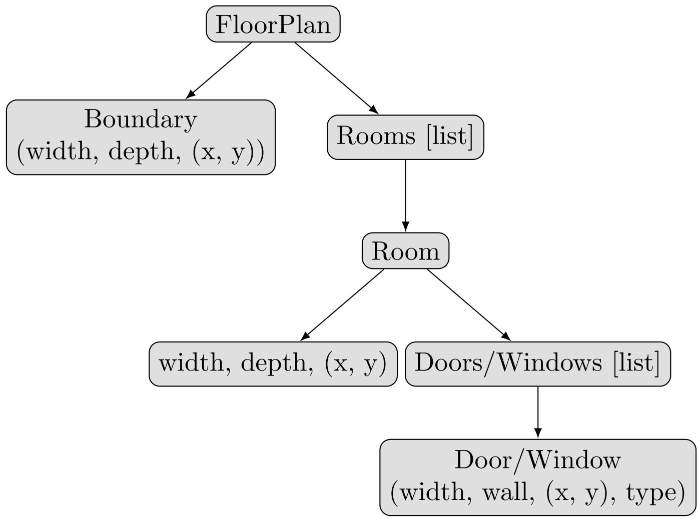


Figure 1: Hierarchical representation of the floor plan data structure.

3.2 Solution Evaluation

Fitness functions will evaluate each floor plan mainly based on these 4 criteria: privacy, comfort, practicality, convenience. We created 24 fitness functions based on Wang and Duan's evaluation indicators [14] along with professional's advice. Our fitness functions are shown in Table 1.

The straight distance between the geometric centres of room m and n is defined as:

$$\mathrm{dis}(m, n) = \sqrt{(x_m - x_n)^2 + (y_m - y_n)^2}$$

but at the late stage of the optimisation process, this distance is calculated through `polygon.distance()` function from the `shapely` library, which is more accurate and convenient than the above formula.

The percentage of area of R to the interior area of the floor plan is defined as:

$$\% \text{ of } R = \left(\frac{\text{Area of } R}{\text{Interior Area of Floor Plan}} \right) \times 100\%$$

The utilisation rate of a house is defined as the percentage of occupied area to the total area of the floor plan:

$$\text{utilisation Rate} = \left(\frac{\text{Occupied Area}}{\text{Total Area}} \right) \times 100\%$$

The overlap rate is defined as the percentage of overlapping area to the total area of the floor plan:

$$\text{overlap Rate} = \left(\frac{\text{Overlapping Area}}{\text{Sum All Rooms Area}} \right) \times 100\%$$

Table 1: List of 24 Fitness Functions (Green means eventually being used)

Function Name	**Description**
dis_MBR_BR	Distance between Main Bedroom and Bedroom
dis_MBR_BA	Distance between Main Bedroom and Bathroom
check_LR_orientation	Living Room orientation check
check_DR_natural_light	Dining Room natural light check
ventilation	Ventilation evaluation
north_facing_area	North-facing area calculation
percentage_hall	Percentage of hall area
percentage_balcony	Percentage of balcony area
utilization_rate	Utilization rate of the floor plan
dis_BR_BA	Distance between Bedroom and Bathroom
dis_BA_LR_door	Distance between Bathroom and Living Room door
check_GAR_side	Garage should be on the same side as the entry
dis_KIC_LR	Distance between Kitchen and Living Room
dis_KIC_DR	Distance between Kitchen and Dining Room
dis_LR_DR	Distance between Living Room and Dining Room
check_no_entry_touch	Entry not touching other closed rooms check
cal_overlap_rate	Overlap rate calculation
diff_public_private	Differentiate private rooms from public rooms
check_KIC_orientation	Kitchen orientation check
dis_LDR_KIC	Distance between Laundry and Kitchen
dis_GAR_KIC	Distance between Garage and Kitchen
dis_MBR_LR	Distance between Main Bedroom and Living Room
dis_MBR_KIC	Distance between Main Bedroom and Kitchen
bigger_MBR	Main Bedroom area > other bedroom area

The differentiation between public and private rooms is based on the classification of rooms into public (e.g., living room, dining room, kitchen, garage and laundry) and private (e.g., bedrooms, bathrooms). The differentiation is evaluated based on the comparison of distance between these rooms and the entry. All public rooms should be close to the entry, while private rooms should be further away. The differentiation is calculated as:

$$\text{diff\_public\_private} = \frac{1}{|P| \cdot |Q|} \sum_{p \in P} \sum_{q \in Q} \mathbb{I}\left(d(\text{entry}, p) < d(\text{entry}, q)\right)$$

where P is the set of public rooms, Q is the set of private rooms, $d(\text{entry}, r)$ denotes the distance from the entry to the centre of room r, and $\mathbb{I}(\cdot)$ is the indicator function that returns 1 if the condition is true and 0 otherwise. This metric measures the proportion of public-private room pairs where the public room is closer to the entry than the private room, thus quantifying the spatial differentiation between public and private spaces.

These proposed methods are implemented in Python. All fitness values are calculated and normalised to a range of 0 to 1, where higher values indicate better performance. The evaluation process involves calculating the fitness values for each floor plan in the population and selecting the best individuals for particles updating their velocity and position in PSO.

3.3 Population Initialisation

The first step in the evolutionary process is to generate an initial population of residential floor plans. The initialisation process involves generating a set of random rooms and arranging them into a floor plan layout, subject to the following constraints and parameter ranges:

- **House boundary:** width = 15 m, depth = 8 m.
- **Entry:** a line segment at the boundary, `LineString([(0, 4), (0, 5)])`.
- **8 Room types and size ranges:**
  - Garage (`GAR`): width 5–6 m, depth 3–6 m
  - Laundry Room (`LDR`): width 3 m, depth 3 m
  - Dining Room (`DR`): width 3–5 m, depth 3–5 m
  - Living Room (`LR`): width 3–6 m, depth 3–6 m
  - Kitchen (`KIC`): width 3–4 m, depth 3–4 m
  - Main Bedroom (`MBR`): width 3–6 m, depth 3–6 m
  - Bedroom1 (`BR1`): width 3–5 m, depth 3–5 m
  - Bathroom (`BA`): width 2–4 m, depth 2–4 m
- **Window width:** 0.5–1.5 m
- **Door width:** 0.8–1.5 m

The rooms are assigned random sizes within the specified ranges and placed within the house boundary. The initialisation strategy is designed to produce a diverse set of floor plans that serve as a starting point for the evolutionary optimisation process. At the late stage of our research, we only need to randomly generate a single floor plan, which will be mutated and evaluated continuously so as to evolve the floor plan towards an optimal solution.

