

Plausible Layout Generation Using Machine Learning, Evolutionary Optimisation and Parametric Methods

Daiva Marcinkeviciute and Wassim Jabi

1 Introduction

Before an architectural competition is announced it is common practice to create a room program with required sizes, adjacencies and travel distances between different spaces, as well as multiple other requirements (for example, daylight). Then test planning is performed—a preliminary design of the building with a goal to make sure the proposed building program is feasible. The test design is also used to calculate and revise the anticipated building costs. The project is however not meant to be built; the actual concept of the building is proposed at the competition.

Test planning is a repetitive, time-consuming task, especially difficult when the topologic relationships of the rooms are complex. At the same time, it does not require high-quality architectural design or creativity. Thus, performing it automatically would liberate the time of architects to work on real architectural projects that are going to be built. While the design of built objects must consider the complex space syntax and aesthetical principles, the test planning can be performed based on simplified rules. This makes test planning an excellent candidate for automatic building layout generation.

The research on generative design and layout generation is still relatively young. There were many attempts to generate layouts using multiple different methods, but they were mostly focused on single-family homes, that have only several rooms and a few simple topologic relationships. The layout generation of more complex, especially multi-storey buildings still has many unresolved issues, especially when

it comes to multi-level circulation. Therefore, further research is needed before more complicated buildings can be efficiently generated in daily practice.

The existing layout generation tools perform in a rather limited way, by either randomly placing the rooms into a given building envelope or subdividing the envelope based on a tree diagram. However, they do not produce efficient logical circulation. An architect on the other hand does not randomly place the rooms but works based on topological principles, first drafting the overall shape and structure of the building, and then laying out the rooms. This approach could be incorporated into a new layout generation method, based on architectural knowledge. It could help to generate more plausible buildings with functioning vertical connections and circulation.

As layout generation is a very broad area, considering the limited time, the scope in this paper is reduced to one building type: school buildings. They were selected because they typically feature a combination of big and small rooms that is difficult to standardise. Plus schools often have a relatively complex but still manageable topological structure. This paper is focused on only generating the ground floor, but the algorithm is designed in a way, that it can be expanded to generate multi-storey buildings (ensuring vertical connectivity and statical plausibility). The algorithm is integrated into the Rhino/Grasshopper environment, as it allows high level of interoperability with other tools for ML, simulation and generative design.

The following research objectives were defined: investigate existing layouts and break them down into planning steps and topological principles; pack these principles into an algorithm that generates school layouts; generate layouts based on a prewritten room program; connect the generative algorithm with an evolutionary solver and optimise the layouts using topologic requirements as optimisation criteria; and, finally, validate the quality of the newly generated layouts according to the requirements in the room program.

2 Related Work

In this section, a survey of the related work of other researchers will be presented.

The automatic layout generation was covered in multiple research papers with a large range of proposed methods. In 2021, Schwartz subdivided the current building generation techniques into three categories: top-down, bottom-up and advanced computational frameworks.

In the top-down approach, a building envelope is created and subdivided into spaces. In the bottom-up approach, the spaces are placed according to their connectivity requirements and allocation is performed starting at room level and going to the building level (Schwartz et al., 2021).

Advanced computational frameworks can combine multiple methods, such as optimisation algorithms and agent-based systems, using k-d tree data and ML (Schwartz et al., 2021).

In the current research, several ML methods are being used for layout generation. The most used method in the current research is Generative Adversarial Network (GAN) (Chen et al., 2020). Nauata proposed a house layout generator, built upon a relational graph-constrained generative adversarial network called House-GAN (Nauata et al., 2003). Para proposed another graph-constrained GAN-based model for layout generation that enables novel capabilities for conditional layout generation (Para et al., 2011). Chaillou proposed an approach, based on stacking models. Using GAN, that first generates a building outline on an empty lot, then generates a room layout inside this boundary and finally places the furniture within the rooms (Chaillou, 2020). The GAN method was combined with graph convolutional network in the research of Chen. His algorithm first translates the verbal description of the house to a structural graph representation and then predicts the layout of rooms with a graph conditioned layout prediction network and generates the interior texture with a language conditioned texture GAN (Chen et al., 2020). All these papers focused on generating layouts for one-storey single-family homes.

Lately, the GAN efforts have been focused on site layout generation. Tian used Pix2Pix GAN to generate an urban fabric by colour-coding building programs and using existing built environment GIS data. To achieve better results, the training data was filtered by different constraints, such as lot shape, orientation, density and purpose of buildings (Tian, 2020). Yao applied a multi-step approach, integrating GAN with a genetic algorithm to generate the layout of residential areas. He used GAN for the layout generation and the genetic algorithm to define the building heights (Yao et al., 2021). Liu used GAN to generate campus layouts, given lot boundary and surrounding streets. The layouts were colour coded by function. He also applied two steps of generation: first generating rough zoning and then using the results to generate a detailed layout (Liu, 2020). In 2021, Zhao used GAN in combination with genetic algorithms to generate hospital layouts, also using multiple steps of generation (Zhao et al., 2021).

Another ML method using Graph Neural Network (GNN) for layout generation was proposed by Huin in the research paper called Graph2Plan. The solution allows the user to input the room count, location and required adjacencies. The training set is then filtered by the user input and selected graphs are fitted into a user-defined building boundary, to find the best match. Only good matches are used to train the graph neural network, which converts the graph into a raster image of the floor plan (Hu et al., 2004).

ML was also used to pick the best match of the apartment layout from a gallery, which was then placed into an available space within a building. The gallery was created by standardising a range of existing apartment layouts. An ML algorithm chose the best match based on the given proportions, shape, entry location and interior and exterior walls. The chosen layout was then reconstructed and adapted to better match the given space (Green, 2020).

