



An evolutionary approach for 3D architectural space layout design exploration

Ipek Gürsel Dino

Middle East Technical University, Department of Architecture, Üniversiteler Mah., Dumlupınar Bulvarı, No:1, 06800 Ankara, Turkey



ARTICLE INFO

Article history:

Received 9 November 2015

Received in revised form 24 April 2016

Accepted 22 May 2016

Available online 24 June 2016

Keywords:

Design exploration

Evolutionary algorithms

Architectural space layout design

ABSTRACT

This work introduces Evolutionary Architectural Space layout Explorer (EASE), a design tool that facilitates the optimization of 3D space layouts. EASE addresses architectural design exploration and the need to attend to many alternatives simultaneously in layout design. For this, we use evolutionary optimization to find a balance between divergent exploration and convergent exploitation. EASE comprises a novel sub-heuristic that constructs valid spatial layouts, a mathematical framework to quantify the satisfaction of constraints, and evolutionary operators to improve alternative layouts' fitness. We test EASE on the design of a library building. We evaluate EASE's performance for different building forms and different evolutionary algorithm parameters. The results suggest that EASE can generate valid layouts, quantify the constraints' degree of satisfaction and find a number of optimal layout solutions. The layouts that EASE generates are intended not as end results but design artifacts that provide insight into the solution space for further exploration.

© 2016 Elsevier B.V. All rights reserved.

1. Introduction

Space layout design (SLD) is one of the key phases of architectural design, which comprises decisions regarding the search for an optimal spatial configuration that satisfies a set of constraints. It is also a complex problem due to the subjective and fuzzy nature of dependencies, the difficulties in quantifying solution quality and its discontinuous and multimodal design space [29]. SLD typically is manually conducted. However, due to the vast number of alternatives, alternative configurations cannot be systematically explored by hand.

Computation-aided design optimization can support SLD by the automated generation, manipulation and evaluation of design alternatives. This way, rational decision-making based on quantitative criteria can be facilitated towards well-performing design solutions. At the same time, creative design problems are said to resist being solved to optimality by deterministic methods, as there is no complete understanding of the problem structure at the outset of the design process and the relationship between the design variables and objective function(s) is not clear. This mismatch between rational methods of exploitation and creative acts of design synthesis is a major hindrance to the effective implementation of optimization in early design. Moreover, it is generally accepted that layout problems are NP-complete, and their time complexity is upper-bounded by exponential functions. This means that such problems cannot be solved with definite optimality in a reasonable amount of time. Therefore, the focus should be the

formulation of efficient heuristics that seek for near-optimal solutions. Evolutionary computing methods have the potential to tackle such complex design problems while expanding opportunities for emergence and creativity.

In our research, we address two distinguishing characteristics of architectural design and SLD. The first is the privilege that the architects place upon architectural form. Typically form addresses higher-order architectural qualities such as esthetics, meaning, context or performance, and therefore precedes the design of its layout configurations. The second characteristic is the importance of divergent thinking and working with multiple alternatives. Similarly in architectural design, architects explore not one but a number of design alternatives simultaneously until a complete understanding of the design context is attained. This means that SLD design tools should be able to deal with arbitrary building forms that architects propose. Quantitative exploration of the layout solutions of a number of different building forms can help benchmark them against each other and aid the selection of the most optimal building forms and layouts. Therefore, it is important to be able to operationalize a heuristic for SLD that tackles arbitrary building forms proposed by the architects.

This paper presents a novel approach to the multi-floor, unequal-area 3D space layout problem. Evolutionary Architectural Space layout Explorer (EASE) is a design tool that facilitates the generation and optimization of 3D space layouts. Within EASE, layouts are generated by a novel heuristics named Precedence-Based Layout Configuration Heuristics (P-LCH) that can satisfy hard constraints of space overlaps and empty areas. Generated layouts are evaluated by a number of constraints that quantify the size, geometry, placement and topology

E-mail address: ipekg@metu.edu.tr.

relations. Evolutionary algorithms (EA) then facilitate the generate/evaluate cycle to improve individuals' fitness by crossover, mutation and repair operators. EASE is tested on a building design case using the form alternatives for a given architectural brief. We comparatively evaluate the layout performances of these forms based on empirical and quantitative data. We present metrics to describe different building forms to be able to find their correlations with constraints. Then we discuss the convergence characteristics of EASE. Finally, the effect of different EA parameters on the performance of EASE is investigated.

2. Concepts for sld support

2.1. Design tools for exploration and exploitation

Design is a creative activity that cannot be solved with certainty, as it resists definitive formulations and lacks objective evaluation criteria [42]. Where the cost of finding an optimal solution is high, designers search for “good enough” solutions that meet the minimum objectives, known as satisficing [48]. By nature, satisficing entails large design spaces and divergent exploration. As it aims to extend the boundary of a design situation to achieve a large and fruitful search space, divergent (exploratory) designer behavior is also associated with design creativity [30]. The ability to simultaneously attend to alternative design threads within such large search spaces is a measure of the frequency of creative leaps during conceptual design [7]. Conversely, *convergent* design narrows and intensifies search towards more promising areas. It invests only in areas of high opportunity through testing and validation. At this phase, uncertainties are resolved, objectives are agreed upon and design variables are identified.

Design is an interplay of *divergence* and *convergence*, where designers engage in a continuous cycle of broadening and narrowing the design space. Well-informed decision-making can be facilitated by

equally encouraging creative variation and rational optimality. Such approaches need to generate and present feedback for a number of design alternatives by assessing the relative impact of design performance parameters. Such quantitative information can be used to compare or benchmark the quality of design alternatives.

2.2. Metaheuristics as design support

A common characteristic of creative, non-routine design is that neither an inherent solution structure nor an a priori problem formulation exists. During design, design formulations constantly change together with the designer's understanding of the problem. As requirements cannot be definitively and exhaustively defined, a direct operational strategy to find a good solution (heuristics) isn't available either. Metaheuristics are suitable for design, or “I know it when I see it” type of problems, as can evaluate a candidate solution once it is instantiated [37]. Metaheuristic methods comprise a class of upper-level approximate methods that aim efficient search space exploration within discontinuous, multimodal solution spaces.

Evolutionary algorithm (EA) is a metaheuristic method that uses principles of biological evolution, wherein successive generations of design instances evolve by means of recombination and mutation operators. As a high-level decision support technique, it can support multi-variate design problems with multimodal and discontinuous design spaces [16]. We support that EA can address creative design by maintaining a balanced amount of divergence for novelty and convergence for utility. EA's micro-level bottom-up organizational principles can motivate macro-level creativity [46]. As such they can facilitate design experimentation and design discovery and eventually contribute to new architectural values that initially remain unnoticed and unexplored.

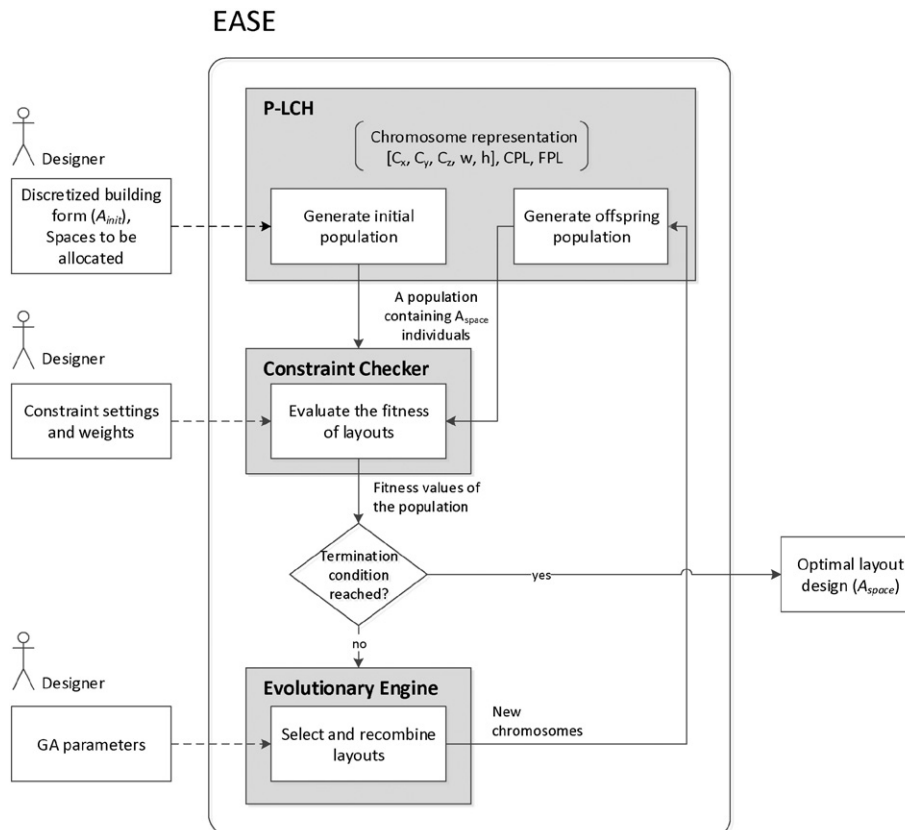


Fig. 1. EASE modules.

Table 1
Nomenclature.

A_{init}	The initial 3D boolean array that represents a discretized building
A_{space}	The 3D array that represents a spatial configuration generated by EASE. The matrix maps the space indices to the corresponding voxels.
$depth_b, height_b, width_b$	The width, height and length values of the of the building's minimum axis-aligned bounding box. This also represents the dimensions of A_{init} and A_{space}
S_i	Space
P_i	The initial rectangular prism of S_i
C_x, C_y, C_z	The x, y and z coordinates of the center voxel of P_i
$width_i, depth_i, height_i$	Width, depth and height of P_i
CPL	Collision Precedence List
FPL	Fill Precedence List
L	A building layout represented by A_{space}
N_s	Number of spaces in a building
NR_v^i	Number of required voxels in S_i
NA_v^i	Number of actual voxels in S_i
CoG _i	Center of gravity point of S_i
V_j^j	Voxel j that belongs to S_i
cor_{max}	The user-defined maximum number of corners for all spaces
cor_i	The actual number of corners of S_i
cnv_i	The total number of concavities of S_i
$facade_i$	The required façade direction for S_i (North, South, East or West)
flo_i	The required floor for S_i
V_{fac}^i	Number of voxels belonging to S_i that are aligned to $facade_i$
V_{flo}^i	Number of voxels belonging to S_i that are on flo_i
rw_i, rh_i, rl_i	The user-defined width, height and length of the AABB that S_i needs to fit in
aw_i, ah_i, al_i	The actual width, height and length of the minimum AABB of S_i
A_{topo}	An array that maintains topology information between spaces, where N = neighborhood, S = separating spaces, and empty otherwise.
$rDist(S_i, S_j)$	The rectangular distance between S_i and S_j
$eDist(P_a, P_b)$	The Euclidean distance between points P_a and P_b
$vDist(V_a, flo_k)$	The vertical distance between voxel V_a and flo_k
$face(S_i, S_j)$	The total number of voxel faces that S_i and S_j share
C_{size}	Space size constraint
C_{dim}	Space absolute dimension constraint
$C_{compact}$	Space compactness constraint
C_{jag}	Space jaggedness constraint
C_{convex}	Space convexity constraint
C_{facade}	Façade constraint
C_{floor}	Floor constraint
C_{neigh}	Neighborhood constraint
C_{sep}	Separation constraint

2.3. The space layout design (SLD) problem

2.3.1. Architectural form and space layout

One of the first tasks of architectural design is architectural form-making (architectural massing). Architectural form is important because it defines the building's identity and its impact on the urban environment by composing forms into a meaningful architectural configuration [5]. For architects, form is usually non-negotiable and almost autonomous, as it expresses key architectural qualities such as esthetics, meaning, context, etc. The design of space layout begins once the building form materializes, guided by both the form and the architectural brief. Together with the building form, the space layout is paramount in determining and evaluating the quality of a building design.

