GenFloor: Interactive Generative Space Layout System via Encoded Tree Graphs

Mohammad Keshavarzi\textsuperscript{a,*}, Mohammad Rahmani-Asl\textsuperscript{b}

\textsuperscript{a}Department of Architecture, University of California, Berkeley, CA, USA
\textsuperscript{b}Autodesk Inc., San Francisco, CA, USA

ARTICLE INFO

Keywords:
Generative Design
Floorplanning
Optimization
Space Layout
Performance-based Design
Optioneering

ABSTRACT

Automated floorplanning or space layout planning has been a long-standing NP-hard problem in the field of computer-aided design, with applications in integrated circuits, architecture, urbanism, and operational research. In this paper, we introduce GenFloor, an interactive design system that takes geometrical, topological, and performance goals and constraints as input and provides optimized spatial design solutions as output. As part of our work, we propose three novel permutation methods for existing space layout graph representations, namely O-Tree and B*-Tree representations. We implement our proposed floorplanning methods as a package for Dynamo, a visual programming tool, with a custom GUI and additional evaluation functionalities to facilitate designers in their generative design workflow. Furthermore, we illustrate the performance of GenFloor in two sets of case-study experiments for residential floorplanning tasks by (a) measuring the ability of the proposed system to find a known optimal solution, and (b) observing how the system can generate diverse floorplans while addressing given a constant residential design problem. Our results indicate convergence to the global optimum is achieved while offering a diverse set of solutions of a residential floorplan corresponding to local optima of the solution landscape.

1. Introduction

Space layout planning is the process of placing a set of discrete but independent spatial elements while attempting to satisfy geometrical, topological, and performance goals in their layout. Due to the combinatorial enumeration of spatial relationships and the subjective nature of their inter-dependencies, large numbers of potential solutions can be generated even with a small number of space elements. Such complexity classifies the floorplanning process as an NP-Hard problem, where the size of solution spaces grows exponentially as the size of the problem increases [31]. Applications of floorplanning can be seen in a wide variety of fields, such as integrated circuits [64], architecture, urbanism, and operational research [2]. More recently, with the rise of augmented and virtual reality platforms, the need to automate surrounding object placements has surpassed virtual applications [38], and tasks such scene synthesis [75] and indoor object augmentation [37] take advantage of various implicit or explicit space layout methodologies, and are seen as promising research directions for next-generation computing platforms.

This level complexity of floorplanning has encouraged researchers to explore this problem by developing generative systems that take advantage of meta-heuristic search approaches. The general process of such systems can be described in three modules (i) generating various layouts via a generative function (ii) analyzing the layouts using certain design objectives via a fitness function, or comparing them to a set of priors; (iii) iterating this search until the optimal solutions are found. Generating layouts often involves encoding a floorplan as a solution to a floorplan representation function. This representation not only defines the complexity of the solution but also impacts the efficiency of the search process to find a desired floorplan in the solution landscape. Evaluation metrics can vary based on the design problem itself. In architectural space planning, for instance, performance metrics such as functional topologies, daylight, energy consumption, etc. come along with geometrical requirements such as site boundaries, room areas, etc. In digital design and VLSI (very large scale integrated-circuit) floorplanning, factors such as wire length and bounding area are optimized while fulfilling the requirements of the fabrication technology (sufficient wire spacing, restricted number of wiring layers, etc.). As each performance objective induces a new dimension within the solution space, searching

*Corresponding author

\textsuperscript{*}mkeshavarzi@berkeley.edu (M. Keshavarzi); mohammad.asl@autodesk.com (M. Rahmani-Asl)

ORCID(s): 0000-0003-2881-165X (M. Keshavarzi)
for the desired solution has been widely incorporated with meta-heuristic search methods such as Genetic Algorithms, Simulated Annealing, Particle Swarm Optimizations, Ant Colony Optimization, etc.

While much of the floorplanning methodologies proposed in current literature can be incorporated into cross-disciplinary challenges, many current space layout systems only target a specific range of design applications. Floorplanning software used by VLSI designers, for instance, is not often used by architects, and vice versa. This is mainly due to fact that such systems differ in performance evaluation criteria and are designed primarily for users targeting a specific application. Moreover, choosing the best heuristic optimization method can also be dependent on the design problem itself. Previous studies have compared these search methods in various floorplanning benchmarks, showing how each may perform better than the other in different design requirements [74, 8, 64]. However, not all optimization methods can be utilized in current floorplanning systems for a non-expert user to explore different optimization strategies. Such investigation, that would incorporate custom performance analysis and allow the user to choose between different search methods often requires advanced proficiency in programming, while allocating considerable time and resources.

In the past two decades, many designers have adopted visual-programming languages (VPLs) such as Grasshopper and Dynamo to automate their design workflows. VPLs break down the functions of software into discrete components or nodes using an intuitive graphical interface. Each component within a VPL has its own inputs and outputs, essentially acting as an individual tool within a larger script or workflow. As a result, users can customize their workflows based on the arrangement of components in their scripts, enabling them to engage different issues as they become relevant and experiment with new creative workflows as unique situations arise on projects [69]. Moreover, many building performance evaluation libraries and third-party tools which facilitate bi-directional data-flow between simulation engines have become increasingly available through these platforms. In addition, meta-heuristic optimization libraries such as SPEA-2 [76], HypE [5], NSGA-2 [13] have been integrated in various VPL plugins, allowing users to perform generative design task, and visualize the results using the incorporated visualization tools. However, with all the design opportunities VPLs have brought for generative designers, there are a limited number of end-to-end workflows available in VPLs that take advantage of established cross-disciplinary floorplanning representations, which are specifically designed for the meta-heuristic optimizations available in VPLs.

Motivated by this challenge, in this paper we introduce an interactive generative floorplanning design system, GenFloor, which is developed as a library package for Autodesk Dynamo, allowing users to incorporate generative floorplanning mechanism with third-party performance evaluators for optimization purposes within commonly used CAD software. With a user-centric design approach and an end-to-end workflow, GenFloor takes geometrical, topological, and performance goals and constraints as input and provides optimized spatial design solutions as output. By proposing three novel permutation methodologies for binary and non-binary trees, and integrating them with meta-heuristic optimization algorithms and defined performance evaluators, GenFloor enables designers to generate and optimize Manhattan-based floorplans for various design applications.

2. Related Work

Since the early 1960s, various methods have been developed for the computer-based solution of floorplanning problems [15, 19]. One of the earliest work is of Armor and Buffa’s in CRAFT [3]. CRAFT executed automated floorplanning tasks on Sketchpad, the first CAD system developed by Sutherland. Floorplanning research followed with the work of of Grason [23], Miller [50], Ligget [42] and Mitchel [51] in the next two decades. Work of March and Steadman [46] and Shaviv [63] review the first 30 years of spatial allocation algorithms.