In conclusion, while the GAN method has been very popular lately, so far, it still faces many issues. First, it produces low-resolution pixel images, that often are blurry and cannot be vectorised. The generation results are also frequently unrealistic (Schwartz et al., 2021). The main issue with GAN though is not allowing the user

to input a room program or any requirements. The only way to influence the output is by filtering the training data. The research has shown that selecting appropriate data (for example, buildings that are similar in size, use, topological principles, etc.) is essential to achieve good results with ML. Also, better results were achieved by subdividing the generation process into 2–3 steps, where first the approximate zoning was generated and then the generated image was detailed in further rounds of GAN generation. Using ML to select existing layout parts from a library and fit them into the available space is a more basic approach that has shown good results.

Another area of advanced computational frameworks is evolutionary computation, which according to the research of Ekici, is the most used method for optimisation of generative design. He has carried out an overview of over 100 papers, where genetic algorithms were used for optimisation of building shape and layouts. He found that evolutionary solvers were used to optimise mostly sustainability criteria, quite often costs and in a few cases functional and structural goals (Ekici et al., 2019).

Most of the reviewed papers share several research gaps: The attempts to generate layouts were mostly focused on single-family homes. There were few attempts to generate bigger, more complex buildings, that would implement a complex room program with multiple topological constraints. One problem that still must be solved before more complex buildings can be generated is circulation. Circulation needs to be solved on multiple levels: primary circulation with building entrances, horizontal and vertical connections, as well as secondary circulation that connects all the rooms.

3 Research Design and Methodology

To develop the layout generation algorithm the Rhino–Grasshopper environment was chosen, as it offers a high level of interoperability with many different tools for ML, evolutionary algorithms and other simulations.

An algorithm is written to perform the following steps (see Fig. 1):

1. The user inputs a closed curve, which marks the approximate building shape.
2. A matching building layout with primary circulation and spaces that will hold rooms is chosen from a gallery and scaled, rotated and mirrored to match the sketch of the user in the closest way possible. The layout is selected using ML.
3. The user chooses a csv file containing a room program. The rooms and secondary circulation are placed according to the room program. They are placed parametrically based on a set of rules.
4. The layout is evaluated and optimised according to room connectivity and daylight. The room placement is optimised using an evolutionary solver.

To create a simplified user interface in Rhino, the Grasshopper Plugin ‘Human UI’ is used (see Fig. 2).

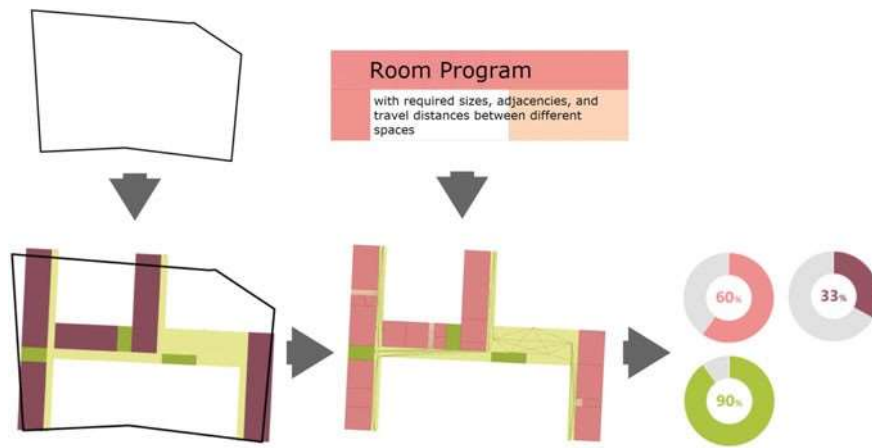


Fig. 1 Layout generation steps

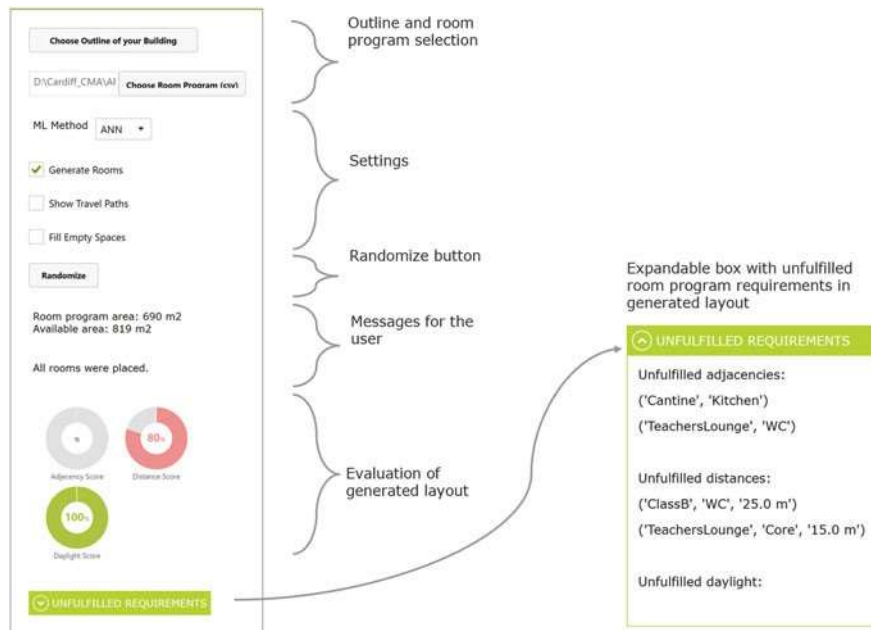


Fig. 2 User interface

4 Placing the Building Shape and Primary Circulation

First, a gallery of existing simplified school layouts was created. Multiple school layouts were selected from literature based on the following criteria: the school must have multiple stories (to have both vertical and horizontal circulation), the total area of the rooms on the ground floor (except circulation) must be between 350 and 1000 m², the school must be plausible for the cold climate zones (circulation inside). It cannot have an open floor plan and it must be possible to break it down into clearly defined areas.