2.3.2. Existing computational approaches

The layout problem can be generalized as an assignment problem, which seeks for the assignment of entities to locations while minimizing cost/time or maximizing the satisfaction of the specified requirements. Several approaches such as greedy algorithms, dynamic programming,

mixed integer programming, or best fit decreasing or first fit decreasing algorithms are widely used to solve the assignment problem. However in its generalized form, the assignment problem does not take into account spatial constraints that architectural layout problems tackle. SLD support tools need to address the domain-specific spatial intentions and propose efficient formulations that take into account architectural qualities.

The existing solution strategies addressing SLD can be classified as construction methods and improvement methods [40]. Construction methods work with incomplete solutions, gradually building layouts in a series of steps using heuristics. Examples include inexact heuristics such as greedy algorithms [2,15], exact algorithms such as branch and bound methods [24,31,55] and dynamic programming [44]. However, it might not be possible to extend partial optimality over the totality of the problem. Moreover, additional sub-evaluation functions to evaluate a partial solution are required when assembling subproblems into a global solution. Improvement methods, on the other hand, construct an initial complete solution and gradually improve it by comparing multiple solutions. This means that, there is at least one sub-optimal solution at any given time even if the algorithm fails. Examples are exhaustive search, local search (such as hill climbing), single-solution metaheuristic methods such as tabu search [1,18], simulated annealing [13,19,39] or variable neighborhood search, and population-based metaheuristic methods such as ant colony algorithms [12,51], particle swarm intelligence and evolutionary algorithms (EA).

2.3.3. Sub-heuristics for the design of space layouts

The principles of EA has been widely used in the layout problems for space, facility, plant, manufacturing system and construction site design with single or multiple floors. However, EA operates in a knowledge-lean context and alone cannot support SLD without a sub-heuristic that constructs layouts. In this section we discuss the existing SLD sub-heuristics that are combined with EA.

A common approach is to assign spaces into a building layout using methods of configuration, resulting in emergent building form. As example is graph theoretic approaches that arrange a number of spaces in a planar graph. The building form emerges from the resulting graph as it is converted into its orthogonal geometric dual. Differently, Rosenman and Gero [45] develop a design grammar that arranges polygonal spaces into house plans. In either case, the building's regularity as well as contextual and esthetic value cannot be fully controlled by the designer.

In contrast, we support that architects demand to maintain control over a building's form. Therefore we study the cases in which a fixed building form already exists prior to the layout design. The slicing tree approach is such an example, which acquires spaces by repeatedly subdividing a given area by vertical and horizontal dissections [3,4,6,8,10,23,25,33,47,52,54]. A binary tree is used for representation, where the terminal nodes are the spaces and the internal nodes specify the vertical or horizontal dissection of its children nodes. However, the subdivision approach fails in 3D layouts, non-sliceable floor plans, non-rectangular spatial and building forms.

An alternative approach is to assign entities to the cell(s) of a matrix representation. This assignment can be generalized as a quadratic assignment problem that matches a number of resources with the same number of cells [9,22,36,38,41]. The one-to-many form is a much complex problem as it needs to ensure that spaces are not split and no empty cells remain. Based on this approach, various spatial forms can be grown by means of a design rule schema encoded in the genotype for design transformations [29]. Other studies combine EA with space filling curves (SFC) that define a continuous path that generates spaces on the matrix [17,28,32,53]. Similarly, spaces can grow from seeds placed by the user by a rule-based heuristics [27]. SFC is advantageous to the slicing tree approach in that it doesn't impose shape restrictions on the building or the spaces. However, these approaches fail to generate 3D vertically-continuous spaces that span multiple floors.

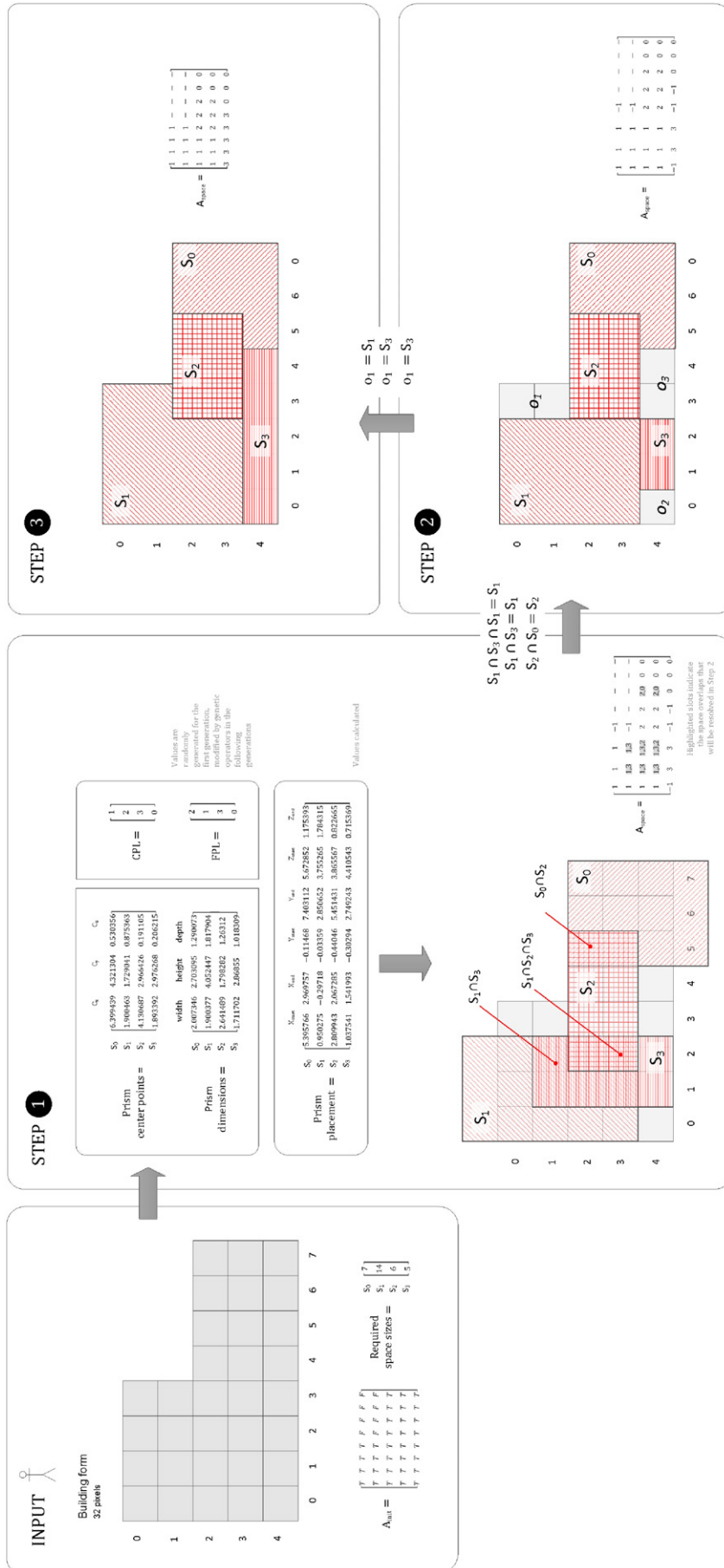


Fig. 2. A diagram of the P-LCH.

INPUT: List of spaces and sizes, Building voxel matrix A_{init} , precedence lists CPL and FPL .

1 Generation of the initial zone form and zone-pixel assignment

```
FOR each  $S_i$  to be allocated in  $A_{init}$  DO
  IF (first generation)
    /* generate random height and depth ratios */
    ratioH ← Random(0, 1)
    ratioD ← Random(0, 1)
    ratioW ← Random(0, 1)
    /* calculate the actual rectangle dimensions */
    width ← ratioW  $\sqrt[3]{\frac{NR_v^i}{ratioH \cdot ratioD \cdot ratioW}}$ 
    height ← ratioH  $\sqrt[3]{\frac{NR_v^i}{ratioH \cdot ratioD \cdot ratioW}}$ 
    depth ← ratioD  $\sqrt[3]{\frac{NR_v^i}{ratioH \cdot ratioD \cdot ratioW}}$ 
    /* generate random z, y and x coordinates */
     $C_z^i$  ← Random (0, bldg.length)
     $C_y^i$  ← Random (0, bldg.length[0])
     $C_x^i$  ← Random (0, bldg.length[1])
    /* the following generations */
  ELSE
    Generate new individuals using genetic operators
  END IF
  /* calculate the start and end points of the prism */
   $X_{start}$  = max(0, (int)Math.floor( $C_x^i$  - depth / 2.0))
   $X_{end}$  = max(0, (int)Math.floor( $C_x^i$  + depth / 2.0))
   $Y_{start}$  = max(0, (int)Math.floor( $C_y^i$  - height / 2.0))
   $Y_{end}$  = max(0, (int)Math.floor( $C_y^i$  + height / 2.0))
   $Z_{start}$  = max(0, (int)Math.floor( $C_z^i$  - width / 2.0))
   $Z_{end}$  = max(0, (int)Math.floor( $C_z^i$  + width / 2.0))
  /* place the prism from upper left to lower right corners */
  drawRect( $X_{start}$ ,  $Y_{start}$ ,  $Z_{start}$ ), ( $X_{end}$ ,  $Y_{end}$ ,  $Z_{end}$ )
End FOR
```

2 Collision (overlap) management

```
FOR each voxel  $V_n$  in the building on which  $S_i \cap S_j \neq \emptyset$ 
  /* let the higher ranking zone keep the voxels */
  IF CPL.indexOf( $S_i$ ) < CPL.indexOf( $S_j$ )
     $V_n$  ←  $S_i$ 
  ELSE
     $V_n$  ←  $S_j$ 
  END IF
```

3 Leftover area management:

```
FOR each empty voxel island o in the building
  calculate the neighboring zones  $S_k, S_l$ 
  /* let the higher ranking zone extend itself */
  IF FPL.indexOf( $S_k$ ) < FPL.indexOf( $S_l$ )
    /* If  $S_k$  hasn't already satisfied its voxel number
    IF  $NA_k^o < NR_k^o$ 
      Extend  $S_k$  to o
    ELSE
      Extend  $S_l$  to o
    END IF
  END IF
```

Fig. 3. P-LCH pseudocode.