3.4 Evolutionary Optimisation Framework

The evolutionary optimisation framework initially was based on particle swarm optimisation (PSO) algorithm, which is suitable for optimising the size of rooms due to parallel processing, and Monte Carlo Tree Search (MCTS) algorithm which can handle discrete variables (i.e., room position) [15]. In our implementation, the width and depth are integer values, which means we are using discrete binary particle swarm optimisation (BPSO) algorithm [6, 15]. The topology of this type PSO algorithm is a von Neumann topology [4], where each particle is connected to its neighbours in a grid-like structure. The PSO algorithm (see Algorithm 1) updates the velocity and position of each particle based on its own best position and the best position of its neighbours. The velocity update equation is given by:

$$v_i(t+1) = w(t) \cdot v_i(t) + c_1 \cdot r_1 \cdot (p_i - x_i(t)) + c_2 \cdot r_2 \cdot (g - x_i(t))$$

where $v_i(t)$ is the velocity of particle i at time t, $w(t)$ is the inertia weight, c_1 and c_2 are the cognitive and social coefficients, respectively, r_1 and r_2 are random numbers uniformly distributed in [0, 1], p_i is the best position of particle i, and g is the global best position among all particles.

Algorithm 1 : PSO-MCTS Hybrid Algorithm

Require: Room size ranges, number of particles N, maximum iterations T, MCTS iterations M
Initialise each particle with random room sizes and zero velocities
Set global best fitness g_{best} to $-\infty$
for $t = 1$ to T **do**
    for each particle i **do**
        Construct room list from particle's sizes
        Use MCTS to generate a layout for these sizes
        Evaluate fitness of the layout
        if fitness > particle's personal best **then**
            Update particle's personal best
        end if
        if fitness > g_{best} **then**
            Update global best layout and fitness
        end if
    end for
    Local search: Swap two random rooms in g_{best} layout, accept if fitness improves
    Record g_{best} fitness in history
    for each particle i **do**
        for each room dimension j **do**
            Update velocity using inertia, cognitive, and social terms
            Update room size within allowed bounds
        end for
    end for
    if t is 0, $T - 1$, or divisible by 25 **then**
        Save snapshot of current g_{best} layout
    end if
end for
return Best layout found

However, the MCTS algorithm did not optimise the room position very well, so it was replaced by a simple greedy algorithm called (1+1) Evolutionary Algorithm (EA). The (1+1) EA algorithm (see Algorithm 2) is a simple evolutionary algorithm that maintains a single individual and applies mutation to generate new individuals. The PSO algorithm will be used to optimise the size of rooms, while the (1+1) EA will be used to optimise the room position. The framework will involve the application of genetic operators, such as randomly add or minus a small perturbation to the current room size, swap two random rooms, and so on, to generate new floor plans from the existing population. The optimisation process will be repeated for a specified number of generations or until a termination criterion is met.

Algorithm 2 : (1+1) EA for Mutating Room Positions

Require: Boundary, List of Rooms, Maximum Iterations max_iter

```
Initialise placed_rooms by randomly placing each room at a legal position
  within the boundary
Set best_layout ← layout with placed_rooms
Set best_fitness ← fitness of best_layout
for j = 1 to max_iter do
  for each room i in placed_rooms do
    Generate a random number r ∈ [0, 1]
    if r < 0.05 then
      Randomly select a new legal position for room i
      Create a new layout by moving room i to the new position
      Compute fitness of the new layout
      if new fitness > best_fitness then
        Update best_layout and best_fitness
        Update placed_rooms with the new position for room i
      end if
    end if
  end for
end for
return best_layout, best_fitness
```

Later, the PSO algorithm was replaced by the (1+1) EA as well, so that the room sizes (i.e., width and depth) could be optimised in a similar way to the room position (see Algorithm 3). The (1+1) EA algorithm randomly selects a room in the floor plan and generates a new room size by adding or subtracting a small random perturbation to the current size. The new room size is evaluated using the same fitness functions, and if it results in a better fitness value, it replaces the current best room size. This process continues until all rooms have been assigned their optimal sizes.

Algorithm 3 : (1+1) EA for Mutating Room Sizes

```
Require: Boundary, Room Size Ranges, Maximum Iterations max_iter
Randomly generate initial room sizes within the specified ranges
Set best_layout_rooms ← initial room sizes
Set best_fitness ← −∞
for i = 1 to max_iter do
    Copy best_layout_rooms to rooms
    for each room j in rooms do
        Generate a random number r ∈ [0, 1]
        if r < 0.125 then
            Mutate width and depth of room j by adding random integers in
            [−3, 3], keeping within allowed range
        end if
    end for
    Use Position_OnePlusOneEA to optimise positions for current sizes, get
    best_layout, fitness
    if fitness > best_fitness then
        Update best_layout, best_fitness, best_layout_rooms
    end if
    Record best_fitness in history
end for
return best_layout, best_fitness
```

If possible, other optimisation algorithms, such as simulated annealing algorithm or differential evolution, will be explored to further enhance the optimisation process. The final output of the evolutionary optimisation framework will be an optimised residential floor plan with high fitness value that meets the specified design criteria and constraints.