The selected layouts were broken down into the main components: building shape, entrances, main horizontal circulation, vertical circulation and spaces holding the remainder of the rooms (all zones are sketched respecting the main load bearing walls of the building). Once the layouts are simplified into components, they are varied geometrically, but maintained topologically to obtain more options of the building shape, all sharing similar circulation logic and proportions (see Fig. 3).

A closed curve is drawn around the building to get its simplified shape, that will be used in ML to find the best-fitting layout (see Fig. 4).

The simplified outline of the layout is first manipulated by slightly tweaking and stretching it into different directions, as well as mirroring it. The goal of this is to obtain multiple closely similar shapes representing the same layout to increase the training set for ML. Then the shapes are described in a vector of numbers that can

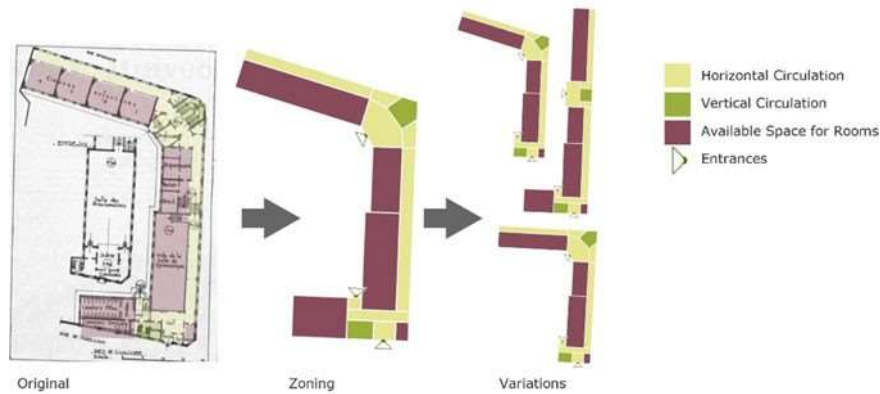


Fig. 3 Creating schematic layouts

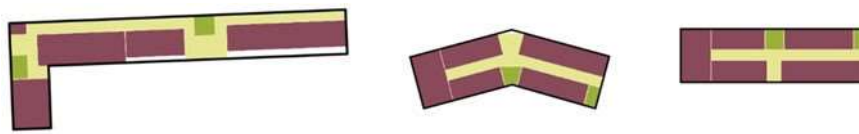
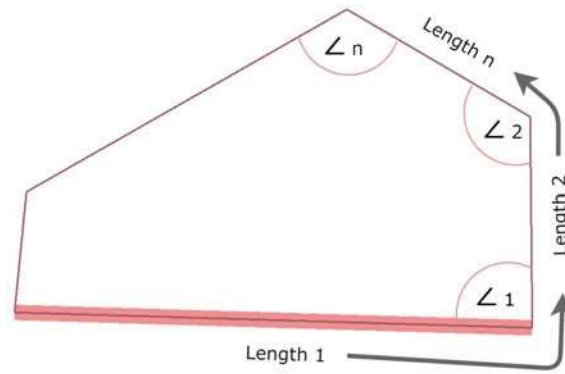


Fig. 4 Sketching the shape of the layout

Fig. 5 Shape example



be interpreted by ML. As the shapes must be recognised also when they are rotated or mirrored, first the longest line is found, and the description of the shape always starts there (i.e. the longest line is always the first line in the vector). The vector is put together as follows: area/perimeter ratio raised by the power of 3; length of the first line; angle between first and second line (in radians); length of the second line; angle between second and third line (in radians); and so on until all the lengths of sides and angles of the shape are in the vector. Then the lengths are unitised (mapped into values between 0 and 1).

As all the vectors must have the same lengths, if a shape has less than eight sides, the rest of the vector is filled with 0 values.

For example, the vector of a five-sided shape can be [2.81, 1, 1.49, 0.42, 1.77, 0.49, 1.92, 0.75, 1.79, 0.29, 1.48, 0, 0, 0, 0, 0, 0] (see Fig. 5).

The collection of vectors is used in the ML software as the training set. The shape sketched by the user is also described with a vector and is used as input.

The scale of layouts is limited, as the circulation can lose its logic at higher scale values. Thus, the algorithm allows scaling layouts by a maximum of 10%. Therefore, only the layouts of a similar size to the input can be considered.

Before the shapes are fed into ML, they are filtered by size (see Fig. 6).

The neural network chooses the most similar shape from the filtered shapes. Then the chosen shape is rotated, mirrored and scaled (by max 10%) to fit closely into the input shape.

5 Machine Learning Experiments

To find the best ML approach, different techniques were tested.

The vector that describes the shapes was assembled in four different ways:

- By area/perimeter ratio, the lengths of the line and angle (see Fig. 5):
 - Unitised (all lengths mapped into the range from zero to one).