A third heuristic allocates fixed or flexible spatial blocks within a given boundary. Dunker et al. use an iterative co-evolutionary method to solve the packing problem of groups of departments [20]. Lee et al. [35] combine EA with graph algorithms for the calculation of the distances between departments through aisles. Bausys et al. determine the detailed layout of units with EA based on lighting, heating, and sizes [11]. Koenig et al. develop a data structure for the hierarchical organization of layout elements to organize larger layout problems into subsets [34]. Rodrigues et al. develop a hybrid evolutionary technique for detailed design with lower-scale architectural elements as parametric objects [43]. In these approaches, the main obstacle is the computation time to detect and avoid collision and empty spaces, and their limited range of simplified spatial forms.

The developed model, EASE, differs from the abovementioned heuristics in that it addresses SLD problems that start with arbitrary orthogonal building forms, within which spaces with arbitrary but meaningful forms are allocated. This is a non-trivial problem because a method to determine these space forms that fulfill requirements including size, absolute dimensions and regularity needs to be formulated. At the same time, it is necessary to ensure the seamless arrangement of spaces as well as the satisfaction of constraints.

3. Evolutionary Architectural Space layout Explorer (EASE)

Evolutionary Architectural Space layout Explorer (EASE) is a design tool for early architectural design that aims to facilitate better informed

Table 2
Constraint formulation.

Constraint category	Constraint name
C.1.1. Space size	<p>C.1.1. Space size: ensures that a space contains the required number of voxels. The ratio of the actual and required number of voxels for each space is aggregated.</p> $C_{size} = \sum_{i=1}^{N_s} \left(\min \left[\frac{NA_i^v}{NR_i^v}, \frac{NR_i^v}{NA_i^v} \right] \right)$
C.1.2. Space geometry constraints: specify a space's acceptable geometries, avoiding extreme forms towards the search for simpler and compact geometries.	<p>C.1.2.1. Absolute dimensions: ensures that a space's dimensions are within user-defined limits. The designer first specifies an axis-aligned bounding box (AABB) for each space S_i, within which it must fit with its height, width and length (rw_i, rh_i, rl_i). Then EASE calculates the number of voxels belonging to S_i that hang outside this AABB in all directions. This number is normalized by the difference between the building and S_i's width, length and height.</p> $C_{dim} = \sum_{i=1}^{N_s} \frac{(\max(0, rw_i - aw_i) + \max(0, rl_i - al_i) + \max(0, rh_i - ah_i))}{((\max(0, depth_b - aw_i)) + \max(0, width_b - al_i) + \max(0, height_b - ah_i))}$ <p>C.1.2.2. Form irregularities</p> <p>C.1.2.2.1. Compactness: ensures the compactness of a space by minimizing its Moment of inertia with respect to the rotation axis defined as the center of gravity (CoG_{<i>i</i>}) of S_i. Then, the distance between CoG_{<i>i</i>} and the voxels of S_i are aggregated and squared. Eventually this value is normalized by the inertia of the building.</p> $C_{compact} = \sum_{i=1}^{N_s} \frac{inertia_i}{inertia_{building}}$ $inertia_i = \sum_{j=1}^{N_v} eDist(CoG_i, CoG_{V_j})^2$ <p>C.1.2.2.2. Jaggedness: ensures a space's geometric regularity by minimizing its number of corners. First, the users define the maximum acceptable corner number (cor_{max}). A space S_i is penalized only if the actual number of corners (cor_i) is above cor_{max}. A voxel V_j^i belonging to S_i with coordinates (x, y, z) is a corner if at least three of its neighboring voxels in the direction of cardinal axes [($x-1, y, z$), ($x+1, y, z$), ($x, y-1, z$), ($x, y+1, z$), ($x, y, z-1$), ($x, y, z+1$)] are of a different space.</p> $C_{jag} = \begin{cases} \sum_{i=1}^{N_s} \max \left[0, \frac{cor_i - cor_{max}}{cor_{max}} \right], & cor_i > cor_{max} \\ 0, & otherwise \end{cases}$ <p>C.1.2.2.3. Convexity: ensures the convexity of a space's form. We first draw axes in cardinal directions through each voxel on $width_b, height_b, length_b$. If an axis cuts through S_i more than once, this means that S_i is concave on that voxel on that direction. We normalize a space's convexity by its number of voxels, then aggregate all the spaces.</p> $C_{convex} = \sum_{i=1}^{N_s} \frac{cnv_i}{NA_i^v}$
C.1.3. Space placement constraints: ensure that the specified spaces are assigned to designated parts of the building.	<p>C.1.3.1. Façade preference: maximizes a space's number of voxels facing towards a given building façade $facade_i$ (N, S, E or W). We first count V_{fac}^i (the number of voxels of S_i that are aligned to $facade_i$). For each voxel V_j^i belonging to S_i with coordinates (x, y, z), we iterate through its horizontal neighbors, and apply the following rule: V_j^i is aligned to: <i>North</i> if ($x, y+1, z$) voxel is empty <i>South</i> if ($x, y-1, z$) voxel is empty <i>East</i> if ($x+1, y, z$) voxel is empty <i>West</i> if ($x-1, y, z$) voxel is empty To normalize this value, we first invert it, and finally ensure its maximization by subtracting in from 1.</p> $C_{facade} = \sum_{i=1}^{N_s} 1 - \frac{1}{1 + V_{fac}^i}$ <p>C.1.3.2. Floor preference: places a space S_i on a user-defined floor flo_i. For this, EASE aggregates for each S_i the distance of each voxel V_j^i to flo_i. This value is normalized by S_i's number of voxels.</p> $C_{floor} = \sum_{i=1}^{N_s} \sum_{j=0}^m \frac{vDist(V_j^i, flo_i)}{NA_i^v}$
C.2. Space topology	<p>C.2.1. Neighborhood: ensures that two spaces are placed next to each other by maximizing their geometric interface. For all the space pairs in M_{topo} that are specified as neighbors (N), the number of voxel faces they share is computed. To normalize this value, we first invert it, and finally ensure its maximization by subtracting it from 1.</p> $C_{neigh} = \begin{cases} \sum_{i=1}^{N-1} \sum_{j=i+1}^N 1 - \frac{1}{face(S_i, S_j) + 1}, & A_{topo}(i, j) = N \\ 0, & otherwise \end{cases}$ <p>C.2.2. Separation: ensures that given two spaces are physically separated as much as possible. For all the space pairs in M_{topo} specified to be separated, the rectangular distance between them is computed. To normalize this value, we first invert it, and finally ensure its maximization by subtracting in from 1.</p> $C_{sep} = \begin{cases} \sum_{i=1}^{N-1} \sum_{j=i+1}^N 1 - \frac{1}{rDist(S_i, S_j) + 1}, & A_{topo}(i, j) = S \\ 0, & otherwise \end{cases}$

decision-making under spatial constraints using evolutionary optimization. EASE tackles multi-floor, unequal-area layout problems starting with arbitrary a priori architectural forms. EASE doesn't aim to generate a finalized layout, but a number of low-resolution solutions that provide insight into the solution space and initiate further exploration. This quantitative inquiry is aimed to facilitate the designers' qualitative and

comparative analysis of layout performances. As such, it offers an alternative to fully automated optimization approaches that seeks to find a single optimal solution.

The three activities of computational design synthesis are generation, evaluation, and guidance. EASE addresses these activities in three integrated modules (Fig. 1). Precedence-based Layout Configuration

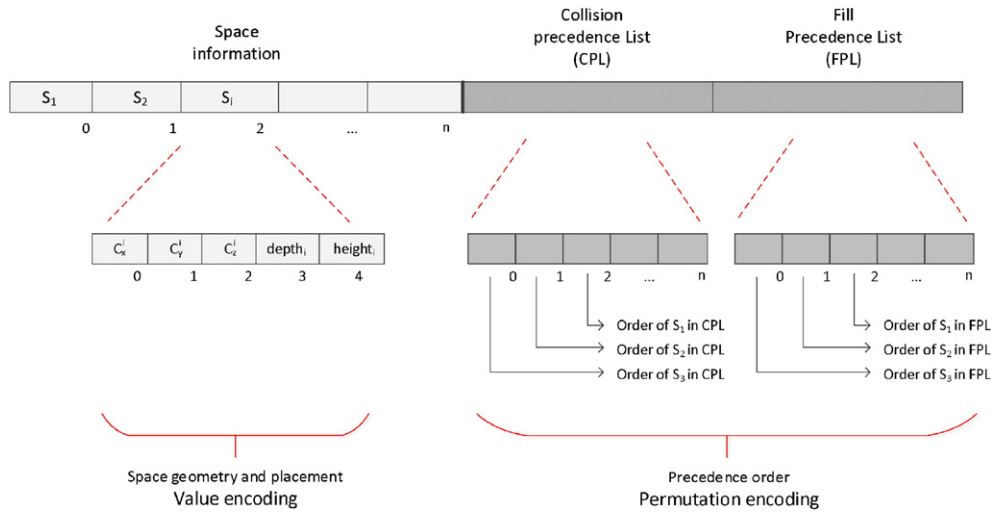


Fig. 4. Chromosome representation.

Heuristics (P-LCH) is a novel heuristic that generates layouts without overlaps or empty spaces. The Constraint Checker quantifies the fitness of each generated layout by means of constraint penalties. Based on the fitness values, the Evolutionary Engine provides feedback to P-LCH through recombination and mutation to improve solutions. Input data is a discretized building form, a list of spaces, constraint settings, weights and EA parameters. Table 1 summarizes the symbols used to formulate EASE.

3.1. Precedence-based layout configuration heuristics (P-LCH)

Precedence-Based Layout Configuration Heuristics (P-LCH) is novel method that generates valid space layouts that avoid overlap conflicts or empty areas within given building forms. First, the building form needs to be discretized into a number of equal-area voxels by the users. This form is represented as a 3D Boolean matrix (A_{init}), following the same scheme as the building's voxel structure. The spaces in the architectural brief need to be discretized in the same manner, such that

the total number of spatial units is equal to the total number of voxels. This reduces both the dimensions of the search space and the computational complexity by avoiding expensive geometric overlap detection operations. As a result, the layout problem is transformed into an assignment problem, where each space with n number of units are to be matched with n number of voxels. As the spaces are assigned to voxels, the space indices that occupy the voxels are registered in a second 3D matrix (A_{space}) with the same voxel structure as A_{init} .

P-LCH takes as input the building representation (A_{init}) and the space sizes. Thereon, three subtasks are repeated for each layout instance, as discussed below. In Fig. 2, we demonstrate P-LCH on a simple example that assigns four spaces in a single-story building. The pseudocode can be seen in Fig. 3.

Step 1 concerns the generation of initial 3D spatial forms and their assignment to building voxels (Fig. 2, Step 1). P-LCH distinguishes between the layouts of the very first generation and the remaining generations.