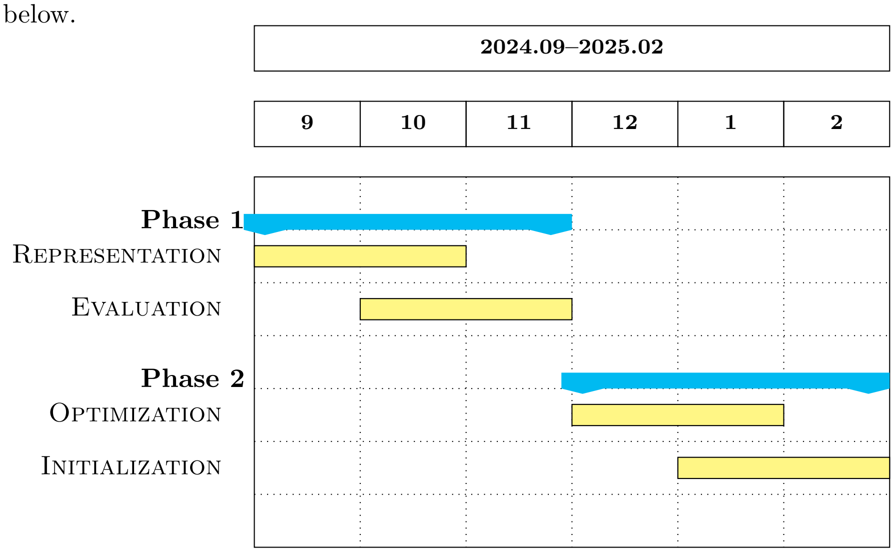
4 Plan vs Progress

4.1 Research Plan

My research plan consists of four main phases through the whole research.

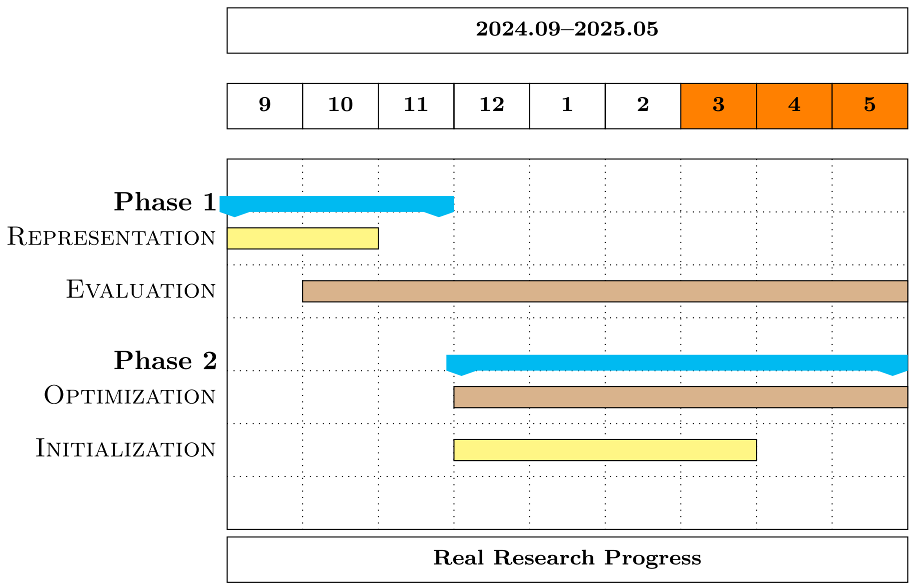
- Phase 1
    1. Solution representation
    2. Solution evaluation
- Phase 2
    1. Population initialisation
    2. Evolutionary optimisation

Each phase involves specific tasks and activities that will be carried out over a period of 6 months (since I did not make a plan for semester 1 of this year). The timeline for the research plan, at that time, is shown in the Gantt chart below.



Last Semester Research Plan Timeline

However, the real research progress did not go as planned. The research plan was adjusted to focus on the solution evaluation and evolutionary optimisation. The solution evaluation is the key to the quality of the generated floor plans, and the evolutionary optimisation is the core of this research. The solution representation is the prerequisite of solution evaluation. The population initialisation can be adjusted according to the progress of the research. The true timeline for the research progress is shown in the Gantt chart below. (Orange means current semester.)



4.2 Progress

It can be seen from the Gantt chart above that the research process has been adjusted to mainly focus on the solution evaluation and evolutionary optimisation.

Last summer holiday, I have completed the solution representation and solution evaluation, respectively using self-defined classes to represent the floor plan and using a set of fitness functions to evaluate each floor plan. All the fitness values are calculated and normalised to a range of 0 to 1.

Fitness functions are designed to evaluate the quality of the generated floor plans based on various criteria, such as privacy, comfort, practicality, and convenience. The evaluation process involves calculating the fitness values for each floor plan in the population and selecting the best individuals for particles updating their velocity and position in PSO.

4.2.1 PSO algorithm + MCTS algorithm

From the start of semester 1 of 2025, I have been working on the evolutionary optimisation process. Initially, I focused on the PSO algorithm, which is suitable for optimising the size of rooms due to parallel processing, and MCTS algorithm, which handles discrete variables (i.e., room position). I had implemented the PSO algorithm (Algorithm 1) and MCTS algorithm (Algorithm 4) in Python, and I tried integrating them into a single framework. The integration process involves combining the two algorithms to create a hybrid optimisation approach that can effectively handle both continuous and discrete variables in the floor plan design process.

Algorithm 4 : Monte Carlo Tree Search (MCTS) for Floor Plan Layout

```
Require: Boundary, List of Rooms, Number of Iterations N
Initialise root node with empty layout state
for i = 1 to N do
    Set node ← root, state ← copy of root state
    {# Selection}
    while node is fully expanded and not terminal do
        node ← child with highest UCB score
        if placing node.action fails then
            break (invalid simulation)
        end if
    end while
    {# Expansion}
    if node has untried actions then
        Randomly select an untried action
        if placing action succeeds then
            Add new child node for this action
            node ← new child
        end if
    end if
    {# Simulation}
    Copy current state to simulation_state
    while not all rooms placed and attempts < max_attempts do
        Randomly select a legal action
        if placing action fails then
            break (invalid simulation)
        end if
    end while
    {# Evaluation}
    if simulation is valid and all rooms placed then
        Compute reward using fitness function
        if reward > best_reward then
            Update best_reward and best_state
        end if
    else
        reward ← 0
    end if
    {# Backpropagation}
    while node is not None do
        node.visits + = 1
        node.value + = reward
        node ← node.parent
    end while
end for
return Layout from best_state or best child of root
```

4.2.2 MCTS algorithm replaced by (1+1) EA

However, the integration process has proven to be more complex than anticipated, and the MCTS algorithm did not optimise the room position very well. It cannot eliminate overlap even after 2000 iterations (see Figure 2) (bold black line means entry).