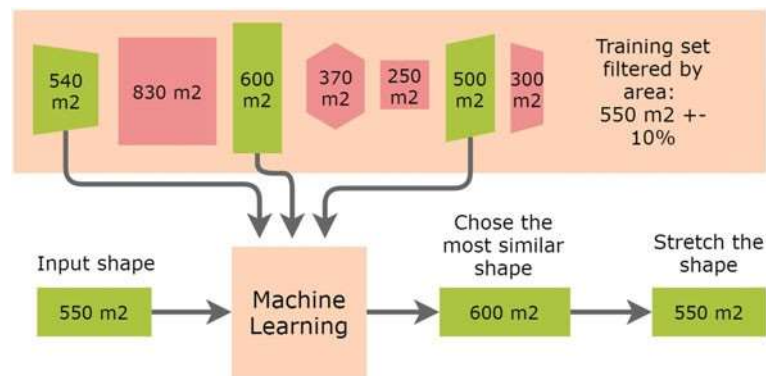


Fig. 6 Filtering and selecting the shape

- Non-unitised (original lengths used in the vector).
- By dividing the curve into thirty equally spaced points and measuring the distances from each point to the centroid. Thirty points were sufficient to represent all the forms in a recognisable manner and did not overload the algorithm with superfluous data. These distances then were put into a vector with thirty values:
 - Unitised.
 - Non-unitised.

To test the efficiency of these different approaches, first, all the training data was split into two groups: 90% for training and 10% for testing. It was split in a way that every label was present in both groups. The data in both groups was shuffled. Then the Artificial Neural Network (ANN) by Lunchbox was trained with the bigger dataset, and the smaller dataset was used as an input. The output labels were compared with the actual ones. In all the cases, the artificial neural network guessed correctly in ~85% of the cases.

As the output data in this research is categorical (layout number), both artificial neural networks and decision trees can give good results. Thus, two different approaches were set up in the Grasshopper definition and trained: the artificial neural network and a custom decision tree, programmed in Visual Studio Code and adapted to run on a parallel server and work in Grasshopper through the Hops component, based on two tutorials (Steve Baer, 2021; W3Schools, 2021).

To compare the results of the ANN and the decision tree, two different methods were applied. First, all the training data was split into two groups: 90% for training and 10% for testing. It was split in a way that every label was present in both groups. The data in both groups was shuffled. Then the ANN and the decision tree were trained with the bigger dataset, and the smaller dataset was used as an input. The output labels were compared with the actual ones. While the ANN guessed correctly in ~85% of cases, the decision tree guessed correctly in ~95% of cases, showing superior results.

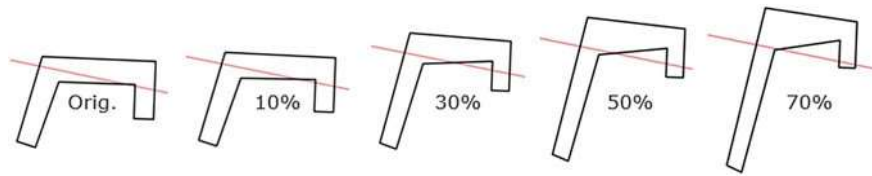


Fig. 7 Deformation of the shapes

In the next step, six different layout outlines were taken from the gallery and gradually transformed. All the shapes were tapered along the same axis by 10, 30, 50 and 70%. They were scaled to keep the area of the original shape (Fig. 7).

On each step, the transformed outline was fed into the ANN and the decision tree as an input and the output result was observed (see Figs. 8, 9, 10, 11, 12 and 13).