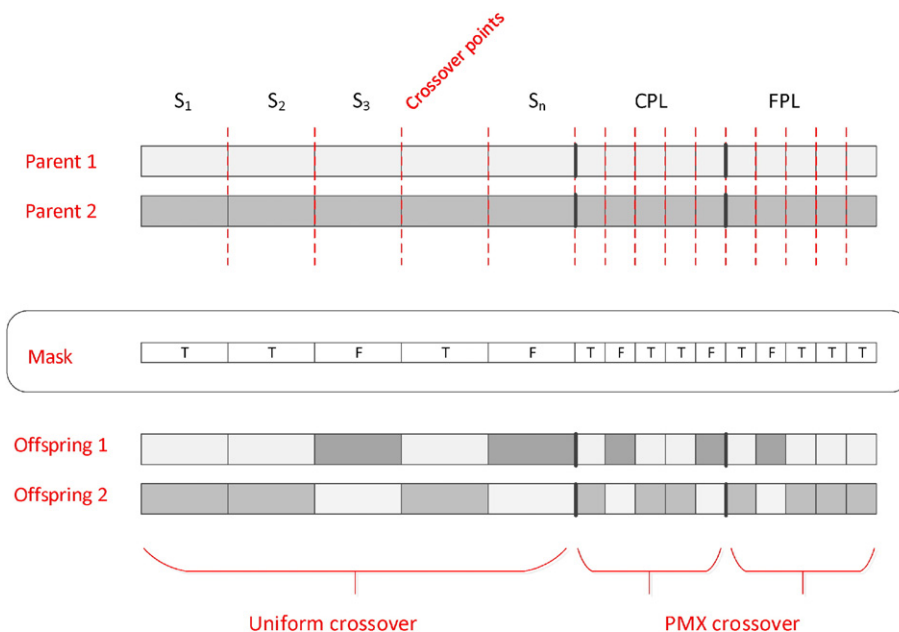


Fig. 5. Crossover points and types.

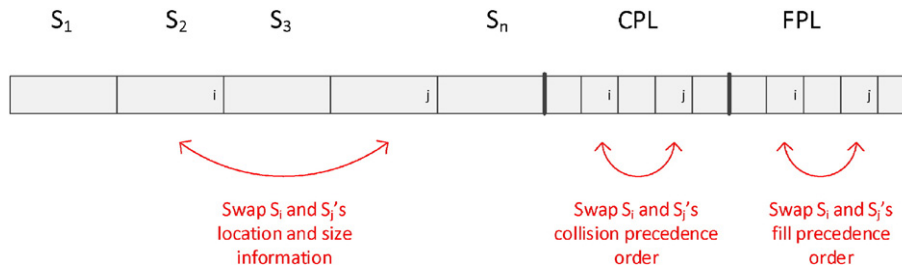


Fig. 6. Swap repair operation.

First generation: P-LCH first generates for each space S_i a rectangular prism P_i that aims to approximate its total number of voxels (NR_v^i) as much as possible. The prisms' dimensions are determined by generating random values over $[0, 1]$ for their width ($ratioW$), depth ($ratioD$) and height ($ratioH$) ratios (Fig. 3, line 5–7). Then the prisms' actual $width_i$ is calculated by multiplying $ratioW$ with the cube root of the voxel size divided by its $ratioD$ and $ratioH$. The same is calculated for $height_i$ and $depth_i$ in the same manner (Fig. 3, line 9–11). To place P_i in the building, first a center point (C_x^i, C_y^i, C_z^i) is randomly generated (Fig. 3, line 13–15) where:

$$0 \leq C_x^i < depth_b \text{ and } 0 \leq C_y^i < width_b \text{ and } 0 \leq C_z^i < height_b$$

The building voxels that will be occupied by P_i in A_{init} are calculated from the upper-left to the lower-right voxels (Fig. 3, line 21–26). If the prism overflows the building, it is trimmed off. Eventually, the calculated spaces are assigned to the corresponding indices of A_{space} . In the resulting configuration, an index can be occupied by multiple spaces, and/or some indices may remain empty. This conflict is resolved in the next two steps by using two permutation lists, Collision Precedence List (CPL) and Fill Precedence List (FPL), which maintain the precedence of each space in case of collision and empty voxels. The higher ranking space has the priority to preserve the overlapping voxels or to annex the empty neighboring voxels. The values of CPL and FPL are generated randomly for the first generation.

Following generations: Recombination operators are used to generate and assign prisms to voxels, and modify the CPL and FPL (see Section 3.3.2 and 3.3.3).

Step 2 eliminates the collisions (overlap) between spaces, if any, by determining which space preserves those voxels (Fig. 2, Step 2). For all voxels in A_{space} , if there are multiple spaces assigned, then CPL is referred to. Only the highest ranking space is allowed to keep the voxel island. Following, the remaining space(s) withdraw from this voxel island.

Step 3 determines which space extends itself towards the unoccupied areas, if any, remaining in the building (Fig. 2, Step 3). First, the voxel islands (adjacent voxel groups) that don't have spaces assigned to them are detected with flood fill algorithm. For each such island, all the neighboring spaces that border these voxels are calculated. Amongst

these, we eliminate the spaces that have already reached their required size ($NR_v^i < NA_v^i$). For all the remaining spaces, the highest ranking in FPL extends itself onto the island. As a result, all the empty voxels are occupied by the neighboring spaces.

3.2. Constraint checker

In the previous step, P-LCH generates a generation of valid layouts that have no overlaps or empty spaces. However, it doesn't guarantee the satisfaction of spatial requirements. The Constraint Checker quantifies the fitness of a layout by means of penalizing constraints that evaluate (C.1) singular spaces and (C.2) pairs of spaces (Table 2). For C.1, each space's penalty is separately calculated and then aggregated into the constraint's total penalty score. For C.2, only the penalties of the indicated space pairs in A_{topo} are calculated and aggregated. To allow the prioritization of some constraints over others, the calculated penalties are multiplied by user-defined weight values (w). Finally, weighted penalty values are aggregated into a single fitness function that is minimized towards zero. The eventual objective function can be formulated as below, where $f(L)$ is the function to be minimized for a given layout L , and w is the user-defined penalty weights.

$$\begin{aligned} \text{minimize } f(L) = & \sum_{i=1}^{N_s} (w_{size} C_{size} + w_{dim} C_{dim} + w_{compact} C_{compact} \\ & + w_{jag} C_{jag} + w_{convex} C_{convex} + w_{facade} C_{facade} + w_{floor} C_{floor}) \\ & + \sum_{j=1}^{A_{topo, length}} (w_{neigh} C_{neigh} + w_{sep} C_{sep}) \end{aligned}$$

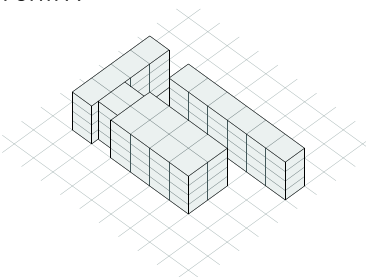
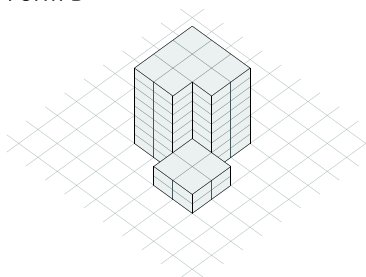
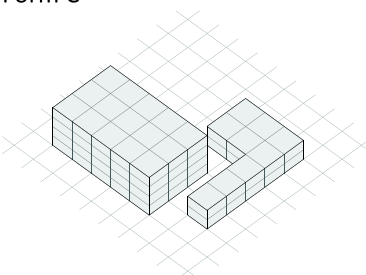
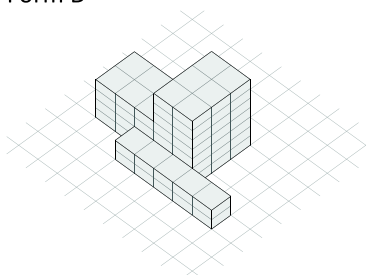
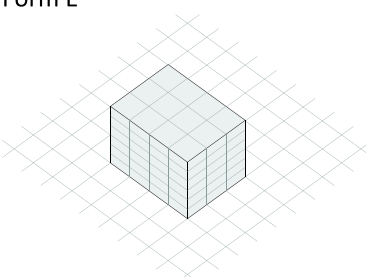
EASE acknowledges that design problems are inherently tolerant to violating constraints as long as the purpose of design is not severely compromised. Therefore, it seeks to find trade-off solutions in which constraints are satisfied at the highest possible level, if not fully. To this end, EASE minimizes the *difference* between the required and actual constraint values. We formalize the constraint calculation procedures such that they are all subject to minimization. Each constraint calculation procedure takes as input a property Pr_i of the space or space pair under evaluation. To avoid different orders of magnitude, we normalize the constraints' penalty values to the (0–1) range using the following rules. For a property Pr_i to be minimized, if a user-defined target value exists (as in c_{size} , c_{dim} and c_{jag}), then we normalize by calculating for each space the ratio between the actual value Pr_i and this target value. If not, (as in $c_{compact}$, c_{convex} and c_{floor}), we normalize the constraint value with other properties of the building or the space. Alternatively, when constraints deal with properties that are to be maximized (as in c_{facade} , c_{neigh} and c_{sep}), we convert it to a normalized value that is also subject to minimization, using the general form $1 - (1/(1 + Pr_i))$.

The Constraint Checker quantifies each layout's fitness by introducing the constraints into a single objective function by means of penalties. In the next step, individuals will be selected and recombined to generate new individuals to improve the individuals' fitness.

Table 3
Library spaces.

Space		Surface area (m ²)
S ₁	Reading 1	1300
S ₂	Reading 2	900
S ₃	Books	900
S ₄	Administration	650
S ₅	Café 1	500
S ₆	Working	1500
S ₇	Conference room	75
S ₈	Café 2	75
	Total	5900

Table 4
Form alternatives.

<p>Form A</p>  <p>Space sizes $NR_v^1=16$ $NR_v^2=11$ $NR_v^3=11$ $NR_v^4=8$ $NR_v^5=6$ $NR_v^6=18$ $NR_v^7=1$ $NR_v^8=1$ Total=72</p>	<p>Form B</p>  <p>Space sizes $NR_v^1=18$ $NR_v^2=12$ $NR_v^3=12$ $NR_v^4=10$ $NR_v^5=6$ $NR_v^6=18$ $NR_v^7=2$ $NR_v^8=2$ Total=80</p>
<p>Form C</p>  <p>Space sizes $NR_v^1=17$ $NR_v^2=12$ $NR_v^3=12$ $NR_v^4=10$ $NR_v^5=8$ $NR_v^6=16$ $NR_v^7=1$ $NR_v^8=2$ Total=78</p>	<p>Form D</p>  <p>Space sizes $NR_v^1=17$ $NR_v^2=12$ $NR_v^3=12$ $NR_v^4=8$ $NR_v^5=6$ $NR_v^6=18$ $NR_v^7=1$ $NR_v^8=2$ Total=76</p>
<p>Form E</p>  <p>Space sizes $NR_v^1=16$ $NR_v^2=11$ $NR_v^3=11$ $NR_v^4=8$ $NR_v^5=6$ $NR_v^6=18$ $NR_v^7=1$ $NR_v^8=1$ Total=72</p>	

4.1. Evolutionary engine

4.1.1. Chromosome representation

In EA the effectiveness of recombination operators is largely due to the chromosome representation. In EASE the chromosomes don't maintain the exact building layout, but only the spatial information dealt by P-LCH. In the chromosome, we use value encoding to represent the information about the spaces' rectangular prisms, and permutation encoding for the two precedence lists. (Fig. 4). In value encoding, a vector of values representing each space's initial rectangular prism's (P_i) dimensions is stored: the center point (C_x^i, C_y^i, C_z^i), followed by $depth_i$ and $height_i$ (width is calculated only when required). For each allele, we use floating-point numbers in the genotype, but in the

phenotype we map this value to its floor (the largest previous integer) to match the indices of A_{space} .