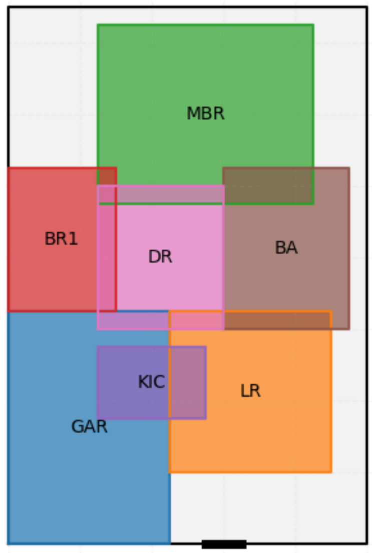


Figure 2: Example of room overlap after 2000 PSO-MCTS iterations. (Bold black line is the entry.)

Therefore, I changed it to a simple greedy algorithm called (1+1) Evolutionary Algorithm, which is a simple evolutionary algorithm that maintains a single individual and applies mutation to generate new individuals. The (1+1) EA algorithm is easier to implement and can be used to optimise the room position (i.e., the (x, y) coordinates) in the floor plan design process. If the new solution has a better fitness value, it replaces the current solution. This process continues until a termination criterion, such as a maximum number of iterations or a satisfactory fitness value, is met. It is a simple yet effective optimisation algorithm that can be used to improve the quality of the generated floor plans, especially in terms of room position.

4.2.3 Swap two random rooms

From this point onwards, swap mutation was adopted to optimise room positions. Swap mutation involves randomly selecting two rooms and exchanging their positions. This operation is simple yet effective, allowing the exploration of a larger design space and the generation of new floor plan solutions. The newly generated floor plan is evaluated using the same fitness functions, and if it achieves a higher fitness value, it replaces the current best solution. The results (see Figure 3) show this can help reduce the overlap rate and improve the overall quality of the floor plans.

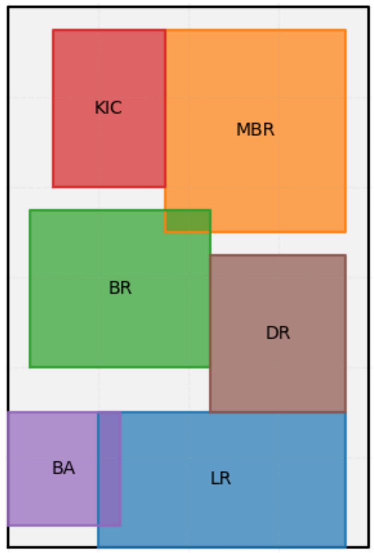


Figure 3: Less room overlap after using swap mutation.

4.2.4 Tune inertia weight

In the PSO algorithm, the inertia weight plays a crucial role in balancing exploration and exploitation. Initially, I implemented a linearly decreasing inertia weight, which reduces the inertia weight from a maximum value to a minimum value over the course of iterations. However, after conducting experiments, I found that a non-linearly decreasing inertia weight performs better in terms of convergence speed and solution quality.

The non-linearly decreasing inertia weight [4] is defined as:

$$w(t) = w_{\max} + (w_{\min} - w_{\max}) \cdot \left(1 - \frac{t}{T}\right)^n$$

where $w(t)$ is the inertia weight at iteration t, $w_{\max}$ and $w_{\min}$ are the maximum and minimum inertia weights, T is the total number of iterations, and t is the current iteration. t is the nonlinear modulation index in the range of [0.9, 1.3].

This non-linear decrease allows the algorithm to explore the search space more effectively in the early stages and focus on exploitation in the later stages. The quadratic term ensures a smoother transition, which helps in avoiding premature convergence and improves the overall performance of the optimisation process.

4.2.5 Prioritized fitness calculation

To improve the performance of the optimisation process, a prioritised fitness calculation strategy has been adopted. This strategy evaluates the fitness of floor plans based on the priority of the criteria, starting with the most important and moving to the less important ones. The prioritised fitness calculation process is outlined as follows:

1. **Define Priorities:** Assign a priority level to each criterion (e.g., privacy, practicality, comfort, convenience). Higher priority criteria are evaluated first. For example, the garage should be on the same side as the entry, and the main bedroom should be larger than other bedrooms. If these criteria are not met, a penalty will be deducted from the final fitness value.

2. **Sequential Evaluation:** Calculate the fitness values for each criterion in the order of their priority. For example, overlap rate is evaluated before utilization rate. This means that if a floor plan fails to meet the zero overlap rate, it will not be evaluated for utilization rate. As can be seen in Figure 4, the floor plan with zero overlap rate is evaluated for utilization rate, while the floor plan with non-zero overlap rate is not evaluated for utilization rate. Later, the privacy criterion is evaluated before the overlap rate optimisation.

3. **Weighted Aggregation:** Combine the rest of fitness values using a weighted sum, where the weights correspond to the significance levels. The overall fitness value is given by:

$$\text{Fitness}_{\text{total}} = \sum_{i=1}^{n} w_i \cdot \text{Fitness}_i$$

   where w_i is the weight for criterion i, and Fitness_i is the fitness value for criterion i.

4. **Early Termination:** If a floor plan fails to meet the minimum threshold for a high-priority criterion, it is discarded without evaluating the lower-priority criteria. This reduces unnecessary computations and speeds up the optimisation process.

This strategy ensures that the optimisation process focuses on the most critical aspects of the floor plan design, improving both efficiency and the quality of the solution generation. It allows for a more efficient evaluation process, since it avoids unnecessary calculations in the early stages.

4.2.6 Change land size and orientation

In the initial version of the code, the land size and orientation were fixed, which limited the flexibility of the design process. To enhance the adaptability of the floor plan generation, I have modified the code to allow for variable land sizes