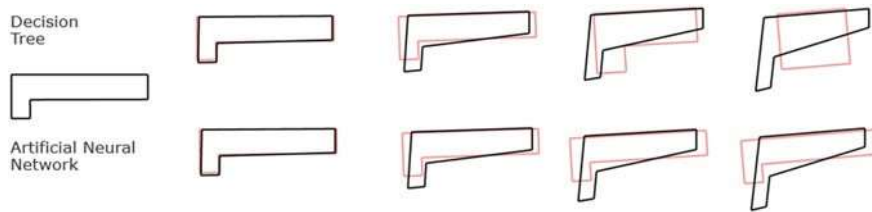


Fig. 8 Layout recognition. Shape 1

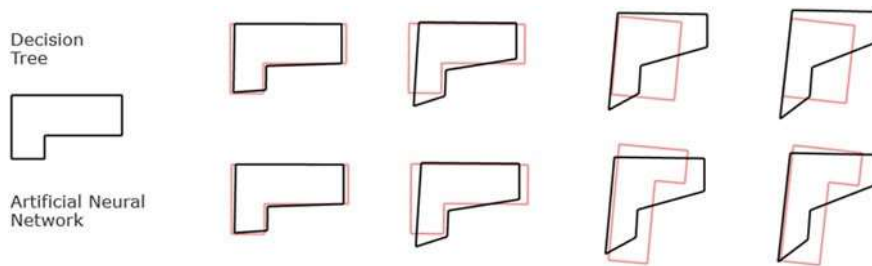


Fig. 9 Layout recognition. Shape 2



Fig. 10 Layout recognition. Shape 3

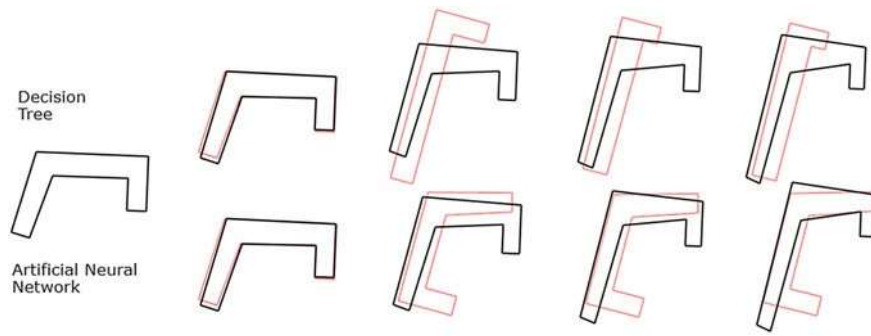


Fig. 11 Layout recognition. Shape 4

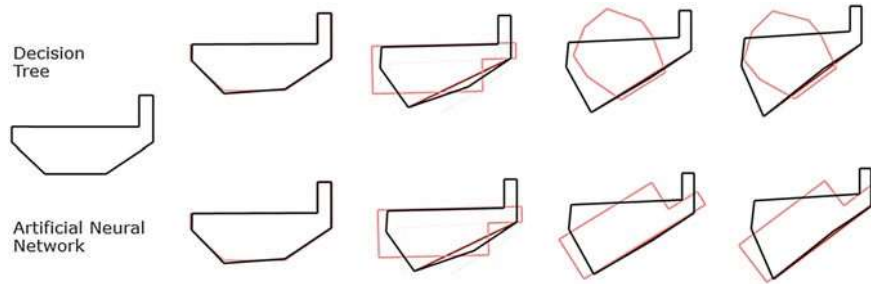


Fig. 12 Layout recognition. Shape 5

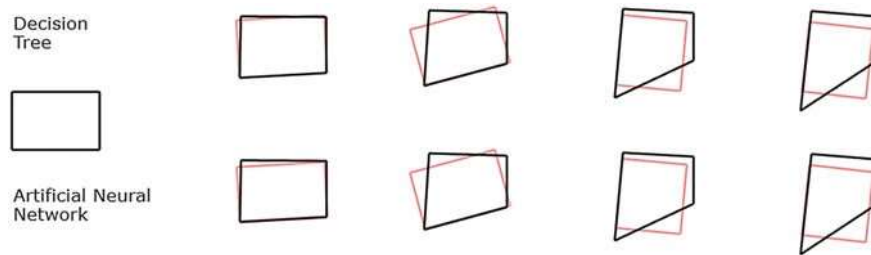


Fig. 13 Layout recognition. Shape 6

The following visual observations were made: As seen in examples 1, 2, 3 and 5, once the outline is transformed by a large factor (70%), the decision tree chose a less similar shape than the ANN. As seen in examples 3 and 6, when the outline is transformed by a small factor (10%), the decision tree recognises the original layout; meanwhile, the ANN interprets it as a different, but similar layout. In conclusion, the decision tree performed better at recognising closely similar shapes; meanwhile, ANN performed better at interpreting shapes that are significantly different from the original.

6 Placing Secondary Circulation and the Rooms

Once the main components of the layout are fitted, the rooms are placed according to the imported room program in csv format (see Fig. 14). The following information for room placement is defined in the program: the number of rooms, the size, the minimum width and the height of the room. The program also defines the following goals for optimisation: the required daylight coefficient (the area of windows divided by the floor area), the required adjacencies (marked in the adjacency matrix as 0) and the required distances (marked in the adjacency matrix as numbers higher than 0). Required distances can be set between the rooms, or from a room to the core and/or entrance of the building.

A set of prewritten rules (see Figs. 15 and 16) define how a room should be placed, depending on its size and proportions. Rooms are either distributed along the main circulation axis or they are placed along secondary circulation, either by putting a corridor in the middle and arranging the rooms on both sides or by placing a room at the end of the block and connecting it with a corridor to the main circulation. To place a more varied combination of bigger and smaller rooms, these rules can be applied recursively.

Once all the rooms are placed and depending on their size and position, the leftover spaces are either turned into a 'generic' room, included in an adjacent room, or turned into a circulation space (see Fig. 17).

While both the available spaces and the rooms must be orthogonal for the algorithm to work, the main circulation can come in any shape, which allows non-orthogonal buildings.

To calculate the travel distances between the rooms, an algorithm first finds the middle point in a wall shared by each room and a corridor and places a dot that represents a door. All these points are connected into a visibility graph representing the travel paths. Only the paths within the circulation system are considered, when calculating the travel distances.

Once all the rooms are placed, the Opossum evolutionary solver evaluates and optimises the criteria defined in the room program: adjacencies (rooms are considered adjacent, if they share a wall that is large enough to fit a door), travel distances (distances are calculated from door to door via circulation spaces) and daylight (the ratio between exterior wall area and floor area). Each layout receives a score for each of the three criteria (0–100%).

Room	Number	Size	Length	height	Daylight_Q	Entrance	Core	ClassB	ClassS	Cantine	Kitchen	WC	GameRoom	Teacher's
ClassB	3	60	11	3	0.2	n	n	n	n	n	n	15	n	n
ClassS	4	40	8	3	0.2	n	n	n	n	n	n	15	n	n
Cantine	1	120	6.5	3	0.1	n	10	n	n	n	0	n	n	n
Kitchen	1	50	4.5	3	0	n	n	n	n	n	n	n	n	n
WC	4	10	4	3	0	n	n	n	n	n	n	n	n	n
MediaRoom	1	70	12	3	0	n	n	n	n	n	n	0	n	n
Teacher'sLounge	1	70	5	3	0.1	20	10	n	n	n	n	0	n	n