The method that was employed in nearly all floorplanning and space layout systems at the time was a combination of a generative mechanism for the creation of alternative variations and an evaluation mechanism to assess these variations [52]. However, many classical approaches attempted to exhaustively search through all possible arrangements with a specified number of rooms. In the 1990s, Flemming presented systems for generating layouts using constraint programming which was developed under the name SEED [18] and LOOS [17, 16]. LOOS used orthogonal structures for the representation of loosely packed arrangements of rectangles. In the same decade, early examples of the use of evolutionary approaches for floorplanning in architecture were explored by Tam [68] and Gero and colleagues [30, 20, 61, 31, 60]. An example of such an approach can be seen in Jo and Gero’s work [31] where topological and geometrical goals of spatial elements were encoded in a GA for a large office layout problem. Though not many variations of the spatial arrangement were highlighted, their work showed the robustness of coupling genetic algorithm-based searches with design workflows to produce good results in space planning.
In the 2000’s Work of Michalek, et al. [49], Knecht [41], and Doulgerakis [14] took advantage of graph and tree-based representation previously used in VLSI floorplanning [32, 70] and were utilized with various optimization search methods in architectural contexts. Arvin and House [4] apply physically-based modeling to layout optimization, representing rooms and adjacencies as a mass-spring system. This heuristic has only been demonstrated on collections of rectangles and is sensitive to the initial conditions of the system. Nassar [58] presented new findings in graph theory with direct implications in space planning problems. He described architectural space plans as simple, connected, labeled planar graphs, and elaborated on the relevance of finding a rectangle dual for every planar graph to increase solution space. This work outlined a tool for architects to generate space plans. Realizing spatial relationships as planar graphs with nodes as rooms and edges as adjacencies, it claimed that the proposed model could provide a truly exhaustive set of potential designs. Work of Slusarczyk et al.[65] follows a similar approach, where they propose a framework for supporting the design process by defining design requirements over Hierarchical layout hypergraphs (HL-graphs) [22] representations of floorplan designs.

More recently, researchers in the realm of Machine Learning have explored how Generative Advisoral Networks (GAN) can facilitate the space layout problem in architectural contexts. Originally introduced by Goodfellow et al, [21] GANs leverage a feedback loop between a Generator and Discriminator model, to slowly build an ability to create synthetic data, factoring in phenomena found among observed data. The work of Isola et al [28] in Pix2Pix extends this generative ability to images, by allowing networks to learn a proper mapping from one image to another. Huang and Zheng [27] take advantage of Pix2PixHD [71] to recognize and generate furnished architectural drawings. They do this by translating floorplan images to programmatic patches of color, and inversely, generate patches of color that are turned into floorplans. Chaillou expands this approach by nesting three models (footprint, program, and furnishing) to generate floorplans, given a set of initial conditions and constraints. Extracting and manipulating existing layouts from images and sketches using model retrieval methods [36, 6] has also been used as a data augmentation method for generating new layouts [34].

However, there are two main drawbacks in current GAN floorplanning methods which can be highlighted especially in performance-oriented design scenarios. First, as with any machine-learning model, GANs learn from statistically significant features from the input data. Hence, the discriminator’s ability to classify various design generations is dependent on the data initially inputted to the discriminator. Given the fact that the reviewed systems above take advantage of the Pix2Pix model, the significant features required for the learning model are extracted from visual features of input floorplan pictures. Therefore, in performance-based design scenarios, the user must ideally provide images of pre-designed layouts with similar design constraints and performance goals, which may be difficult to gather subject to the nature of the problem. Even from a visual standpoint, the results generated by GAN floorplanning systems may lack emergence and layout diversity when limited to previously designed datasets. Second, training current GAN systems is time-consuming procedures and often achieved with high-performance computational hardware. If we consider the design process as an iterative process, where the designer may constantly change its performance priorities and goals based on various outputs in each design step, re-training and running such systems to alter design strategies may be a time-consuming procedure.

Floorplanning is also considered an essential step in the VLSI physical design process. In such context, floorplanning methodologies mostly aim at optimizing chip area and wire lengths to reduce interconnections and improve timing [32]. The floorplan representation plays an important role in the robustness of the algorithm [29] and determines the size of the search space and the complexity of transformation between a representation and its corresponding floorplan. Floorplanning representations are generally divided into two main categories: slicing and non-slicing representations [70]. In slicing methodologies, the floorplan is recursively bisected until each part consists of a single module. Commonly represented by a binary tree, modules are represented at the leaves of the tree and horizontal and vertical cuts are represented as internal nodes. The normalized Polish Expression [72] is considered as a popular slicing representation, where the post-fix ordering of a binary tree is implemented after post-order traversal on a binary tree and does include any node of the same cut type as its right child.

Non-slicing representation are utilized for more general use cases where no recursive bisection of a certain area takes place.[56, 9, 25, 24, 45, 44] Such representation are often defined in constraint graphs defining the horizontal and vertical relations among the modules. While no bisection of specific boundary is performed, non-slicing representations can be bounded by a given fixed outline. Representations such as Sequence Pair [55], Bounding Slicing Grid [56], B*Tree [9], Integer Coding [25], OTree[24], Corner Block List[45], Transitive Closure Graph [44] are among widely used non-slicing methodologies. Multiple studies have integrated these representations with various optimization algorithms such as Simulated Annealing (SA) [39, 40, 72], Genetic Algorithms (GA) [59, 57, 43, 25, 73] and...
Particle Swarm Optimization (PSO) [67, 11, 33, 66, 53]. These methods are also used in different versions of commercial floorplanning software such as PARQUET Floorplanner[1] where an inbuilt SA optimizer is implemented for the search and optimization process. Our system takes advantage of non-slicing OTree and B*Tree representations but differs in the perturbation methodology previously used in these methods.

It is important to note, all representations covered above provide an initial general layout for spatial elements. Additional operators can be implemented to transform the rectangular solutions into more complex geometries, or allocated circulations space in applications such as architecture and facility layout planning. Such an approach would result in more satisfactory results. An example of such an approach can be seen in the work of Marson and Musse [47] for instance, implemented a squarified treemap algorithm [7] to compartmentalize the input space into different zones or regions. These zones are organized into a hierarchy that satisfies design goals and site constraints and is visualized into a square treemap to generate various floor layouts. Their work excels at building sophisticated circulation networks. After the rooms are located in the boundary, a circulation network graph is built to understand which rooms are connected or disconnected from each other. They use an A* algorithm to traverse the connectivity graph and find the shortest paths to access spaces from the lobby.