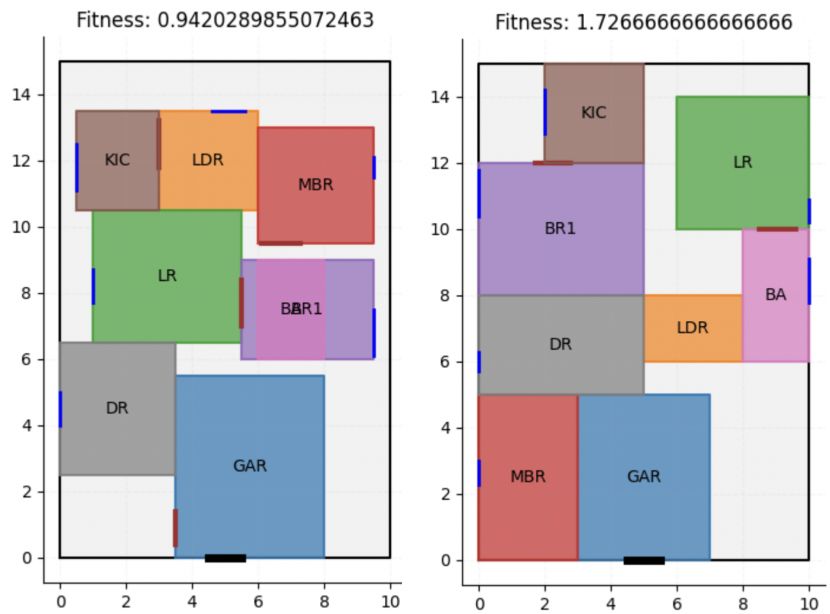


Figure 4: Example of prioritised fitness calculation. The left one is still working on reducing overlap rate. Utilization rate is only calculated for floor plans with zero overlap rate (the right one). (Bold blue lines are windows; bold brown lines are doors)

and orientations. This change enables the optimisation process to spot potential problems and explore a wider range of design possibilities, accommodating different land configurations and orientations. The new approach allows for more diverse and flexible floor plan designs, making it easier to meet the needs and preferences of specific users.

4.2.7 Remove windows and doors

In the previous version of the code, the windows and doors were added to each room immediately after the floor plan was generated no matter whether overlap exists or not. However, this approach led to a significant increase in the complexity of the optimisation process, as the presence of windows and doors added additional constraints to the evaluation. As a result, the optimisation process became slower and less efficient, and the overall quality of the generated floor plans was not satisfactory even after 2000 iterations. To address this issue, I have removed the windows and doors from the early generation process. Instead, the optimisation process will focus on generating the basic layout of the floor plan, including room sizes, positions, and an entry, without any additional features. Once the optimisation process is completed and a satisfactory floor plan has been generated, windows and doors can be added to the floor plan.

This approach simplifies the optimisation process and allows for more efficient exploration of the design space at the early stages.

4.2.8 PSO algorithm replaced by (1+1) EA

After that, we found that the PSO algorithm is not suitable for optimising the size of rooms, because it always gets trapped at a local optimum and can hardly break through it. It also takes a huge amount of time to complete one iteration through all particles. Therefore, I changed it to (1+1) EA too, so that the room size (i.e., width and depth) can be optimised in a similar way to the room position optimisation. The (1+1) EA algorithm will randomly select a room in the floor plan and generate a new room size by adding or deducting a small random perturbation to the current size. The new room size will be evaluated using the fitness functions, and if it results in a better fitness value, it will replace the current best room size. This process continues until all rooms have been assigned their optimal sizes.

4.2.9 Dotted line for open space

In the earlier design phase, the open space in the floor plan was represented by a solid line, which made it difficult to distinguish between open space and closed rooms (e.g., dining room, living room, and kitchen. See Figure 5). To improve the clarity of the design, I have changed the representation of open space to dotted lines. This change allows for a clearer visualisation of the floor plan, making it easier to identify open spaces from other rooms in the house.

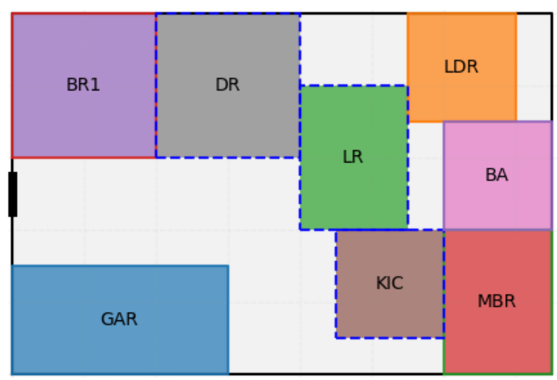


Figure 5: Example of open space surrounded by dotted line.

4.2.10 Tune weights for fitness functions

In the final stages of the optimisation process, I have been tuning the weights of the fitness functions to achieve better results. The weights are used to balance the importance of different criteria in the evaluation process. By adjusting the weights, I can prioritise certain aspects of the design, such as low overlap or high utilization rate, to achieve better overall results. This tuning process is iterative and may require multiple rounds of testing and evaluation to find the optimal balance for the specific design goals.

4.2.11 Filling Blank Areas

In the final stages of the optimisation process, I have been working on filling blank areas in the floor plan. The goal is to improve the overall layout and utilisation of space by expanding rooms into adjacent blank areas. This process involves heuristically expanding public areas first (Algorithm 5), and then closed rooms (Algorithm 6) to four directions (up, down, left, right) until they reach the boundary of the house or another room (see example result in Figure 6). The expansion is done in a way that maintains the overall structure and flow of the floor plan while maximising the use of available space. This approach helps to create a more efficient and functional layout, making better use of the available area.

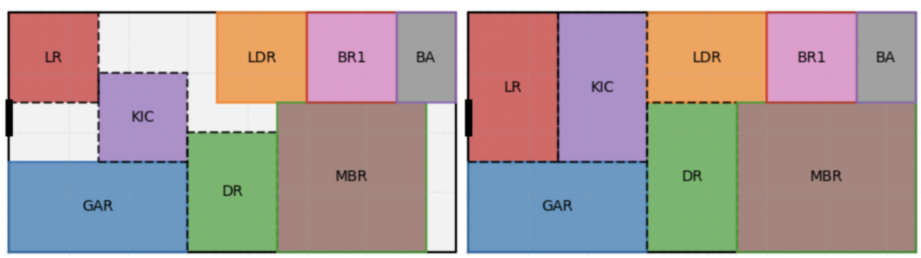


Figure 6: Filling blank areas in the floor plan. The left one is the original final floor plan, and the right one is the floor plan after filling blank areas.