Fig. 14 Room program example

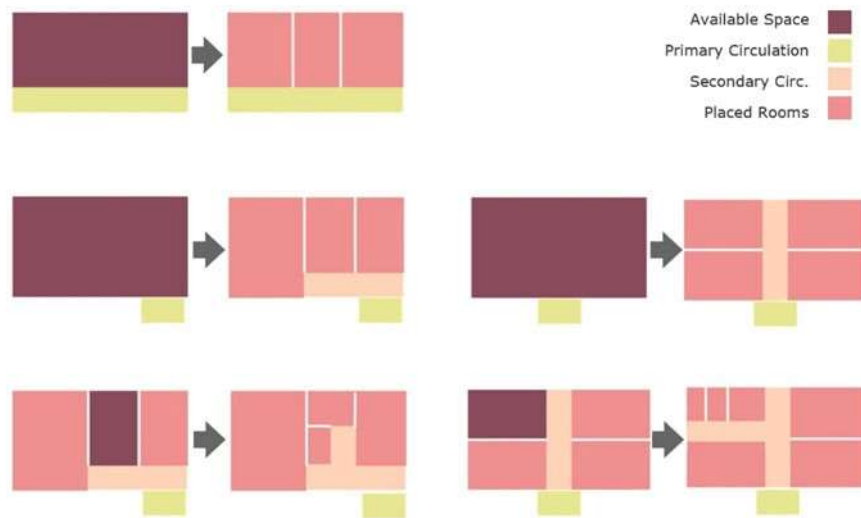


Fig. 15 Concept of placing rooms and secondary circulation

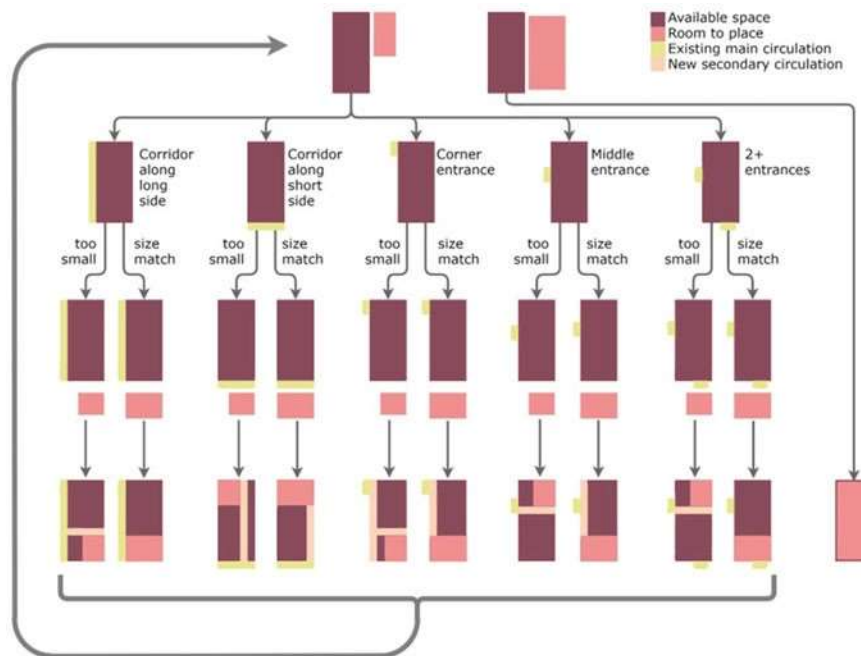


Fig. 16 Rules for placing rooms and secondary circulation

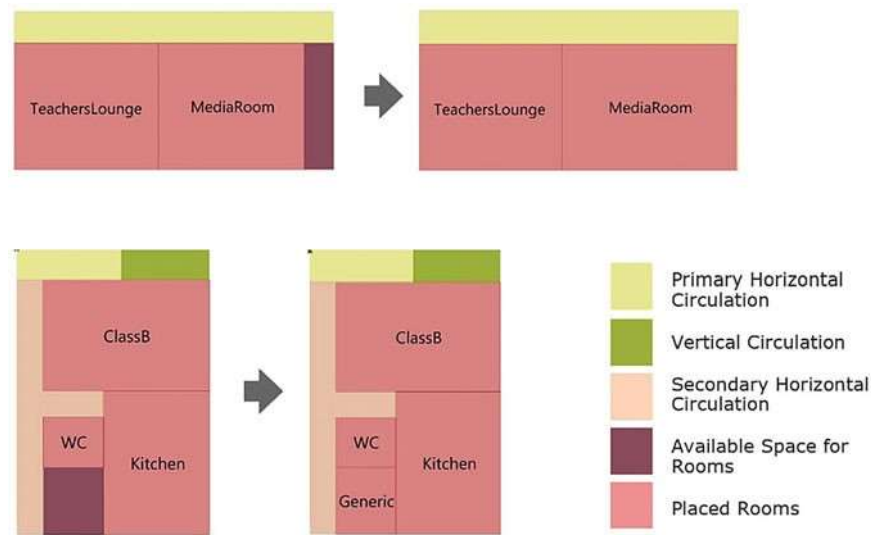


Fig. 17 Filling empty spaces

7 Results

The result is a schematic 2D layout of the ground floor of the school, where the rooms are represented as rectangles. The layout contains all the rooms from the room program. The movement paths within the building are represented with lines. The evaluation of the space connectivity and daylight criteria is calculated and visualised.

Multiple layouts were generated using a room program with the following requirements: Daylight Coefficient: 0.2 for classrooms, 0.1 for the canteen and teachers' lounge. Adjacencies: canteen adjacent to kitchen; teachers' lounge and media room adjacent to a WC. Distances: a WC within 15 m from each classroom, a core within 10 m of the canteen, teachers' lounge within 10 m from core and 20 m from entrance.

As the required distances are very short, the evolutionary solver could not optimise all the goals to 100% (see Figs. 18 and 19).

The same room program was changed by increasing the required distances, to make the goals more achievable. While daylight requirements stayed the same, the media room was no longer required to be adjacent to a WC and the distances between the classrooms and WCs as well as between the canteen and the core was increased to 25 m; meanwhile, the distance between the teachers' lounge and the core was increased to 15 m.

This time the evolutionary optimisation achieved 100% of the goals within 5 min (see Fig. 20).