There is also a large body of literature focused on how to interpret generative design solutions, and to provide the user with appropriate workflows to modify and interact with the generated solution space [62, 26, 48]. Chaszar et al. [10] explored methods and tools for multivariate interactive data visualization of the generated designs and simulation results by enabling designers to not only focus on high-performing results but also examine suboptimal ones. Mueller and Ochsendorf [54] proposed a computational design exploration approach that takes advantage of an interactive evolutionary algorithm to integrate the designers’ preferences within the solution search. Work of Keshavarzi et al. [35] extends the user interaction with the solution space from 2D input to 3D exploration, by developing a virtual reality generative analysis framework for navigating large-scale solution spaces. In this work, we take advantage of native generative design visualization and exploration tools in Dynamo VPL, enabling the user to visualize, filter and analyze Pareto-optimal solutions in real-time in different performance metrics.

The work of Das et al. [12] is probably the most similar work to our system where they also take advantage of efficient tree representation via Kd-trees as an initial generation method. Furthermore, they implement spatial grouping and circulation functions to achieve more feasible architectural floorplanning results. Their system is also developed as a library that can be loaded to Dynamo VPL, allowing users to generate and evaluate various floorplanning representations. However, from our analysis of their system, their generative model includes multiple random functions and therefore does not generate deterministic results on multiple runs. Such attributes can cause challenges for solution convergence in evolutionary solvers, especially population-based meta-heuristic optimization algorithms, where different results are generated in each generation of the search. In contrast, our system uses a deterministic encoding mechanism, providing a constant result for the same input and does not integrate any random functions with its generative module.

3. Methodology

GenFloor is developed with the goal to be utilized by users with limited proficiency in visual scripting and programming while allowing advanced users to be able to modify and adjust input parameters of the system, dependent on their needs. As a first step, users define the geometrical and topological requirements of each space by producing Spatial Requirement entities. Next, using encoding functions, a tree structure and its corresponding Spatial Blocks are generated, resembling a spatial layout corresponding to an array of Permutation Parameters. The Spatial Blocks can then be analyzed using various performance evaluators customized by the user. Finally, by integrating the system with a meta-heuristic multi-objective search mechanism, the optimization of the floorplans takes place allowing the system to find optimal tree permutations and subsequently their corresponding floorplans matching the defined goals and constraints specified by the user.

We implement our proposed methodology as a software package for the Autodesk Dynamo platform. Dynamo is a visual programming language (VPL) used by engineers, architects, and urban designers allowing custom visual scripting in conventional design and engineering software such as Revit, Fusion, and FormIt. Moreover, visual programming platforms allow third-party developers to build various add-on libraries and custom functions as packages. Such packages can serve as custom performance-based analysis functions when integrated with GenFloor’s generative design framework. Floorplans generated via GenFloor can be analyzed using custom libraries available in the visual programming platform developed by other developers, while in contrast, standalone software outside VPL plat-
GenFloor: Interactive Generative Space Layout System via Encoded Tree Graphs

Figure 1: GenFloor components available in Dynamo VPL. The Define components provide SpatialRequirements for the Generate module, which outputs SpatialBlocks. The SpatialBlocks are further analyzed and used as fitness for the search mechanism. Finally, the outcome of the results can be visualized using the corresponding components.

forms may not have the ability of such integration. GenFloor is developed using ZeroTouchNodes and CustomUINodes libraries which are all written in C# language.

In the following subsections, we elaborate on the details of the system in four major steps, commonly used in a generative design workflow: (1) Define, where the user provides the system the Spatial Requirements; (2) Generate, where the system generates the Spatial Blocks; (3) Evaluate and Search, where the fitness of each solution is calculated and is input to the ongoing search (4) Visualize, where the final floorplan, its tree representation and adjacency graphs are visualized or converted to a BIM model.

4. Define

As a starting step, users of the system define geometrical and topological spatial requirements by either manually defining each spatial block using a customized interactive GUI node in the Dynamo interface, input from other dynamic data streams in Dynamo, or importing program information from an excel spreadsheet. All three approaches are shown in Figure 2. Block dimensions and adjacency goals can be defined in this step. Defining the spatial requirements using dynamic data streams would allow more advanced users to parameterize the input data with more control (eg. parameterize the dimension of the block), while the spreadsheet data parsing method is targeted for general and traditional floorplanners that organize their program requirements in spreadsheet format. The data inputted in this step is primarily used for generating the tree structures but can also serve as parameters defining the floorplanning goals and constraints.

By knowing the quantity of the blocks \( n \), a Standard Tree with \( n \) nodes is generated. Since we use an O-Tree and B*Tree representation in our system, we develop two methods of Standard Tree generation for each representation. In a B*Tree generation scenario, a Standard Tree is a complete binary tree of \( n \) nodes, where each node corresponds to Spatial Requirement defined by the user. A complete binary tree is a binary tree in which every level, except possibly the last, is filled, and all nodes are as far left as possible. The order of the nodes in the Standard Tree resembles the order in which the user inputted each entity of the geometrical block properties. Therefore, the first entity corresponds to the tree root and the final entity is the far-right node on the last level. In an O-Tree scenario, the nodes are located on one single level, ordered from left to right. Figure 3 shows an example of a Standard Tree for a B*Tree and O-Tree.
5. Generate

The goal of this module is to convert the Spatial Requirements to a tree representation and further to a set of Spatial Blocks using the Permutation Parameters. As part of our work, we propose three novel permutation methods for existing space layout graph representations, namely O-Tree and B*-Tree representations to perturb the Standard Tree with Permutation Parameters. Permutation Parameters $P$ are an array of values, which vary in size and attributes depending on the permutation method. We elaborate more on the properties of different types of Permutation Parameters when describing each permutation method. Our methods provide deterministic results of tree arrangements and do not utilize any random functions in their process. After perturbing the Standard Tree with one of the permutation modules, the subsequent tree is converted to a floorplan using O-Tree or B*-Tree representations. Users of our system can generate space layouts by choosing one of the block generation methods. In simple terms, the Generation modules take Spatial Requirement data and Perturbation Parameters as input and output a list of Spatial Blocks which formulate together a determined layout.

To facilitate the explanation of our system, we initially describe how O-Tree and B*-Tree non-slicing floorplanning representations are defined, and then present our proposed permutation methods. Furthermore, we explain our implementation with the goal of allowing non-expert users to generatively design and manipulate floorplanning outputs. Finally, we introduce a procedural mechanism to extend the non-slicing representations to a slicing representation, allowing our system to generate layouts for applications such as indoor furniture layout.