Algorithm 5 : Expand Public Rooms to Fill Blank Areas

Require: House layout with all rooms placed

Define public room types: Kitchen, Living Room, Dining Room
for each room in house **do**
    if room is a public room **then**
        for each direction in [left, right, up, down] **do**
            while room can be expanded by 1 unit in this direction without overlap and still within boundary **do**
                Expand room by 1 unit in this direction
            end while
        end for
    end if
end for

Algorithm 6 : Expand Other Rooms to Fill Blank Areas

```
Require: House layout with all rooms placed
Define public room types: Kitchen, Living Room, Dining Room
for each room in house do
  if room is not a public room then
    for each direction in [left, right, up, down] do
      while room can be expanded by 1 unit in this direction without
      overlap and still within boundary do
        Expand room by 1 unit in this direction
      end while
    end for
  end if
end for
```

4.2.12 Add windows and doors

After filling the blank areas, I added a post-processing step to place windows and doors in each room. Windows and doors are added according to design criteria and constraints, such as their width and position relative to the rooms and the house. This step enhances the aesthetics and functionality of the floor plan by providing natural light and ventilation. An example of the final floor plan with windows and doors is shown in Figure 7.

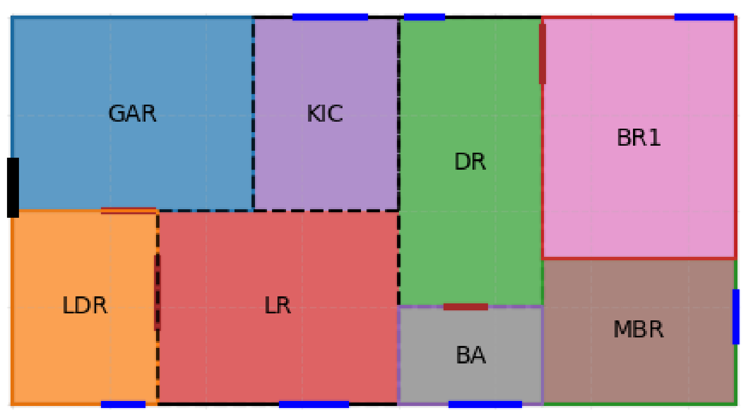


Figure 7: Final floor plan with windows and doors. (Bold blue lines are windows; bold brown lines are doors)

Windows are placed on room sides that face outside, while doors are added to walls between rooms. Both are represented as lines in the floor plan, with properties such as width and position. Importantly, windows and doors are only added after the optimisation process is complete, ensuring they do not interfere with the optimisation of room sizes and positions.

The algorithms for generating windows and doors are shown in Algorithm 7 and Algorithm 8. Window placement is based on exterior-facing walls, and door placement is based on walls between rooms. The width and position of each window and door are randomly determined within allowed ranges, ensuring they fit within the room and meet design constraints.

Algorithm 7 : Generate Windows for Rooms

```
for each room in the house do
    if room is not a garage then
        Initialise empty window list for the room
        Find all sides of the room that face outside
        if there is at least one such side then
            Randomly select one side as the wall for the window
            Randomly determine window width within allowed range and not
            exceed the room size
            Randomly determine window position along the selected wall
            Add window to the room
        end if
    end if
end for
```

Algorithm 8 : Generate Doors for Rooms

```
for each room in the house do
  if room is not kitchen, living room, or dining room then
    Initialise empty door list for the room
    Find all sides of the room that do not face outside and are adjacent to
    public areas
    if there is at least one such side then
      Randomly select one side as the wall for the door
      Randomly determine door width within allowed range and not exceed
      the room size
      Randomly determine door position along the selected wall
      Add door to the room
    else
      for each direction do
        if the side does not face outside then
          Add this direction to options
        end if
      end for
      if options is not empty then
        Randomly select one side from options as the wall for the door
        Randomly determine door width and position as above
        Add door to the room
      end if
    end if
  end if
end for
```

5 Experimental Results

5.1 Run same iterations for 3 Algorithm Combinations

To compare the performance of different algorithm combinations, I ran three different algorithms for 30,000 iterations each. The first algorithm is a dual (1+1) EA approach, which uses two (1+1) EAs to optimise room positions and sizes separately. The second algorithm is a PSO-(1+1) EA combination, where PSO optimises room sizes and (1+1) EA optimises room positions. The third algorithm is a PSO-MCTS combination, where PSO optimises room sizes and MCTS optimises room positions.

From the line chart in Figure 8, it can be seen that the fitness value of the floor plan has been improved significantly after about 2000 iterations using PSO-(1+1) EA and dual (1+1) EA, except PSO-MCTS. When running the PSO-MCTS algorithm, the fitness value gets stuck at 2.7, which means it is struggling on reaching an utilisation rate of 0.8. It cannot break through the local optimum even after 30000 iterations.

The PSO-(1+1) EA algorithm attains a fitness value of 16.26 after 30,000

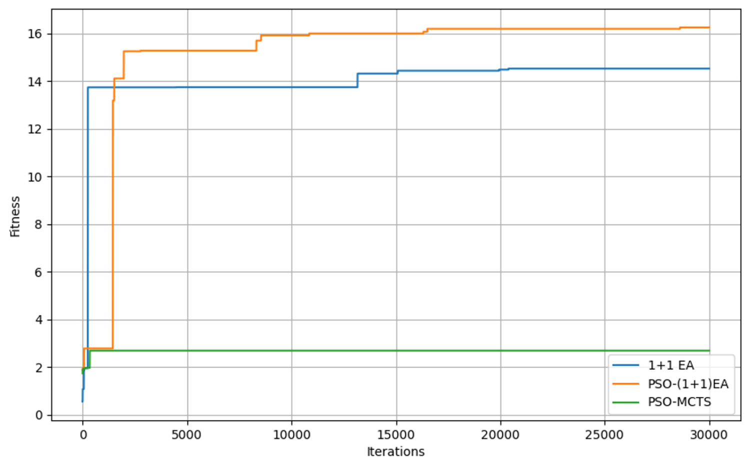


Figure 8: Fitness comparison between 3 Algorithm Combinations.

iterations, whereas the dual (1+1) EA approach surpasses a fitness value of 2.8 in a shorter duration compared to the PSO-based method, and can generate a good-enough floor plan in less than 2000 iterations (Table 2). Consequently, the dual (1+1) EA algorithm is identified as the most effective strategy for floor plan optimisation in this study.

Table 2: Comparison of Execution Time and Best Fitness

Method	**Execution Time**	**Best Fitness**
Dual (1+1) EA	22 min 31 sec	14.53
PSO–(1+1) EA	42 hr 19 min 30 sec	16.26
PSO–MCTS	133 hr 33 min 30 sec	2.7

5.2 Comparison by running same amount of time

To further evaluate the performance and stability of each algorithm, I conducted five independent runs for each of the three algorithm combinations, with each run limited to a maximum execution time of one hour. For each run, the highest fitness value achieved was recorded. Table 3 summarises the maximum, average, and standard deviation (std) of the best fitness values for each algorithm.