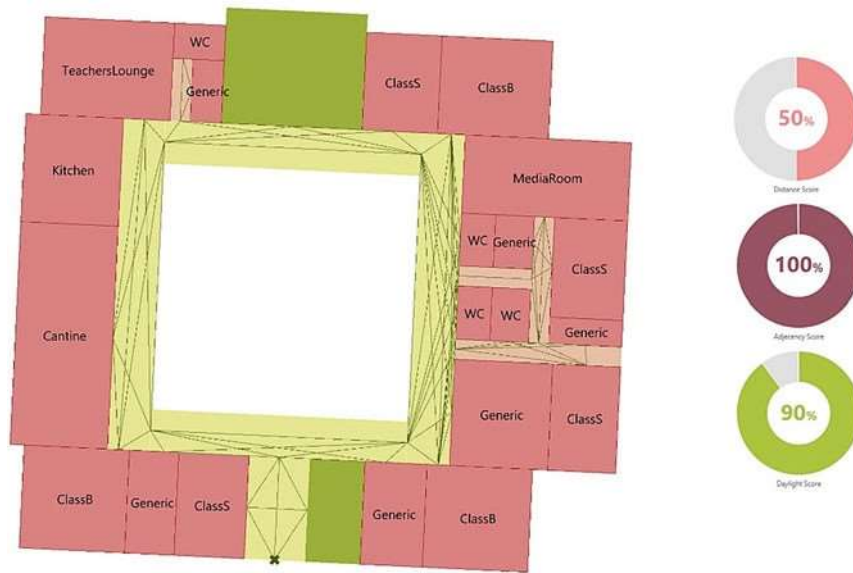


Fig. 18 Optimised layout (5 min, 87 iterations)

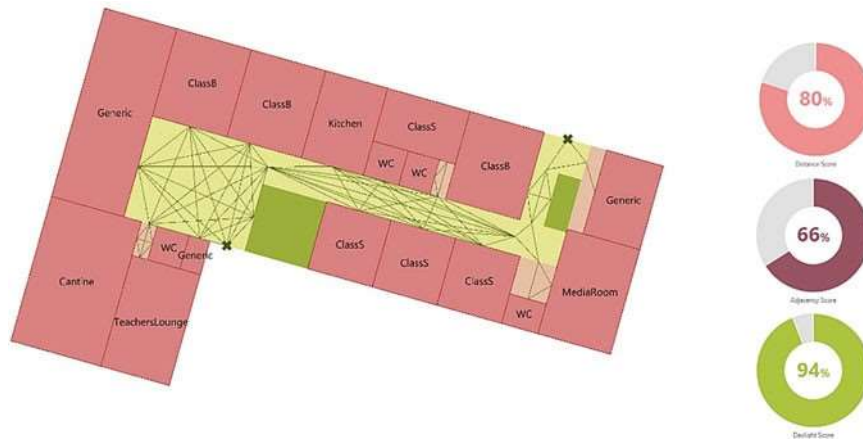


Fig. 19 Optimised layout (5 min, 102 iterations)

8 Next Steps and Improvements

The algorithm can be expanded by copying the schematic layout from the gallery vertically to create a multi-storey building. This way the vertical circulation as well as the façade and main load bearing walls would overlap vertically. Possibly the

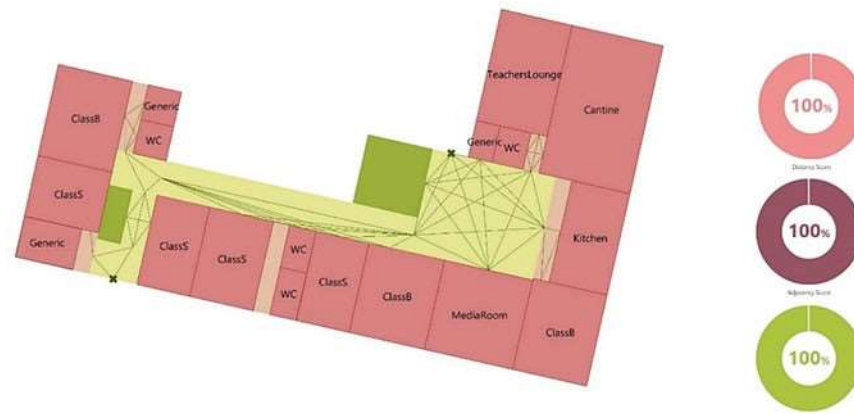


Fig. 20 Optimised layout (5 min, 72 iterations)

gallery could contain the schematic layout of the ground and the upper floor and use the upper floor zoning for higher levels. Then the rooms from the program could be distributed on all the levels according to the parametric workflow described in this paper. The algorithm checking the travel distances between the rooms would have to be adapted for a multi-storey building to include also vertical connections. This way multi-storey buildings could be generated and evaluated.

Laying out the building shape and the main zones could also be done parametrically based on a set of rules, like the ones, used to lay out the rooms. Different rules could be defined for different categories of schools, according to their shape and topologic relationships. This would allow more flexibility and more diverse results. Also, more functional zoning could be added (for example separating the loud and quiet zones of the school or formal and informal learning areas).

At the moment room placing algorithm only places orthogonal rooms into orthogonal spaces. It could be improved to handle non-orthogonal shapes.

By adding the height to the room objects a topologic cell complex representing the building can be created. This would allow multiple further simulations, such as thermal and structural performance and access to views. The resulting conceptual model could also be used to generate more detailed building elements, using vertices, edges, faces and cells as references. The vertices of the visibility graph could be used to generate doors.