5.0.1. O-Tree Representation

An O-tree is an ordered non-binary tree representing a determined floorplan layout. We define and implement a modified version of O-Trees first introduced by Guo et. al [24] mainly altering perturbation methodologies. Given a non-binary tree of function blocks, we first order the nodes using a depth-first search (DFS). Note that the root node does not correspond to any function block and represents the left boundary of the placement. To generate the floorplan using the ordered nodes we take the following steps for each function block. Each child $j$ of a parent node $i$ is placed on the right side of the parent node $i$ where $x_j = x_i + w_j$. For determining the height of the function block placement, the block is placed on top of the horizontal contour of the placement and is updated recursively when each function block is placed.

5.0.2. B*-Tree Representation

A B*-Tree is an ordered binary tree representing a floorplan layout. Our implementation of the B*-Tree floor planning representation is a modified version of B*-Tree, introduced in Chang et al study [9] while having major differences in the perturbation operations. In a B*-Tree representation, the block assigned to the root node is placed on the bottom-left corner. Using a depth-first search (DFS) procedure, we recursively construct the floor-plan layout by keeping the geometric relationship between two modules as follows. If node $b$ is the left child of node $a$, the block assigned to node $b$ must be located on the right-hand side to the block assigned to node $a$ in the floor planning placement such as $x_b = x_a + w_a$, where $x_i$ is the starting point and $w_i$ is the width of block $i$. During the placement of
5.2. Ascend/Descend Perturbation Method of B*-Trees

Using the Proceeding method perturbation. Figure 4 shows the permutation of an O-Tree via a given

active node

placements, we perturb the O-tree in each step, the active node is to be relocated or swapped with another node. We define the target node as the new position of the active node. Starting with Standard Binary Tree, the permutation parameters of the Ascend/Descend method can be found in Table 1. Figure 5 illustrates the permutation of an O-Tree via a given P and it’s tree representation using the Proceeding method perturbation.

5.1. Proceeding Perturbation of O-Tree Representations

The Proceeding Perturbation method converts a Standard O-Tree T_s = \{D_0, D_1, \ldots D_n\} into various tree structures via input Permutation Parameters P = \{p_1, p_2, \ldots, p_n\} where p_i \in \{0, 1, \ldots, n\}. To produce alternative floorplanning placements, we perturb the O-tree in n steps, where in each step i, D_{(p_i)} becomes the parent of D_i. In other words, in each step, the active node proceeds the target node which holds the index of the corresponding Permutation Parameter. However, the perturbation is discarded if p_i = i; or D_i is to proceed a target node that has been defined as it’s child or descendant in previous steps. Figure 4 shows the permutation of an O-Tree via a given P and it’s tree representation using the Proceeding method perturbation.

5.2. Ascend/Descend Perturbation Method of B*-Trees

Similar to the Proceeding Perturbation method, the Ascend/Descend method takes place in n steps, wherein each step the active node is to be relocated or swapped with another node. We define the target node as the new position of the active node. Starting with Standard Binary Tree, the permutation parameters of the Ascend/Descend method are defined as P = \{(p_{1,1}, p_{1,2}), (p_{2,1}, p_{2,2}), \ldots, (p_{n,1}, p_{n,2})\}, where p_{i,1}, p_{i,2} \in \{0, 0.5, 1, 1.5, \ldots, 2n\}. p_{i,1} indicates the maximum levels the node moves up (ascend) in a tree, and p_{i,2} indicates the maximum number of steps the node is to move down (descend) in the binary tree in a breadth-first search (BFS) format. Half-step traversals are also defined to allow the insertion of a node between two levels. After locating the end position of the node relocation, if the end position is occupied by another node, swapping takes place. If no other node occupies the target position, the node creates an internal (a position between two nodes) or an external branch within the tree. We follow the similar procedures for deleting, inserting, and switching block function nodes explained in [9]'s study. Additional details of the different operations of the Ascend/Descend method can be found in Table 1. Figure 5 illustrates the permutation steps of the Ascend/Descend perturbation method.

5.3. Available Nodes of B*-Trees

Similar to the Ascend/Descend method, the Available Nodes method aims to relocate nodes of a input Standard binary-tree T_s = \{D_1, D_2, D_3, \ldots, D_n\} in n steps, each time relocating one active node. The Permutation Parameters for this method is defined as P = \{p_1, p_2, \ldots, p_n\}, where p_i \in \{0, 0.5, 1, 1.5, \ldots, 2n\}. For each binary-tree T with n nodes, there are 4n possible target locations in which a node can be relocated resulting in a creation of either an internal node, external node, or a node swap. We call these potential target node locations available nodes and represent them in BFS order in T_{Av} = \{D_{0,0}^1, D_{0,0.5}^1, D_{0,1}^1, \ldots, D_{0,2n}^1\}. In each step, D_i will be relocated to D_{p_i} using the same operations used in the Ascend/Descend method (Table 1), and T_{Av} will be updated given the new tree structure. In contrast to theAscend/ Descend method where the solution space holds a 2n^n + (4n)^n complexity, the Available nodes method results in (4n)^n complexity achieving faster results. Figure 6 illustrates the process of the Available Nodes method.
5.4. Extended B*Tree

While both O*Tree and B*Tree are tree representations of admissible non-slicing floorplan layouts, they can also be extended as slicing methodologies for a given spatial boundary. Such an approach would facilitate the layout planning in scenarios where all the given boundary is expected to be used without leaving empty space. Converting non-slicing representations to slicing representations can extend the potential applications of our work to indoor furniture layout. Given an input boundary space, the system generates allocated areas for each furniture entity, while maintaining object-to-object topological requirements between objects.

We implement our extended B*Tree method in four main steps: (i) generate B*Tree layout; (ii) scale the generated layout height and width to fit the input bounding area. We call the new blocks, “extended blocks”. If the ratio of the height or width is bigger than one, we penalize the B*Tree solution as the function blocks cannot fit within the bounding space without overlapping each other; (iii) We extend each extended block to the closest boundary edge, if not blocked by another extended block. This would allow the extended blocks to fill all available spaces within the boundary space. (iv) Finally, for non-flexible block functions such as furniture, where the dimension of the block should be constant, we allocate a section of the new extended blocks as the final location of the block while considering block constraints such as direction and edge requirements. This step does not execute for blocks that can have flexible dimensions. Figure 7 illustrates the process and Figure 8 shows examples of indoor furniture generation using the Extended B*Tree algorithm.