To visually compare the quality of floor plans generated by the three algorithm combinations, see Figure 9 in Appendix for representative snapshots of the best solutions generated by each method within the same time limit. These snapshots illustrate the differences in room arrangement, space utilisation, and overall layout quality achieved by each approach.

Table 3: Statistical Comparison of Algorithm Performance (5 runs, 1 hour per run)

Method	Max Fitness	Average Fitness	Std
Dual (1+1)EA	18.5998	17.2614	0.7728
PSO-(1+1)EA	17.8418	14.5881	5.9101
PSO-MCTS	1.9775	1.9530	0.0215

5.3 Same amount of time & Add one bedroom

To further investigate the impact of increased design complexity, I added an extra bedroom (Bedroom2) to the floor plan and repeated the experiments. For each of the three algorithm combinations, I conducted five independent runs, each limited to one hour, and recorded the best fitness value achieved in each run. Table 4 presents the maximum, average, and standard deviation (std) of the best fitness values for each algorithm under this more challenging scenario.

Table 4: Statistical Comparison of Algorithm Performance (5 runs, 1 hour per run, updated results)

Method	Max Fitness	Average Fitness	Std
Dual (1+1)EA	17.6344	13.1540	5.3013
PSO-(1+1)EA	17.7966	5.1424	6.3271
PSO-MCTS	1.8860	1.8544	0.0221

The results indicate that, with the additional bedroom, both (1+1)EA-based methods still outperform PSO-MCTS in terms of maximum and average fitness, but also show greater variability across runs. PSO-MCTS remains the most stable, but its best solutions are consistently lower in quality compared to the other methods. This demonstrates that the (1+1)EA-based approaches are more effective at handling increased design complexity, though their results may fluctuate more between runs.

Figure 10 in the Appendix presents representative snapshots of the best floor plans generated by each algorithm combination after adding an extra bedroom.

6 Conclusion

This thesis presented a comprehensive study on the optimisation of residential floor plans using evolutionary algorithms. The research addressed key aspects of floor plan design, including solution representation, evaluation criteria, and the application of advanced optimisation techniques. While the initial approach combined PSO and MCTS algorithms, experimental results revealed limitations in the effectiveness of MCTS for optimising room positions. Consequently, the methodology shifted towards a (1+1) Evolutionary Algorithm, which proved to be more effective and easier to implement.

Through systematic experimentation and iterative refinement, the dual (1+1) EA approach emerged as the most robust and efficient strategy, consistently achieving high fitness values within reasonable computational time. The findings demonstrate that evolutionary algorithms, particularly the (1+1) EA, are capable of generating high-quality residential floor plans that satisfy a range of design criteria and constraints. This work highlights the potential of evolutionary optimisation as a valuable tool for automating and improving the architectural design process.

7 Future Work

Future work will focus on further enhancing the optimisation process by exploring additional evolutionary algorithms to improve the quality of the generated floor plans and develop a robust framework for real-world applications.

Planned improvements include:

- Explicitly representing corridors in the floor plan to ensure better connectivity and circulation.
- Incorporating corridors, doors, and windows into the evaluation process, so their placement and properties are considered in the fitness calculation.
- Applying other algorithms, such as simulated annealing and differential evolution, to further explore the solution space and escape local optima.

8 Plagiarism Declaration

I hereby declare that this submission is my own work and to the best of my knowledge, it contains no material previously published or written by another person, except where due to acknowledgement is made. Furthermore, I believe that it contains no material which has been accepted for the award of other degree or diploma in any university or other tertiary institutions.

9 References

[1] A. M. Brintrup, H. Takagi, and J. Ramsden. Evaluation of sequential, multi-objective, and parallel interactive genetic algorithms for multi-objective floor plan optimisation. In F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J. H. Moore, J. Romero, G. D. Smith, G. Squillero, and H. Takagi, editors, *Applications of Evolutionary Computing*, pages 586–598, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[2] B. Chaichi and A. Andaji Garmaroodi. Multi-objective optimization of a residential zone by proposing appropriate comfort factors using none dominated sorting genetic algorithm. *Journal of Building Engineering*, 86:108842, 2024.

[3] A. de Almeida, B. Taborda, F. Santos, K. Kwiecinski, and S. Eloy. A genetic algorithm application for automatic layout design of modular residential homes. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 002774–002778, 2016.

[4] E. H. Houssein, A. G. Gad, K. Hussain, and P. N. Suganthan. Major advances in particle swarm optimization: Theory, analysis, and application. *Swarm and evolutionary computation*, 63:100868–, 2021.

[5] S. Kang and S. K. Kim. Floor plan optimization for indoor environment based on multimodal data. *The Journal of supercomputing*, 78(2):2724–2743, 2022.

[6] J. Kennedy and R. Eberhart. A discrete binary version of the particle swarm algorithm. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 5, pages 4104–4108 vol.5. IEEE, 1997.

[7] G. Laignel, N. Pozin, X. Geffrier, L. Delevaux, F. Brun, and B. Dolla. Floor plan generation through a mixed constraint programming-genetic optimization approach. *Automation in construction*, 123:103491–, 2021.

[8] M. Nisztuk and P. B. Myszkowski. Hybrid evolutionary algorithm applied to automated floor plan generation. *International Journal of Architectural Computing*, 17(3):260–283, 2019.

[9] J. Ploennigs and M. Berger. Automating computational design with generative ai. *Civil engineering design*, 6(2):41–52, 2024.

[10] Y. Shi, K. Xue, L. Song, and C. Qian. Macro placement by wire-mask-guided black-box optimization. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

[11] T. Singha, H. Dutta, and M. De. Optimization of floor-planning using genetic algorithm. *Procedia technology*, 4:825–829, 2012.

[12] R. Subramanian, T. DheenaDayalan, T. Badhrirajan, C. Dhinakaran, C. Devakirubai, and R. R. Sudharsan. An automated house plan generator leveraging genetic algorithms. In *2021 International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)*, pages 1–6, 2021.