9 Conclusions

A new method for layout generation was created which combines computational methods and architectural patterns learned from existing buildings. It addresses complex buildings with hierarchical circulation. The school layouts were chosen

and broken down into planning steps based on topological principles. A two-rounds approach was applied, where the zoning and main circulation are placed first and then the rooms and secondary circulation are laid out next. The steps were combined into an overall algorithm. This resulted in plausible layouts that could be generated from a room program where all the rooms are accessible via a hierarchical and functional circulation system. Confirming precedents from the literature, the two-rounds approach has proven to deliver good results.

For placing the zoning, a gallery of zoning layouts was used. ML helped select the best-fitting layout from a gallery of precedent layouts. Two ML approaches were tested: an ANN and a decision tree. Both approaches have shown good results: while the ANN was better at interpreting new shapes that are significantly different from the training set, the decision tree was better at interpreting shapes that are more closely similar to the input shape. A layout gallery, however, takes a lot of time to assemble and limits the amount of possible building shapes. Thus, placing the zoning could also be broken down into algorithmic steps (just as the room placement was broken down in this project), and generated parametrically for any given building shape.

Room placement was improved using evolutionary optimisation based on the topologic requirements defined in the room program. Three criteria (adjacencies, distances and daylight) were optimised, but the number of criteria to optimise is flexible and can be expanded according to the needs of the user. The success of maximising the optimisation criteria was dependent on the strictness of criteria (the easier criteria are set up in the room program, the easier they all can be fulfilled). While it took many iterations to achieve good results (50–200), the process is fast and automatic, and is still incomparably more efficient than sketching and evaluating different options manually (minutes versus weeks). The optimisation success also depended on the schematic layout of the gallery. The same room program could be successfully optimised in one schematic layout but can fail in another one.

While the current workflow is set up to only generate the ground floor, it can be expanded to generate multi-storey buildings.

To sum up, the algorithm can be used for test planning. It can check the plausibility of a room program on different building shapes within minutes. This not only frees architects to work on projects that are going to be built but also allows them to change and optimise the room program more easily, as it can be checked much faster than before.

The research brought together architectural knowledge and computational methods, like ML and evolutionary simulation. It showed, that extracting patterns found in existing architecture and incorporating them into the workflow can significantly improve plausibility of the generated results (in comparison to examples in the literature review). This approach based on combining human and artificial intelligence could uncover new opportunities for automatic building generation.

10 Related Software and Tools

Layout generation tools (Rhino–Grasshopper environment):

- Yconst, PackRat. 2018.
- Tabari, M., Termite Nest 2021.
- Vestartas, P., OpenNest. 2021.
- hellguz, Magnetizing Floor Plan Generator. 2020.

Other layout generation tools:

- MINC, Finch. 2021. (in development)
- TestFit Inc, Testfit. 2021
- Matterlab, Refinery Toolkit. 2020. (Dynamo environment)

Machine learning tools (Rhino–Grasshopper environment):

- Miller, N., LunchBox for Grasshopper. 2020.
- Zwierzycki, M., Owl. 2019.
- Greco, L., Dodo. 2019.
- Felbrich, B., Crow—Artificial Neural Networks. 2020.

Evolutionary solvers (Rhino–Grasshopper environment):

- Rutten, D., Galapagos. 2019.
- Delft, T., Optimus. 2020.
- Mohammed Makki, M.S., Yutao Song, Wallacei 2021.
- Olsen, C.B., Biomorpher. 2020.
- Vierlinger, R., Octopus. 2018.
- Wortmann, T., Opossum. 2021.

Other related tools:

- Jabi, W., Topologic. 2020.
- Andrew Heumann, M.S., Nate Holland, Brian Ringley, Human UI. 2019.
- Steve Baer, S.D. Hops Component. 2021.

References

- Chaillou, S. (2020). Archigan: Artificial intelligence x architecture. *Architectural intelligence* (pp. 117–127). Springer.
- Chen, Q., et al. (2020). Intelligent home 3d: Automatic 3d-house design from linguistic descriptions only. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*.
- Ekici, B., et al. (2019). Performative computational architecture using swarm and evolutionary optimisation: A review. *Building and Environment*, 147, 356–371.
- Green, O. (2020). *An introspective approach to apartment design*.
- Hu, R., et al. (2020). *Graph2Plan: Learning floorplan generation from layout graphs*. arXiv preprint [arXiv:2004.13204](https://arxiv.org/abs/2004.13204)

- Liu, Y., et al. (2020). Exploration of campus layout based on generative adversarial network. In *The international conference on computational design and robotic fabrication*. Springer.
- Nauata, N., et al. (2020). *House-GAN: Relational generative adversarial networks for graph-constrained house layout generation*. arXiv preprint [arXiv:2003.06988](https://arxiv.org/abs/2003.06988)
- Para, W., et al. (2020). *Generative layout modeling using constraint graphs*. arXiv preprint [arXiv:2011.13417](https://arxiv.org/abs/2011.13417)
- Schwartz, Y., et al. (2021). A decision support tool for building design: An integrated generative design, optimisation and life cycle performance approach. *International Journal of Architectural Computing*, 1478077121999802.
- Steve Baer, S. D. (2021). *Hops component*. <https://developer.rhino3d.com/guides/grasshopper/hops-component/>
- Tian, R. (2020). Suggestive site planning with conditional GAN and urban GIS data. In *The international conference on computational design and robotic fabrication*. Springer.
- W3Schools. (2021). *Machine learning—Decision tree*. [cited 2021]. https://www.w3schools.com/python/python_ml_decision_tree.asp
- Yao, J., et al. (2021). *Generative design method of building group—Based on generative adversarial network and genetic algorithm*.
- Zhao, C., et al. (2021). Two generative design methods of hospital operating department layouts based on healthcare systematic layout planning and generative adversarial network. *Journal of Shanghai Jiaotong University (science)*, 26(1), 103–115.