6. Evaluation and Search

The resulting floorplan can be then analyzed by various customized fitness functions corresponding to the project requirements. In this module, performance evaluations and simulations are incorporated to evaluate the fitness of the floorplan. Other than the geometrical and topological analysis functions provided by GenFloor, users can integrate third-party libraries and simulators to analyze the generated layout. GenFloor provides basic evaluation functions that can be used as objectives or constraints of the search system. The number of adjacency goals achieved, bounding boxes,
Table 1
Ascend/Descend scenarios and operations.

| Function ID | Scenario                      | Operation                                                                 |
|-------------|-------------------------------|---------------------------------------------------------------------------|
| SU          | Step up                       | Target node is equal with target node parent \( T = T_p \).               |
| HSU         | Half step up                  | Target node is inserted between target node parent and target node.      |
| SR          | Up step count is larger than  | Target node is equal to root node \( T = D_0 \).                         |
|             | node level \( N_{ip} > N_i \) |                                                                            |
| HSN         | Up step is 0.5 \( N_{ip} = 0.5 \) | Active nodes equal to target node and graph stays unchanged.  |
| SD          | Step down                     | Target node is updated with next graph member in BSF format.              |
| HSD         | Half step down                | Target node is placed between Ndown ceiling and Ndown ceiling parent.   |
| SW          | Target node points to a graph | Active node and target node switch places.                                |
|             | member                        |                                                                            |
| IN          | Target node is not graph      | Active node is deleted and inserted between two graph member nodes.       |
|             | member and is between two     |                                                                            |
|             | graph members                 |                                                                            |
| DN          | Node is deleted and has no    | Parent node pointer to deleted node is set to NULL.                      |
|             | child                         |                                                                            |
| DL          | Node is deleted and has left  | Node is switched with left child and then deleted.                       |
|             | child                         |                                                                            |
| DR          | Node is deleted and has right | Nodes is switched with right child and then deleted.                     |
|             | child only                    |                                                                            |
| SP          | Target node has parent that   | Target node is switched with previous step until target node has parent  |
|             | is not graph member           | that is a graph members.                                                 |

and closest distances between blocks can be calculated using native functions available in the GenFloor package.

Adjacency evaluators allow users to analyze how the generated floorplan achieves topological and adjacency goals. The evaluator (1) calculates all adjacent blocks of each Spatial Block in a given list (resulted adjacency); (2) compares the resulted adjacencies and target adjacencies (goals) informing the user whether each resulted adjacency is present in the target adjacency list or not (adjacency check); and (3) outputs the number of achieved adjacencies in all the Spatial Blocks (adjacency achieved count). Other evaluators that are incorporated in the system include bounding area calculation (returns smallest enclosed rectangle, or the minimum bounding rectangle for the generated floorplan), closest distance (returns the closest distance of Spatial Block to a single or multiple Spatial Block and bounding curves test (Calculates whether Spatial Blocks of a generated floorplan fall inside a given polygon).

Finally, GenFloor can take advantage of Dynamo’s integrated Non-Sorting Genetic Algorithm II (NSGA-2) which would allow the user to define constraints and goals for the floorplanning problem. The solver would find Permutation Parameters that correspond to tree representations of optimal solutions. While GenFloor is tested to work smoothly with Dynamo’s NSGA-2 algorithm, the generation module of the system is independent of the search mechanism, and therefore, can be replaced other search mechanisms such as simulated annealing, particle swarm optimization or other evolutionary solvers, etc. Such approaches have not been experimented with within this study, however, given the modular development of our GenFloor library, such integration is feasible.
7. BIM Integration and Visualization

Generated floorplans can be directly converted to BIM models using incorporated functions available in the GenFloor library. Users can generate walls, doors, and rooms of the populated GenFloor floorplans in Revit. Spatial Blocks are converted to two sets of non-overlapping walls representing the interior and exterior walls. Moreover, meta-data such as room names, spaces IDs, etc. can be transferred to Revit for additional documentation purposes. A procedural door placement system is also available for users to further automate the process of floorplan generation. Figure 9 illustrates an example set of results of BIM models, generated as an output of the GenFloor system.

Furthermore, users can visualize the corresponding adjacency graphs of the generated layout with color codes determining which of its topological and adjacency goals it has been achieved and which has not. Adjacency graphs or bubble diagrams are a common method of representing topological relationships and requirements in a space layout problem. The corresponding tree structure (O-Tree or B*Tree) of the generated space layout can also be generated by the user to allow additional analysis of the system workflow. Examples of such an approach can be seen in Figure 10.

8. Case Studies

In performance-based generative design systems, the general goal is to find optimal solutions that address the constraints and goals defined by the user. Yet in applications such as space layout planning, automation can serve in various stages of the design process with different objectives. In design scenarios where goals and constraints are finalized, programmable, and clearly defined, the user is simply seeking the best solution (or a limited number of solutions) which achieves optimal performance. In such settings, an ideal performance-based generative design system is able to generate and converge towards the global optima of the solution landscape without getting trapped in local optimas.

The other scenario is when all performance metrics are not programmable by the designer or have not yet finalized the performance metrics for its design. Programming evaluators often require time and expertise, and not always can programmable due to the qualitative nature which they may hold. In such cases, the designer may be yet flexible with...
the defined metrics and would want to explore different design strategies. In such a scenario the user is looking for more diverse solutions. Instead of looking for one final answer, the designer is eager to explore a diverse set of solutions, all addressing the goals and constraints to a good level of extent. Diverse design options sampled from different parts of the solution landscape can provide the user with multiple design directions or initiate a feedback loop so a redefinition of the design problem takes place.

We, therefore, conduct two preliminary experiments to address both scenarios explained above. In the first experiment, we measure the system’s ability to find a known optimal solution, given that the goals and constraints are finalized and defined. In the second experiment, we observe how the system can generate diverse floorplans while

**Figure 7:** Extended B**\(^*\)**Tree process: (a) generate B**\(^*\)**Tree layout (b) scale the generated layout (c) extend to the closest boundary edge (d) allocate furniture location based on heuristic constraints (e) add furniture objects to location.
addressing given a constant design problem. From a search perspective, we intend to measure the ability of the system to converge to the global optimum in the first experiment, while in the second experiment assessing its ability to gather all potential solutions for the user often corresponding to local optima of the solution landscape. We further elaborate on the experiments below.

8.1. Optimal Solution Search

The goal of this experiment is to evaluate whether our system can find a deterministic optimal solution for a given space layout problem. To do so, we define a pre-designed layout as an optimal design. We then use the topological properties of a pre-defined design (optimal design) as goals and assess if the system is able to find a solution similar both in topological performance and geometrical layout. Such an approach would allow us to observe whether the system would (i) converge to a deterministic optimal solution in different runs, (ii) compare the speed and memory allocation for various permutation methods, and finally (iii) understand how various properties of the evolutionary solver such as population size, generation count and mutation rate impact the ability of our system to find the optimal design.