The two precedence lists are represented by permutation lists that maintain the indices of the spaces to be allocated in A_{space} . The spaces' ordering in these lists specify their precedence to gain voxels in case of overlap and empty area conflicts.

The chromosome representation of EASE indirectly encodes a layout design, as opposed to more straightforward approaches, i.e. one-to-one mapping of voxels to the spaces. In biological evolution, this principle is known as the *genotype/phenotype distinction*, which states that an individual's genetic information and its observable characteristics need not be the same. This may seem disadvantageous from the point that a valid layout (the phenotype) can be instantiated only by means

Table 5
Façade, floor and absolute dimension constraint values.

	C_{facade}	C_{floor}	C_{dim}		
			w	d	h
S_1			5	5	4
S_2			5	5	4
S_3			5	5	4
S_4			5	5	3
S_5	South	3	5	5	1
S_6			5	5	5
S_7	North		1	1	1
S_8		0	1	1	1

Table 6
 $A_{topo} \cdot C_{neigh}$ (N) and c_{sep} (S) constraint values.

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
S_1	-	-	-	-	-	-	-	-
S_2	-	-	-	N	-	-	-	-
S_3	-	-	-	N	-	-	-	-
S_4	-	N	N	-	N	N	-	-
S_5	-	-	-	N	-	-	N	S
S_6	-	-	-	N	-	-	-	-
S_7	-	-	-	-	N	-	-	-
S_8	-	-	-	-	S	-	-	-

Table 7
Constraint weights.

Constraint	Weight
C.1.1. c_{size}	200
C.1.2.1. c_{dim}	250
C.1.2.2.1. $c_{compact}$	10
C.1.2.2.2. c_{jag}	750
C.1.2.2.3. c_{convex}	10
C.1.3.1. c_{facade}	20
C.1.3.2. c_{floor}	80
C.2.1. c_{neigh}	30
C.2.2. c_{sep}	1

of the P-LCH algorithm. But at the same time, the high level of abstraction entailed by this distinction is advantageous, as it condenses the chromosome into merely $N_s \times 7$ genes. An alternative representation, which maps each voxel and to the spaces one-to-one, would be significantly longer and the recombination operators would not ensure the connectivity of a space's voxels.

4.1.2. Selection

To ensure genetic diversity and high fitness in the selection of the individuals that will contribute to the following generation, we implement operations for elitism, crossover, mutation and repair. First we sort the current generation in descending fitness values. The elitist strategy copies the best individuals in the sorted list into the new generation with a rate of P_e so to preserve them in the population. For the remaining individuals, we loosen the selection pressure and perform a selection scheme that is random but biased towards fitter individuals. We generate a random value rnd ($0 \leq rnd < 1$), and select the individuals with the index ($rnd^2 \times N_p$) in the sorted list with a rate of P_c , where N_p is the population size. Mutation is performed randomly with the rate P_m , which is to be multiplied by the number of generations where no fitness improvement is observed (N_{ni}). This way, it increases genetic variation when EASE starts to stagnate. Similarly, repair operators randomly select individuals with the rate of P_r , but decides whether to repair the selected chromosome or not based on its satisfaction of certain constraints (see Section 3.3.3). The number of individuals to be selected for each strategy is as follows:

$$\text{The number of elites} = P_e \times N_p$$

$$\text{The number of individuals to be crossed-over} = P_c \times N_p$$

$$\text{The number of individuals to be mutated: } P_m \times N_{ni}$$

$$\text{The number of individuals to be repaired (initial selection)} = P_r \times N_p$$

4.1.3. Reproduction

4.1.3.1. Crossover. According to the building block hypothesis, an efficient EA assembles solutions by combining short, low-order and highly fit schemas with strong non-linear interdependencies to form fitter, higher-order schemas [26]. Therefore, the crossover operation should accordingly identify the dependent and independent blocks in the chromosome. IN EASE, a space's information (size and location combined) is strongly interdependent, while CPL and FPL operate independently of

spatial information. Therefore, we select the crossover points such that space information remains intact as a building block (Fig. 5).

When selecting the crossover method, problems regarding the loss of diversity and premature convergence play a role. We implement uniform crossover, which provides a rather disruptive but also efficient recombination technique for exploratory search [49]. First, a boolean mask with the same length as the chromosome is created, encoding True with a probability P_o of 70%. This mask indicates which parent will provide its alleles to its offspring. The allele is taken from the first parent if the mask suggests True, and from the second parent if otherwise. Here we can change the level of disruption by parameterizing P_o .

For ordered lists, a different method is necessary to prevent invalid chromosomes with repeating alleles [50]. For CPL and FPL, special recombination operators that maintain and combine parents' adjacency information are implemented. We use Partial Mapping Crossover (PMX) due to its emphasis on absolute position rather than adjacency. PMX generates valid offspring that both inherit parents' alleles while introducing random new traits. PMX selects split points and swaps allele values one by one, immediately followed by swapping the duplicate values in a chromosome and repairing damages as they occur.

Due to our biased selection strategy that privileges fitter individuals, crossover might introduce to the population clones of the parents. As a result, that individual can dominate the population and cripple the genetic diversity. Therefore, after the reproduction operations, we inspect the new generation for clones, and apply random mutation if we find any.

4.1.3.2. Mutation. Mutation in general ensures that new genetic material is introduced into the population and the unexplored areas in the search space are reached. On the chromosome's value encoded part, we apply mutation by replacing alleles with randomly generated values, such that $0 \leq C_x^i < depth_b$ and $0 \leq C_y^j < width_b$ and $0 \leq C_z^k < height_b$, and $1 \leq width_i < rw_i$ and $1 \leq height_i < rh_i$. The mutation of CPL and FPL is performed differently by switching two random gene positions with each other. As such, it increases genetic variation when EASE starts to stagnate.

4.1.3.3. Repair operators. In EASE, we implement repair mechanisms to transform infeasible solutions into feasible ones. Repair mechanisms operate not on the phenotype but the genotype level. Due to the fact that P-LCH can only map the genotype to the phenotype, there is no direct feedback mechanism and therefore we can only indirectly repair a layout. This also means that a repair act only creates an opportunity to improve constraint satisfaction, without any certainty. For this reason, it is similar to the mutation operator that exploits random changes for improvement, but different from it as it is directed towards chromosome parts that are not performing well.

EASE's repair mechanism implements two separate repair operators that modify a chromosome internally. Each repair operator is activated sparingly based on a violated constraint. The first is initiated if the selected individual L consists of two spaces S_i and S_j that have a similar number of voxels, and at the same time perform poorly in terms of c_{neigh} . We mathematically express this condition as follows: Higher swap mutation ratio (SMR) values make sure that only space pairs with approximately the same number of voxels are selected to swap.

$$\left(\min \left[\frac{NR_v^i}{NR_v^j}, \frac{NR_v^j}{NR_v^i} \right] \geq SMR \right) \wedge (S_i \cdot c_{neigh} = 0)$$

If the above condition is satisfied, S_i and S_j 's size and location information as well as their ordering in CPL and FPL are swapped Fig. 6. This swap changes S_i and S_j 's neighboring spaces together with their physical location in the building, generating a random opportunity to improve their c_{neigh} .

The second repair operator is activated when c_{size} of the selected individual

Table 8
EA parameters.

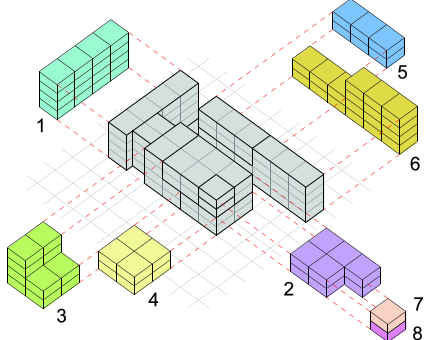
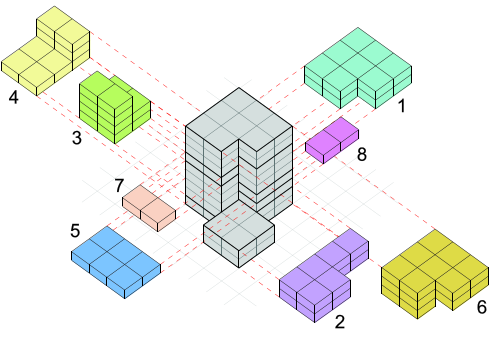
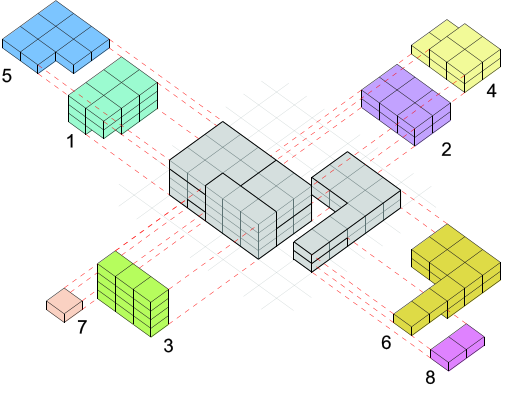
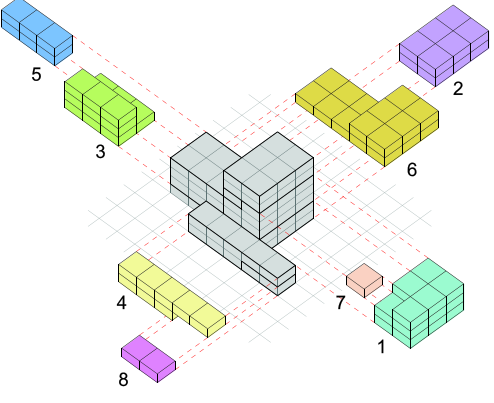
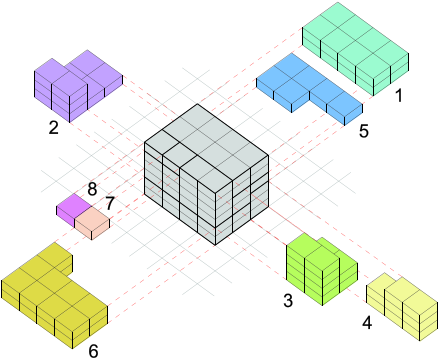
EA parameter		Value
P_c	Crossover rate	0.9
P_e	Elitism rate	0.1
P_m	Mutation rate	0.1% * number of non-improving generations
P_r	Repair rate	0.1% * number of non-improving generations
N_p	Population size	1000 individuals
N_t	Termination condition	300 non-improving generations

Table 9

The fitness values and raw constraint penalties of ten simulations for five building forms. The best individual of each form is highlighted, and visualized in Table 10.