[13] L. Wang. A hybrid genetic algorithm–neural network strategy for simulation optimization. *Applied Mathematics and Computation*, 170(2):1329–1343, 2005.

[14] T.-K. Wang and W. Duan. Generative design of floor plans of multi-unit residential buildings based on consumer satisfaction and energy performance. *Developments in the Built Environment*, 16:100238, 2023.

[15] S. Yan and N. Liu. Computational design of residential units' floor layout: A heuristic algorithm. *Journal of Building Engineering*, 96:110546, 2024.

[16] M. Zawidzki and J. Szklarski. Multi-objective optimization of the floor plan of a single story family house considering position and orientation. *Advances in engineering software (1992)*, 141:102766–, 2020.

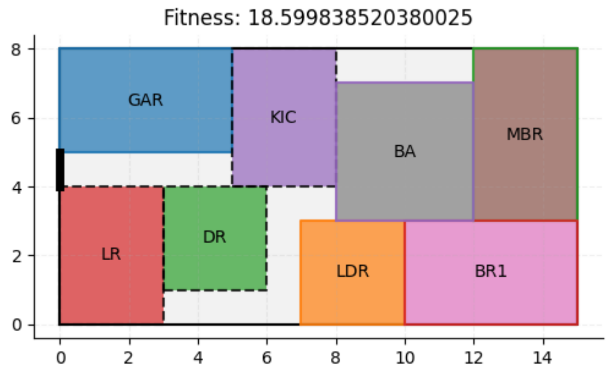
[17] D. Zou, G.-G. Wang, A. K. Sangaiah, and X. Kong. A memory-based simulated annealing algorithm and a new auxiliary function for the fixed-outline floorplanning with soft blocks. *Journal of ambient intelligence and humanized computing*, 15(2):1613–1624, 2024.

A Appendix

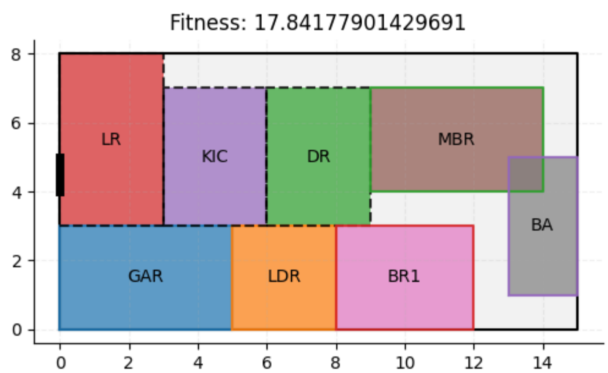
A.1 Code

Our code is available on GitHub: https://github.com/ZuxingGit/EvolutionaryOptimisation/blob/main/code/3_optimization/run_pso_mcts.py

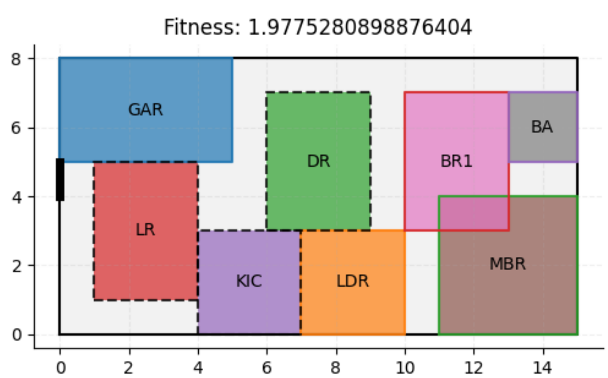
A.2 Best Floor Plan Snapshots



(a) Dual (1+1) EA

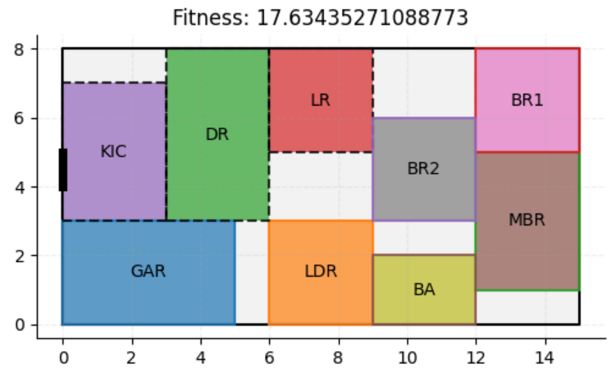


(b) PSO–(1+1) EA

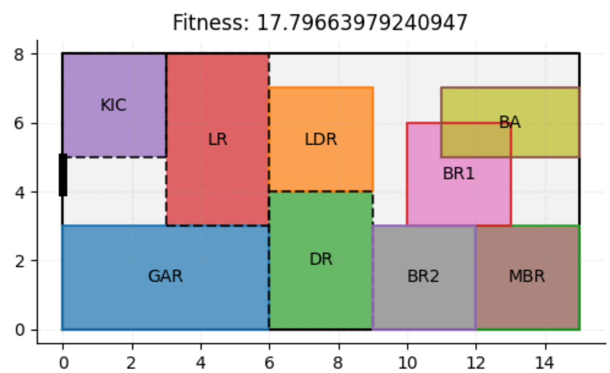


(c) PSO–MCTS

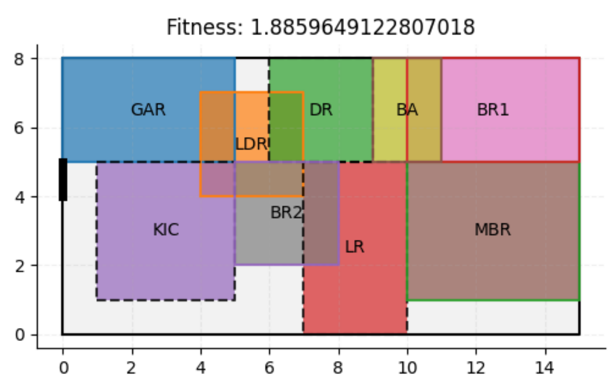
Figure 9: Best floor plan snapshots generated by each algorithm combination.



(a) Dual (1+1) EA with extra bedroom



(b) PSO–(1+1) EA with extra bedroom



(c) PSO–MCTS with extra bedroom

Figure 10: Best floor plan snapshots generated by each algorithm combination after adding Bedroom2.