Figure 11 shows the general workflow of the experiment. We use a residential floorplan as a pre-designed optimal layout with 8 spatial blocks, each corresponding to a spatial function. Next, by extracting the topological properties of each block, we produce the adjacency matrix of the floorplan. This would serve as input to the GenFloor system- as an optimization goal- alongside the geometrical properties of the corresponding blocks. We then examine whether the system would generate the ideal output: a floorplan with the maximum adjacency goals achieved and a spatial layout similar to the layout defined as the optimal design.

We perform the experiment with three different settings of the GA solver. In the first setting, we initiate a limited population size \( p = 40 \) with limited generation runs \( g = 5 \). The second setting resembles a larger population with the same limited generation run \( p = 100, g = 5 \), while the third setting consists of a larger population with a larger number of generations. The cross-over rate for all runs is 20% and the mutation rate is set at 10%. Moreover, we define adjacency goals in three levels. The \( L_1 \) contains all the adjacency requirements \( A_{\text{Count}} = 28 \) extracted from the pre-designed layout, the \( L_2 \) resembles high priority adjacencies from a designers perspective \( A_{\text{Count}} = 20 \). In \( L_3 \), if we require \( S_A \) to be adjacent to \( S_B \), while \( S_B \) is adjacent to \( S_C \), then \( S_C \) adjacency to \( S_A \) will not be assigned as a goal. In such context, the number of adjacency required where decreased to \( A_{\text{Count}} = 16 \).

Table 2 shows the results for our optimal solution search experiment. In \( L_1 \) where all extracted adjacencies are present, the system is not able to find the optimal pre-designed solution with the various settings and does not meet all.
the adjacency goals. The maximum score is 0.86 resembling an over-constrained problem for the system. However, when the adjacency requirements are relaxed to a lower amount $L_1$, the system is able to generate the exact optimal layout. In such a context, $(p = 40, g = 4)$ starts with lower scores than other levels, but with increasing the gene population and generation runs, the system is able to find the optimal solution. Moreover, In the event where adjacency goals are minimum $(a = 16)$, we observe better adjacency scores in $S_{(40,5)}$ and $(p = 100, g = 5)$ than other adjacency levels. While all adjacency goals are achieved in $(p = 100, g = 15)$ the output solution is not similar to our pre-designed optimal layout.

8.2. Emerging Layout Search

In this experiment, we intend to illustrate our system’s ability to generate diverse results given constant geometrical and topological goals and constraints. In addition to the permutation parameters, we add a binary rotation parameter allowing the Spatial Block to switch its width and height. We use the same residential problem in the previous subsection with $L_1$ adjacency requirements where all spaces have at least one adjacency requirement, with the corridor having 4 adjacency goals. However, instead of generating solutions that address all adjacency requirements ($A_{Count} = 28$), we aim to visualize a broader set of solutions that maintain at least ($A_{Count} = 20$). Figure 12 illustrates the results from a $(p = 100, g = 15)$ generation and their corresponding bubble diagram visualization explained in Sec 7. The results
show how GenFloor is able to generate a diverse set of solutions for the defined space layout problem.

9. Discussions

One of the main drawbacks of using non-slicing floorplanning representation methods in floorplan generation is their lack of intuitiveness between the encoding mechanism and the output floorplanning representation. In such context, while GenFloor aims to provide additional visualization modules for generative designers to view the resulted tree structures of the input array strings, it still falls short of providing a clear and straightforward connection between the input parameters and output results for amateur users. In general, integrating slicing methods with optimization solvers is more straight-forward as control parameters can simply correspond with the $H, V$ parameters. These parameters can be explicit without requiring any post-processing conversion to floorplan representation. Such property does not always hold in non-slicing methods, as encoding methodologies and perturbation operators prevent straight-forward representations of the control parameters.

When comparing some of the limitations of the proposed permutation methods, we should highlight that the O-Tree Proceeding method comes with a larger number of discarding scenarios when compared to the proposed B*Tree permutation methods. Such property may be misleading for an evolutionary search mechanism, as solutions of neighboring parameters may need to be discarded because they don’t hold the requirements of permutation parameters, potentially disrupting the ability to converge to optimal solutions. The Ascend/Descend method comes with the limitation of potentially generating similar solutions given different parameters. For example, if the active node is located on the third level of the tree, any $p_{i,2} > 2$ would give the same ascending target node. The Available Nodes method is probably the most efficient permutation model when compared to other methods. However, it lacks an intuitive cor-
response to the floorplanning representation. Since the traversal takes place using a BSF approach, understanding the relationship between the permutation parameters and the resulting floorplan can be a challenge for a user. Whereas the Ascend/Descend and the Proceeding methods come with much more intuitive permutation parameters, allowing the user to have potentially more direct control over the floorplanning if desired.

It is important to note, all representations introduced in this paper provide an initial general layout for spatial elements. Additional operators can be implemented to transform the rectangular solutions into more complex geometries, or allocated circulations space in applications such as architecture and facility layout planning. Such an approach would result in more satisfactory results. In scenarios where computationally expensive evaluations take place, such as physical-based simulations (e.g., building energy simulations, daylighting simulations, etc.), a limited solution space would be beneficial for the search. Such an approach would also benefit workflows with limited computational resources, such as mobile applications or real-time gaming.

10. Conclusion

In this paper, we introduce an end-to-end interactive generative floorplanning design system, GenFloor, which takes geometrical, topological, and performance goals and constraints as input and provides optimized spatial design solutions as output. By proposing three novel permutation methodologies for binary and non-binary trees, and integrating them with genetic algorithms with defined performance evaluators, we are able to generate and optimize Manhattan-based floorplans for various design applications. Our system is developed as a library package for Autodesk Dynamo, a visual programming platform, allowing users to incorporate the generated floorplans with conventional CAD software and third-party performance evaluators for optimization purposes.

Future work on the development of this system can fall into three main categories. First, extending the floorplanning generation to non-Manhattan-based design solutions. This can be explored when integrating physical-based simulations into the current workflow. In such an approach GenFloor can generate initial positioning for the non-rectangular blocks, and then the system can take advantage of mass-spring simulations to configure the geometrical conflicts and resemble a balanced layout. Second, we intend to increase the speed of our system by engineering the current development code with more efficient functions and run-time procedures. The current version released does not resemble the potential speed in which tree representations can deliver in floorplanning optimization. Initially, additional interactivity in post-modeling generated floorplanning can be added to the current system. While our integration with Revit’s BIM modeling system does allow parametric modification of our generated layouts, but slight re-arrangements of the outputted layouts are not achieved smoothly in the current workflow. Such an approach would enable users to easily implement slight modifications to the generated solutions while obtaining instant feedback to control how the modified
layout affects the defined performance criteria

11. Acknowledgements

This research was initiated and supported by Autodesk’s AEC Generative Design Team. We would like to thank their support in Generative Design for Revit (formerly Project Refinery).