	Constraints	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	Raw avg.	Weig. avg.
Form A	C.1.1. c_{size}	0	0	0	0	0	0.0217803	0	0	0	0.01736111	0.00391414	0.78282828
	C.1.2.1. c_{dim}	0	0	0	0	0	0	0	0	0	0	0	0
	C.1.2.2.1. $c_{compact}$	0.29479578	0.29479578	0.29479578	0.26218709	0.31048254	0.2714372	0.26171132	0.26171132	0.29479578	0.27539945	0.28221121	2.82211205
	C.1.2.2.2. c_{jag}	0.00195313	0.00195313	0.00195313	0.00390625	0.00195313	0	0.00195313	0.00195313	0.00195313	0.00195313	0.00195313	1.46484375
	C.1.2.2.3. c_{convex}	0	0	0	0	0	0	0	0	0	0	0	0
	C.1.3.1. c_{facade}	0.32142857	0.32142857	0.32142857	0.32142857	0.32142857	0.32142857	0.32142857	0.32142857	0.32142857	0.32142857	0.32142857	6.42857143
	C.1.3.2. c_{floor}	0	0	0	0	0	0	0	0	0	0	0	0
	C.2.1. c_{neigh}	0.72083333	0.72083333	0.72685185	0.74166667	0.70601852	0.73333333	0.85185185	0.85185185	0.72685185	0.73333333	0.75134259	22.5402778
	C.2.2. c_{sep}	0.76923077	0.76923077	0.76923077	0.69230769	0.92307692	0.84615385	0.53846154	0.53846154	0.92307692	0.46153846	0.72307692	0.72307692
	FITNESS	33.2356038	33.2356038	33.4161593	34.9224375	33.1018731	36.3451579	36.6045454	36.6045454	33.5700055	36.5811704		34.7617102
	Form B	Constraints	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	Raw avg.
C.1.1. c_{size}		0	0	0	0	0	0.01907895	0	0	0	0	0.00190789	0.38157895
C.1.2.1. c_{dim}		0	0	0	0	0	0	0	0	0	0	0	0
C.1.2.2.1. $c_{compact}$		0.15593668	0.15778364	0.15514512	0.15760774	0.18693931	0.16291719	0.16072999	0.16864556	0.15976253	0.19593961	0.16614074	1.66140737
C.1.2.2.2. c_{jag}		0.00520833	0.00173611	0.00520833	0.00260417	0.00173611	0.00260417	0.00260417	0	0.00434028	0.00520833	0.003125	2.34375
C.1.2.2.3. c_{convex}		0	0	0	0	0	0	0	0	0	0	0	0
C.1.3.1. c_{facade}		0.23809524	0.29166667	0.35	0.29166667	0.32142857	0.23809524	0.23809524	0.375	0.23809524	0.29166667	0.28738095	5.74761905
C.1.3.2. c_{floor}		0	0.00925926	0	0.02777778	0.03472222	0.02777778	0.04861111	0.09722222	0	0.02314815	0.02685185	2.14814815
C.2.1. c_{neigh}		0.64444444	0.67222222	0.53888889	0.52777778	0.61666667	0.74166667	0.6	0.62222222	0.65	0.56944444	0.61833333	18.55
C.2.2. c_{sep}		0.92857143	0.42857143	0.5	0.64285714	0.5	0.78571429	0.92857143	0.64285714	0.92857143	0.71428571	0.7	0.7
FITNESS		30.4894263	30.0492319	29.1243679	28.0609484	31.3778257	37.4179276	31.13979	36.2737572	30.0433099	31.3484503		31.5325035
Form C	Constraints	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	Raw avg.	Weig. avg.
	C.1.1. c_{size}	0.02859477	0.0316585	0.01475694	0	0.01475694	0	0	0	0.01475694	0.01736111	0.01218852	2.43770425
	C.1.2.1. c_{dim}	0	0	0	0	0	0	0	0	0	0	0	0
	C.1.2.2.1. $c_{compact}$	0.20187542	0.18654326	0.17173476	0.17105512	0.16610851	0.23344562	0.16040542	0.17383607	0.18450547	0.18376667	0.18332763	1.83327632
	C.1.2.2.2. c_{jag}	0.00297619	0	0.00446429	0.00744048	0.0014881	0.00892857	0.00446429	0.00892857	0.00372024	0.0014881	0.00438988	3.29241071
	C.1.2.2.3. c_{convex}	0.01190476	0	0	0	0	0	0	0	0	0	0.00119048	0.01190476
	C.1.3.1. c_{facade}	0.30555556	0.35	0.30555556	0.30555556	0.35	0.30555556	0.375	0.35	0.35	0.35	0.33472222	6.69444444
	C.1.3.2. c_{floor}	0	0	0	0	0	0	0	0.03125	0	0	0.003125	0.25
	C.2.1. c_{neigh}	0.38055556	0.40833333	0.47083333	0.36527778	0.56666667	0.33955026	0.38492063	0.36732804	0.43214286	0.47685185	0.41924603	12.577381
	C.2.2. c_{sep}	0.78571429	0.71428571	0.78571429	0.78571429	0.64285714	0.92857143	0.71428571	0.64285714	0.78571429	0.85714286	0.76428571	0.76428571
	FITNESS	28.402391	28.1614177	29.0387762	25.1460671	30.3714025	26.2570753	24.7141733	29.5974877	28.3366222	28.5886588		27.8614072
Form D	Constraints	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	Raw avg.	Weig. avg.
	C.1.1. c_{size}	0	0	0	0	0	0	0	0	0	0	0	0
	C.1.2.1. c_{dim}	0	0	0	0	0	0	0	0	0	0	0	0
	C.1.2.2.1. $c_{compact}$	0.24429632	0.19742227	0.22939955	0.21919095	0.180609	0.19761316	0.18880948	0.1804792	0.21293751	0.17199621	0.20227536	2.02275365
	C.1.2.2.2. c_{jag}	0.00510204	0.00510204	0.00892857	0.00255102	0.00637755	0.00255102	0.00510204	0.00318878	0.00255102	0.00255102	0.00440051	3.30038265
	C.1.2.2.3. c_{convex}	0	0	0	0	0	0	0	0	0	0	0	0
	C.1.3.1. c_{facade}	0.32142857	0.32142857	0.375	0.375	0.32142857	0.32142857	0.32142857	0.375	0.375	0.32142857	0.34285714	6.85714286
	C.1.3.2. c_{floor}	0	0	0.01785714	0	0	0	0	0	0.01785714	0.01785714	0.00535714	0.42857143
	C.2.1. c_{neigh}	0.49074074	0.42222222	0.28492063	0.41269841	0.35555556	0.42222222	0.53769841	0.37777778	0.25833333	0.4	0.39621693	11.8865079
	C.2.2. c_{sep}	0.73333333	0.8	0.73333333	0.6	0.66666667	0.73333333	0.93333333	0.8	0.6	0.93333333	0.75333333	0.75333333
	FITNESS	28.1536208	25.6959914	27.1999479	24.5861272	24.3511581	23.7179683	29.2074826	23.829707	21.3212118	24.4237036		25.2486919
Form E	Constraints	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	Raw avg.	Weig. avg.
	C.1.1. c_{size}	0.0217803	0	0.0217803	0.0217803	0.0217803	0	0.0217803	0.0217803	0	0.01822917	0.0148911	2.9782197
	C.1.2.1. c_{dim}	0	0	0	0	0	0	0	0	0	0	0	0
	C.1.2.2.1. $c_{compact}$	0.29116915	0.27527137	0.24614428	0.2431592	0.2778607	0.26633876	0.26927861	0.25062189	0.25763229	0.24677748	0.26242537	2.62425373
	C.1.2.2.2. c_{jag}	0	0	0	0	0	0.00347222	0	0	0.00347222	0	0.00069444	0.52083333
	C.1.2.2.3. c_{convex}	0	0	0	0	0	0	0	0	0	0	0	0
	C.1.3.1. c_{facade}	0.35	0.375	0.375	0.32142857	0.32142857	0.32142857	0.375	0.32142857	0.41666667	0.375	0.3552381	7.1047619
	C.1.3.2. c_{floor}	0	0	0.02083333	0	0	0.01388889	0	0.01388889	0	0	0.00486111	0.38888889
	C.2.1. c_{neigh}	0.27103175	0.45	0.35277778	0.39047619	0.30714286	0.39351852	0.32222222	0.39444444	0.32222222	0.31111111	0.35149471	10.5448413
	C.2.2. c_{sep}	0.4	0.7	0.4	0.4	0.7	0.9	0.7	0.6	0.4	0.6	0.58	0.58
	FITNESS	22.7987045	26.2065463	26.9675034	25.3305098	23.4775247	25.5127924	24.9155133	26.8352954	23.5804896	23.5469414		24.7417988

Table 10
The best layout of each form alternative.

<p>Form A layout 5</p> 	<p>Form B layout 4</p> 
<p>Fitness: 33.1018730853989 Faults: $c_{dim}: S_6 (+1x)$ $c_{jag}: S_2 (14)$ $c_{neigh}: S_4 \text{ and } S_6, S_3 \text{ and } S_1, S_3 \text{ and } S_5, S_7 \text{ and } S_1$</p>	<p>Fitness: 28.06094843 Faults: $c_{floor}: S_8$ $c_{jag}: S_6 (15)$ $c_{neigh}: S_4 \text{ and } S_6.$</p>
<p>Form C layout 7</p> 	<p>Form D layout 6</p> 
<p>Fitness: 24.71417326 Faults: $c_{jag}: S_1 (14) \text{ and } S_6 (16)$ $c_{neigh}: S_4 \text{ and } S_6.$</p>	<p>Fitness: 23.71796831 Faults: $c_{jag}: S_1 (14) \text{ and } S_6 (14)$ $c_{neigh}: S_3 \text{ and } S_2$</p>
<p>Form E layout 1</p> 	
<p>Fitness: 22.79870453 Faults: $c_{size}: S_1 (-1) \text{ and } S_2 (+1)$</p>	

L is low. In this case, first the difference d_i between the required and actual number of voxels NR_v^i and NA_v^i for each space S_i in L is calculated. Following, we select either CPL or FPL with a probability of 50%, and entirely reorder the selected one, such that the spaces with higher d values are assigned to the higher indices in the selected list. As a result, spaces that fall short of voxels will gain higher precedence to occupy more voxels during for collision or empty areas. The reason we reorder only one of the lists is that the combined effect would be severely biased towards the currently unsuccessful spaces.

4.1.4. Termination condition

EASE is terminated when it stagnates, or if the highest fitness value remains the same for a predetermined number of successive generations (N_{ni}).

5. Design explorations with ease

In this section, we evaluate EASE by testing it in the actual design of a library building. We implement the EASE model into a prototype software application using Java. Using this prototype, we first empirically evaluate if EASE can generate well-performing spatial layouts in different building forms provided by the architects. Then we investigate the

convergence characteristics of EASE. Finally we study the effect of different EA parameter values on EASE's performance.

5.1. Form explorations

We evaluate EASE's capacity to generate layouts within arbitrary forms. For this, we study the design of a library building being designed by an architectural firm. The brief specifies a total surface area of approximately 5900 m², and consists of 8 spaces (Table 3). We asked the architects to provide five form alternatives with approximately the same size (Table 4). First, each form and the spaces in the brief are discretized into units varying slightly between 72 and 80 depending on the building size. The façade, floor, absolute dimension and topology constraints are specified by the designers (Tables 5, 6), and the maximum corner constraint is set to 12. Finally, the constraint weights (Table 7) and EA parameters (Table 8) are quantified. These parameters, too, are determined in a trial-and-error fashion after experimentation. Neighborhood was the most important constraint to satisfy, so its weight was intentionally kept high. A discussion on the effect of other parameter values can be found in Section 3.3.

To evaluate EASE's performance, we performed ten simulations for each form, and plotted the raw and weighted constraint penalties together their fitness scores in Table 9. We also visualize the best layout

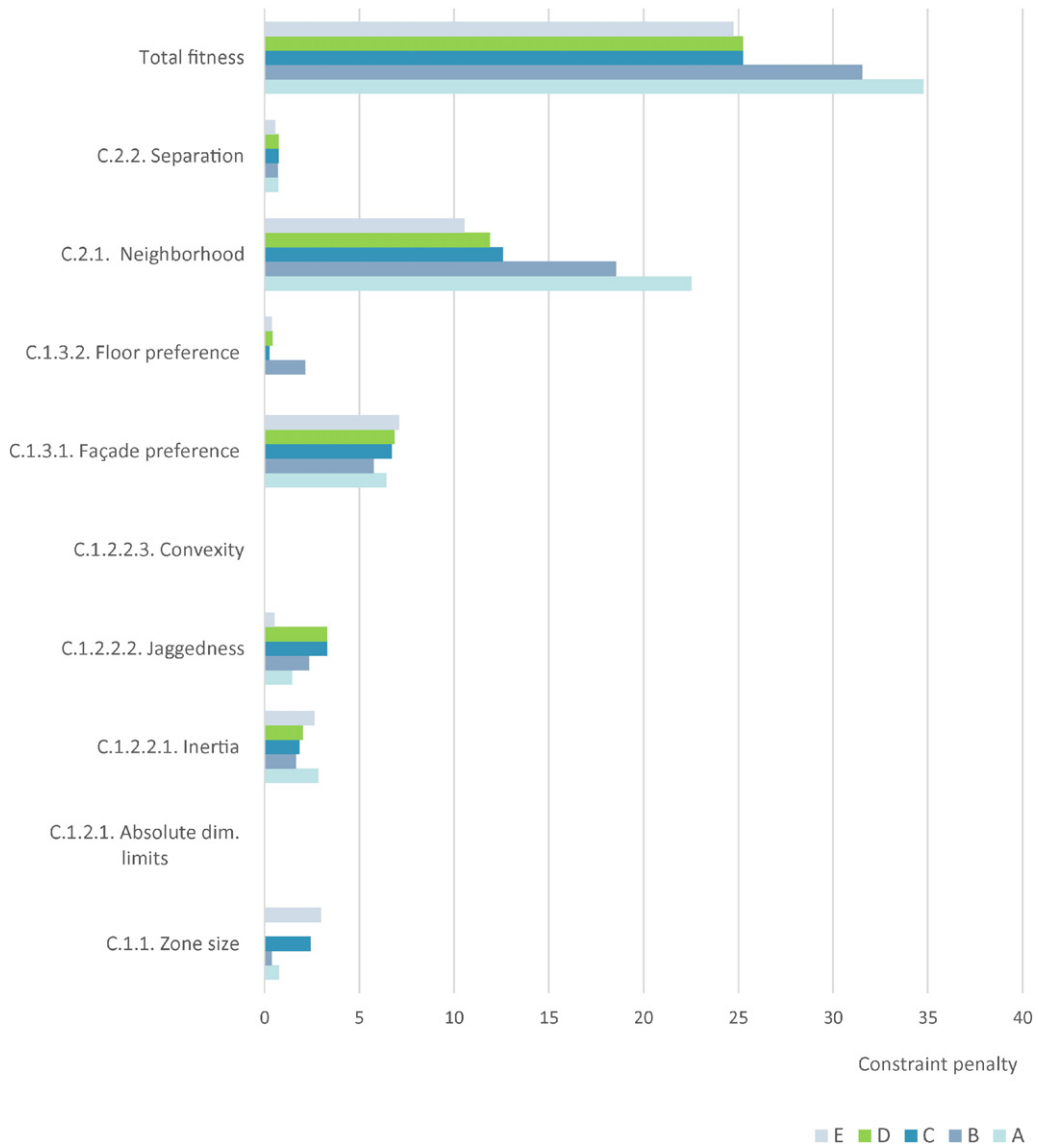


Fig.7. The average fitness and weighted constraint penalty values of the ten layout alternatives of buildings A,B,C,D and E.

Table 11
The SA/V and connectivity (CON) values of buildings A,B,C,D and E.

Building	Surface area (voxel faces)	Volume (voxels)	Surface/volume ratio (SA/V)	Connectivity (CON)
A5	164	72	2.277778	264
B4	138	80	1.725	300
C7	144	78	1.846154	324
D6	124	76	1.631579	302
E1	108	72	1.5	328

alternative of each building form in Table 10. As P-LCH already eliminated overlaps and empty voxels, all layouts can be said to be acceptable. However, the constraint penalties and fitness values vary from building to building (Fig. 7). To understand the influence of form on constraints and fitness (the latter two are already quantified in Tables 9 and 10), we introduce and calculate two metrics regarding building form: compactness and connectivity (Table 11). We quantify building compactness by the ratio of a building's surface area to its volume (SA/V). To quantify building connectivity (CON), we calculate and aggregate for each voxel the number of its neighbors in the building in three cardinal dimensions. For instance for a voxel occupying the (x,y,z) location in the building, we count the voxels that occupy six possible neighbors (x + 1,y,z), (x-1,y,z), (x,y + 1,z), (x,y-1,z), (x,y,z + 1), (x,y,z-1). Finally we investigate the correlations between building form, constraints and fitness by calculating Pearson's correlation coefficient (*r*) for the best individuals of the five forms (Table 12). The results show that c_{neigh} is negatively correlated with CON ($r = -0.96039$ in cell L8) and positively correlated with SA/V ($r = 0.8803$ in cell K8). This can be because well-connected and compact voxel structures can accommodate a higher number of adjacent spaces. As in our example the neighborhood penalty dominates the fitness value, the same correlations apply to the final fitness as well ($r = -0.90929$ in cell L10 and $r = 0.90608$ in cell K10 respectively). As a result, building forms with minimum SA/V and maximum CON earn higher fitness values.

Other high correlations between constraints that Table 12 reveals are more difficult to interpret. For instance, the apparent correlation between size and jaggedness, separation and neighborhood (cells D1, H1 and I1) are less straightforward, because aggregation-based evaluation functions are compensative. This implies that a constraint may be compromised to make up for another without showing an effect on fitness. Despite these difficulties, it is still possible draw several empirical observations. The positive correlation between c_{neigh} and c_{sep} (cell I8) suggests that the success of separation increases as spaces marked for adjacency pull themselves together. The negative correlation between façade and floor constraints (cell G6) suggests that two constraints that try to assign spaces to different building parts cannot both succeed.

Table 12
The Pearson correlation coefficient values between the building form metrics (AS/v and CON), constraints and fitness.

	1	2	3	4	5	6	7	8	9	10	11	12
	c_{size}	c_{dim}	$c_{compact}$	c_{jag}	c_{convex}	c_{facade}	c_{floor}	c_{neigh}	c_{sep}	Fitness	SA/V	CON
A c_{size}												
B c_{dim}	-											
C $c_{compact}$	0.520714	-										
D c_{jag}	-0.80831	-	-0.74961									
E c_{convex}	-	-	-	-								
F c_{facade}	0.318956	-	0.0560344	0.1990357	-							
G c_{floor}	-0.25	-	-0.506367	0.1011544	-	-0.70926						
H c_{neigh}	-0.65144	-	0.206527	0.1880838	-	-0.555106	0.2225804					
I c_{sep}	-0.83629	-	0.0200225	0.5254468	-	-0.2169	-0.117891	0.85051				
J Fitness	-0.489506	-	-0.378297	0.0793768	-	-0.462854	0.2104612	0.9668	0.74724			
K SA/V	-0.556161	-	0.3194574	0.2935297	-	-0.097007	-0.133549	0.8803	-0.81593	0.90608		
L CON	0.5354999	-	-0.378297	0.001662	-	0.5871195	-0.079008	-0.96039	-0.81593	-0.90929	-0.81649	

Highly correlated items are indicated in bold. Highly correlated items are indicated in bold.

There are other situations that Table 12 doesn't reveal, in which initial conditions (building form) conflict with the constraints. In such cases, penalties remain unavoidable from the beginning:

- The satisfaction of c_{fac} is directly proportional to the total vertical area oriented towards that direction. Similarly c_{flo} satisfaction is proportional to the required floor's surface area;
- c_{dim} and $c_{inertia}$ can be satisfied to the extent that the space can fit within the building. Difficult aspect ratios for both spaces and buildings challenge this;
- Low space convexity inevitably means high jaggedness, but not vice versa;
- A space with an odd number of required voxels cannot easily minimize c_{jag} , as it cannot avoid many corners.

In these cases, the users can either tolerate the suboptimal solutions if searching for 'good enough' solutions. On the other extreme, such difficulties can be seen as an indication of the infeasibility of the building form, leading to the abandonment of that form altogether. Alternatively, the weight of the troublesome constraint can be reduced to relax the selection pressure. However, assigning correct constraint weights remain a challenge, as the relative ranking between a high numbers of constraints with no apparent correlations is difficult to find. Moreover, the influence of weights on the eventual fitness is almost untraceable, as this kind of evaluation is compensative.

Despite the aforementioned difficulties, design explorations with constraint weights yield an expanded search space exploiting fuzzy information. While exhaustively characterizing the parametric space dependencies does not seem realistic for early design, the design process can greatly benefit from a systematic exploration of the relationships between constraints. Future work for EASE includes the study of the interrelationships between the initial form, constraints and the layout solutions.