References

[1] Adya, S., Chan, H., Markov, L. Parquet: Fixed-outline floorplanner. online: http://vlscad.eecs.umich.edu/BK/parquet.
[2] Anjos, M.F., Vieira, M.V., 2017. Mathematical optimization approaches for facility layout problems: The state-of-the-art and future research directions. doi:10.1016/j.ejor.2017.01.049.
[3] Armour, G.C., Buffa, E.S., 1963. A heuristic algorithm and simulation approach to relative location of facilities. Management Science 9, 294–309.
[4] Arvin, S.A., House, D.H., 2002. Modeling architectural design objectives in physically based space planning. Automation in Construction 11, 213–225.
[5] Bader, J., Zitzler, E., 2011. Hype: An algorithm for fast hypervolume-based many-objective optimization. Evolutionary computation 19, 45–76.
[6] Bergin, M.S., Nourbakhsh, M., Keshavarzi, M., Cheng, C.Y., 2020. Automated parametrization of floor-plan sketches for multi-objective building optimization tasks. US Patent App. 16/681,591.
[7] Bruls, M., Huizing, K., Van Wijk, J.J., 2000. Squarified treemaps, in: Data visualization 2000. Springer, pp. 33–42.
[8] Caldas, L., Ken, 2001. An evolution-based generative design system: Using adaptation to shape architectural form. Ph.D. thesis. MIT. URL: http://hdl.handle.net/1721.1/8188.
[9] Chang, Y.C., Chang, Y.W., Wu, G.M., Wu, S.W., 2000. B*-trees: a new representation for non-slicing floorplans, in: Proceedings 37th Design Automation Conference, p. 458–463. doi:10.1109/DAC.2000.855354.
[10] Chaszar, A., Von Buelow, P., Turrin, M., 2016. Multivariate Interactive Visualization of Data in Generative Design, in: Symposium on Simulation for Architecture and Urban Design, SimAUD, London.
[11] Chen, G., Guo, W., Cheng, H., Fen, X., Fang, X., 2008. Vlsi floorplanning based on particle swarm optimization, in: 2008 3rd International Conference on Intelligent System and Knowledge Engineering, IEEE, pp. 1020–1025.
[12] Das, S., Day, C., Hauck, J., Haymaker, J., Davis, D., 2016. Space plan generator: Rapid generation & evaluation of floor plan design options to inform decision making.
[13] Deb, K., Agrawal, S., Pratap, A., Meyarivan, T., 2000. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii, in: International conference on parallel problem solving from nature, Springer, pp. 849–858.
[14] Doulgerakis, A., 2007. Genetic programming+ unfolding embryology in automated layout planning. Ph.D. thesis. UCL (University College London).
[15] ELDARS, M.Z.A., 1964. An Approach to the Optimum Layout of Single Storey Buildings from the Point of View of Circulation with Special Reference to Operating Theatre Suites. Thesis Ph. D., Liverpool University.
[16] Flemming, U., 1989. More on the representation and generation of loosely packed arrangements of rectangles. Environment and Planning B: Planning and Design 16, 327–359.
[17] Flemming, U., 1990. Planning in design synthesis, abstraction-based loos. Applications of Artificial Intelligence in Engineering V 1, 91–111.
[18] Flemming, U., Woodbury, R., 1995. Software environment to support early phases in building design (seed): Overview. Journal of architectural engineering 1, 147–152.
[19] Frew, R.S., 1980. A survey of space allocation algorithms in use in architectural design in the past twenty years, in: Proceedings of the 17th Design Automation Conference, pp. 165–174.
[20] Gero, J.S., Kazakov, V.A., 1997. Learning and re-using information in space layout planning problems using genetic engineering. Artificial Intelligence in Engineering 11, 329–334.
[21] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets, in: Advances in neural information processing systems, pp. 2672–2680.
[22] Grabksa, E., Lachwa, A., Šukarczyk, G., Grześiak-Kopeć, K., Lembas, J., 2007. Hierarchical layout hypergraph operations and diagrammatic reasoning. Machine Graphics and Vision 16, 23–38.
[23] Grason, J., 1970. Fundamental description of a floor plan design program, in: EDRA [1]. Proc. of the 1st ann. Design Research Association Conf. held at Chapel Hill, North Carolina. Stroudsburg, Pennsylvania: Dowden, Hutchinson & Ross, Inc, pp. 175–180.
[24] Guo, P.N., Cheng, C.K., Yoshimura, T. 2003. An o-tree representation of non-slicing floorplan and its applications, in: Proceedings 1999 Design Automation Conference (Cat. No. 99CH36361), IEEE, p. 268–273. URL: http://ieeexplore.ieee.org/document/781324/, doi:10.1109/3909847.309928.
[25] Gwee, B.H., Lim, M.H., 1999. A ga with heuristic-based decoder for ic floorplanning. INTEGRATION, the VLSI journal 28, 157–172.
[26] Herdy, M., 1996. Evolution strategies with subjective selection, in: Voigt, H.M., Ebeling, W., Rechenberg, I., Schwefel, H.P. (Eds.), Parallel Problem Solving from Nature — PPSIV, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 22–31.
[27] Huang, W., Zheng, H., 2018. Architectural drawings recognition and generation through machine learning, in: Proceedings of the 38th Annual Conference of the Association for Computer Aided Design in Architecture, Mexico City, Mexico, pp. 18–20.
[28] Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1125–1134.
Shaviv, E., 1987. Generative and evaluative caad tools for spatial allocation problem, in: Principles of computer-aided design: computability.

Scott, S.D., Lesh, N., Klau, G.W., 2002. Investigating human-computer optimization, in: Proceedings of the SIGCHI conference on Human.

Rosenman, M., Gero, J., 1999. Evolving designs by generating useful complex genestructures. Evolutionary design by computers, 345–364.

Mueller, C.T., Ochsendorf, J.A., 2015. Combining structural performance and designer preferences in evolutionary design space exploration.

Morita, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y., 1995. Rectangle-packing-based module placement, in: Proceedings of the 1995.

Nassar, K., 2010. New advances in the automated architectural space plan layout problem. Proceedings Computing in Civil and Building

Kiyota, K., Fujiyoshi, K., 2005. Simulated annealing search through general structure floor plans using sequence-pair. Electronics and Communications in Japan (Part II: Fundamental Electronic Science) 88, 28–38.

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. science 220, 671–680.

Kiyota, K., Fujiiyoshi, K., 2005. Simulated annealing search through general structure floor plans using sequence-pair. Electronics and Communications in Japan (Part II: Fundamental Electronic Science) 88, 28–38.

Knecht, K., König, R., 2010. Generating floor plan layouts with kd trees and evolutionary algorithms, in: Generative Art Conf. pp. 238–253.

Liggett, R.S., Mitchell, W.J., 1981. Optimal space planning in practice. Computer-Aided Design 13, 277–288.

Lin, C.T., Chen, D.S., Wang, Y.W., 2002. An efficient genetic algorithm for slicing floorplan area optimization, in: 2002 IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No. 02CH37353), IEEE. pp. II–II.

Lin, J.M., Chang, Y.W., 2005. Tcg: A transitive closure graph-based representation for general floorplans. IEEE transactions on very large scale integration (VLSI) systems 13, 288–292.

Ma, Y., Dong, S., Hong, X., Cai, Y., Cheng, C.K., Gu, J., 2001. Vlsi floorplanning with boundary constraints based on corner block list, in: Proceedings of the 2001 Asia and South Pacific Design Automation Conference, ACM. pp. 509–514.

March, L., Steadman, P., 1971. Spatial allocation procedures. The Geometry of Environment.

Marson, F., Musse, S.R., 2010. Automatic real-time generation of floor plans based on squarified treemaps algorithm. International Journal of Computer Games Technology 2010, 1–10. URL: http://www.hindawi.com/journals/ijcgt/2010/624817/, doi:10.1155/2010/624817.

Meynian, D., Knust, S., Frayret, J.M., Pesant, G., Gaud, N., 2015. A review and taxonomy of interactive optimization methods in operations research. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 17.

Michalek, J., Choudhary, R., Papalambros, P., 2002. Architectural layout design optimization. Engineering Optimization 34, 461–484. URL: http://www.tandfonline.com/doi/abs/10.1080/03052150214016, doi:10.1080/03052150214016.

Miller, W.R., 1971. Computer-aided space planning-an introduction-. dmg newsletter vol. 5 no4/5. Design Methods Group.

Mitchell, W.J., 1975. The theoretical foundation of computer-aided architectural design. Environment and Planning B: Planning and Design 2, 127–150.

Mitchell, W.J., 1990. The logic of architecture: Design, computation, and cognition. MIT press.

Moni, D.J., Arumugam, S., 2009. Vlsi floorplanning based on hybrid particle swarm optimization. Karunya Journal of Research 1, 111–121.

Mueller, C.T., Ochsendorf, J.A., 2015. Combining structural performance and designer preferences in evolutionary design space exploration. Automation in Construction 52, 70–82.

Murata, H., Fujiyoshi, K., Nakatake, S., Kajitani, Y., 1995. Rectangle-packing-based module placement, in: Proceedings of the 1995 IEEE/ACM International Conference on Computer-aided Design, IEEE Computer Society, Washington, DC, USA. pp. 472–479. URL: http://dl.acm.org/citation.cfm?id=224841.226094.

Nakatake, S., Fujiyoshi, K., Murata, H., Kajitani, Y. 1997. Module placement on bsg-structure and ic layout applications, in: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design, IEEE Computer Society. pp. 484–491.

Nakaya, S., Koido, T., Wakabayashi, S., 2000. An adaptive genetic algorithm for vlsi floorplanning based on sequence-pair, in: 2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No. 00CH36353), IEEE. pp. 65–68.

Nassar, K., 2010. New advances in the automated architectural space plan layout problem. Proceedings Computing in Civil and Building Engineering.

Reboudengo, M., Reorda, M.S., 1996. Gallo: A genetic algorithm for floorplan area optimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15, 943–951.

Rosenman, M., Gero, J., 1999. Evolving designs by generating useful complex gene structures. Evolutionary design by computers, 345–364.

Schmier, T., Gero, J.S., 1996. Learning genetic representations as alternative to hand-coded shape grammars, in: Artificial Intelligence in Design’96. Springer, pp. 39–57.

Scott, S.D., Lesh, N., Klauf, G.W., 2002. Investigating human-computer optimization, in: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM. pp. 155–162.

Shaviv, E., 1987. Generative and evaluative caad tools for spatial allocation problem, in: Principles of computer-aided design: computability of design, Wiley-Interscience. pp. 191–212.

Keshavarzi and Rahmani-Asl: Preprint submitted to Elsevier
[64] Singh, R.B., Baghel, A.S., Agarwal, A., 2016. A review on VLSI floorplanning optimization using metaheuristic algorithms, in: International Conference on Electrical, Electronics, and Optimization Techniques, ICEEOT 2016, Institute of Electrical and Electronics Engineers Inc., pp. 4198–4202. doi:10.1109/ICEEOT.2016.7755508.

[65] Słusarczyk, G., 2018. Graph-based representation of design properties in creating building floorplans. Computer-Aided Design 95, 24–39.

[66] Sowmya, B., Sunil, M., 2013. Minimization of floorplanning area and wire length interconnection using particle swarm optimization. International Journal of Emerging Technology and Advanced Engineering 3.

[67] Sun, T.Y., Hsieh, S.T., Wang, H.M., Lin, C.W., 2006. Floorplanning based on particle swarm optimization, in: IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures (ISVLSI'06), IEEE, pp. 5–pp.

[68] Tam, K.Y., 1992. Genetic algorithms, function optimization, and facility layout design. European journal of operational research 63, 322–346.

[69] Tedeschi, A., 2014. AAD, Algorithms-aided design: parametric strategies using Grasshopper. Le penseur publisher.

[70] Wang, L.T., Chang, Y.W., Cheng, K.T.T. (Eds.), 2009. Electronic Design Automation: Synthesis, Verification, and Test. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[71] Wang, T.C., Liu, M.Y., Zhu, J.Y., Tao, A., Kautz, J., Catanzaro, B., 2018. High-resolution image synthesis and semantic manipulation with conditional gans, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 8798–8807.

[72] Wong, D.F., Liu, C.L., 1986. A New Algorithm for Floorplan Design, in: Proceedings of the 23rd ACM/IEEE Design Automation Conference, IEEE Press, Piscataway, NJ, USA. pp. 101–107. URL: http://dl.acm.org/citation.cfm?id=318013.318030.

[73] Xiaogang, W., et al., 2002. Vlsi floorplanning method based on genetic algorithms [j]. Microprocessors , 1.

[74] Youssef, H., M. Sait, S., Adiche, H., 2001. Evolutionary algorithms, simulated annealing and tabu search: A comparative study. Engineering Applications of Artificial Intelligence 14, 167–181. doi:10.1016/S0952-1976(00)00065-8.

[75] Zhang, S.H., Zhang, S.K., Liang, Y., Hall, P., 2019. A survey of 3d indoor scene synthesis. Journal of Computer Science and Technology 34, 594–608.

[76] Zitzler, E., Laumanns, M., Thiele, L., 2001. SPEA2: Improving the strength Pareto evolutionary algorithm. TIK-report 103.