5.2. Population convergence

We investigate the convergence characteristics of EASE to understand the pace with which it reaches the optimal solution. First, we compare the convergence of the best layout alternatives of the five building forms (Fig. 8). The results show the expected asymptotic convergence for each layout. However, layouts with worse fitness values converge slower, as the intervals between improvements in their fitness are much longer. When we study the best, worst and average fitness values of each generation of layout E1 (Fig. 9), we observe that best fitness gradually increases as expected, while the worst fluctuates within a value range. However, average fitness value starts to decrease as the best fitness value ceases to improve, because P_m increases and inserts random variation in the population. Moreover, average fitness increases

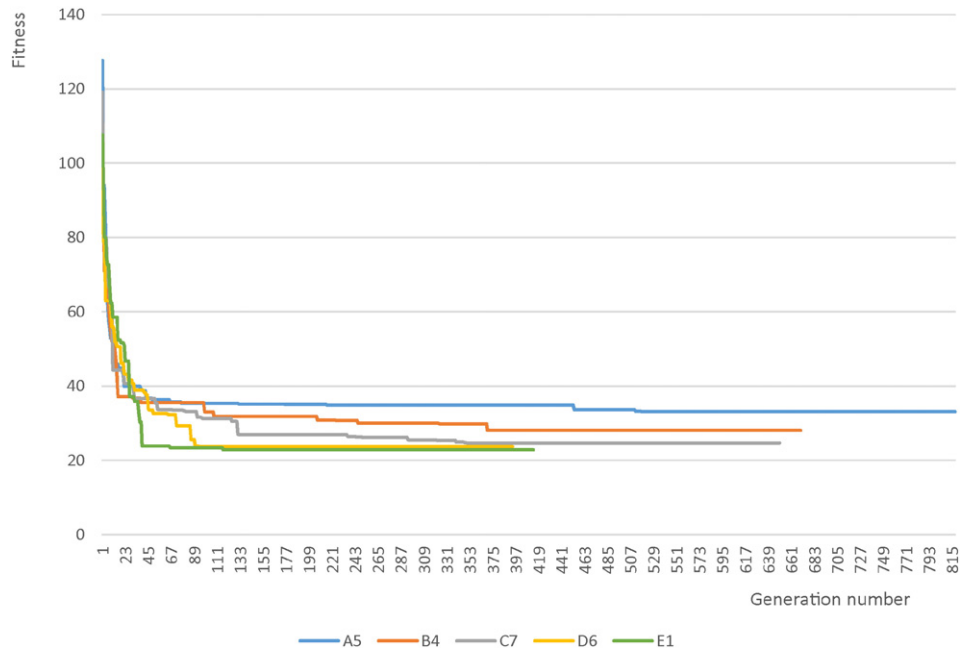


Fig. 8. Convergence graph of the best layouts (A5, B4, C7, D6, E1).

with best fitness, as the population is momentarily dominated by that successful individual's genetic traits. In time, the genetic variance also increases together with P_m .

Finally, we investigate the change in E1's raw constraint penalties over generations (Fig. 10). Here we see that $c_{neigh}, c_{sep}, c_{fac}$ and $c_{inertia}$ start with higher values, and cannot be improved after a degree. To compensate for this difference, weight values need to be selected accordingly low (see the previous section). Moreover, we observe that

sudden fluctuations in multiple constraints occur more frequently in the first generations. Such huge value changes stabilize as a population converges towards an optima.

5.3. Evolutionary operations

As the success of evolutionary algorithms depend largely on their parameterization and parameter settings. We investigate the effect of different EA parameter values (Table 13) on evaluation criteria

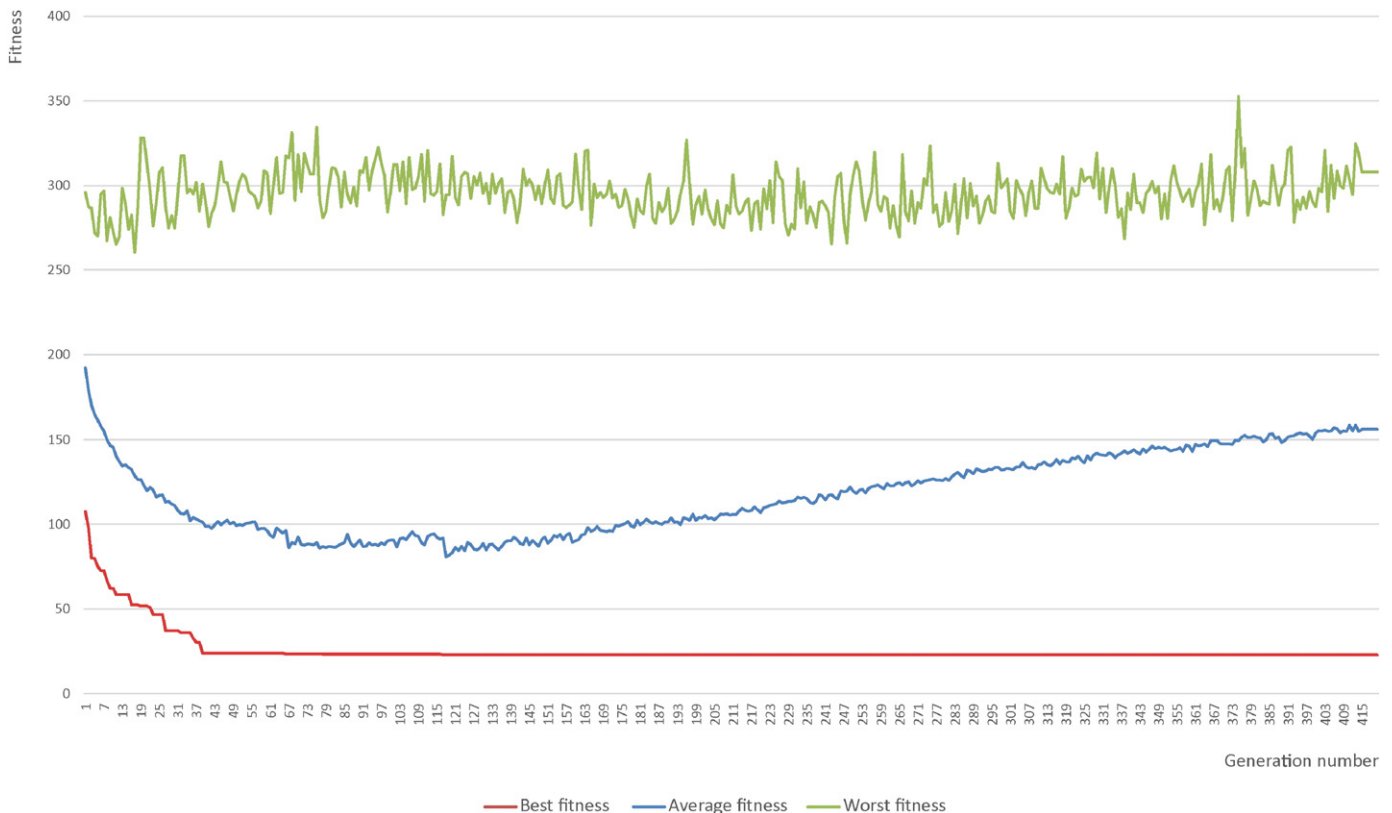


Fig. 9. The convergence of the best, average and worst fitness values for E1.

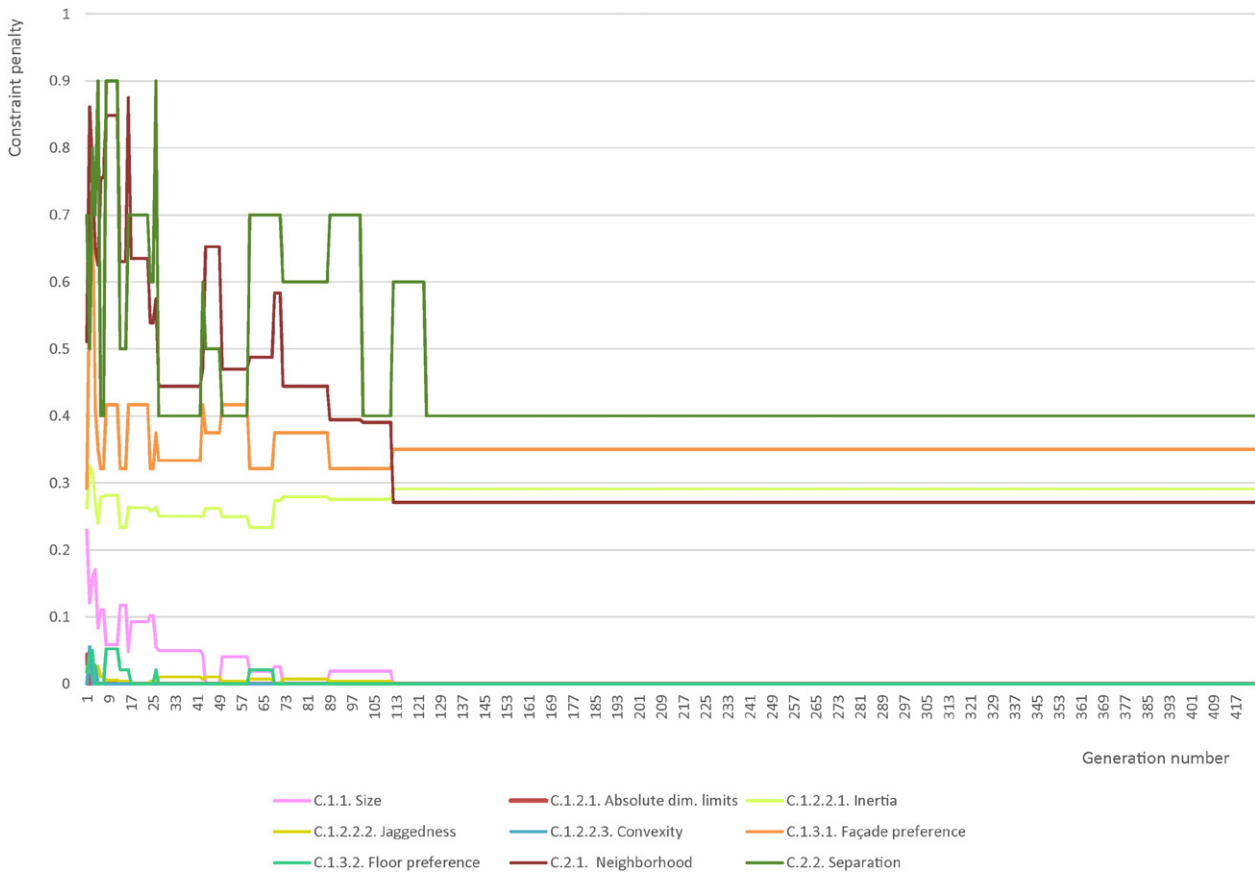


Fig. 10. The raw size constraint penalty values for layout E1.

(Table 14) using form E1, with the same constraint values presented above. We ran EASE ten times for each parameter value and compared the results of the fittest individual and the average fitness value. For all simulations, we used a PC Intel Core i7-4700HQ CPU (2.40 GHz) with a 16.0 GB RAM and 64-bit operating system.

The crossover experiments (Fig. 11A) suggest that all crossover rates can generate acceptable layouts with similar fitness values, but on average higher crossover rates perform better. We had to limit the crossover rate to 90%, as the elitism rate occupies the remaining 10%. Mutation experiments (Fig. 11B) showed greater deviation, such that high mutation

Table 13
EA parameter values used for testing. The bold are the benchmark parameters used in the previous section.

Parameter name	EA Parameter values			
	Parameter 1	Parameter 2	Parameter 3	Parameter 4
P_c Crossover rate	0.50	0.65	0.80	0.90
P_m Mutation rate – to be multiplied by the number of non-improving generations (%)	0.01	0.1	0.25	1
P_r Repair rate – to be multiplied by the number of non-improving generations (%)	0.1	0.025	0.01	0.001
P_e Elitism rate – % fittest of the generation	1	5	10	25
N_p Population size (individuals)	250	500	1000	2500
N_t Termination condition (Number of non-improving generations)	100	300	600	1000

rates became too disruptive, and too low mutation rates fail to converge (Fig. 12). Similarly for repair experiments, repairing an individual help improve fitness, but the stochastic nature of repair operators allow no more than 0.025 (Fig. 11C). For elitism (Fig. 11D), a balance needs to be maintained between oversaturating the population with the elites (resulting in inbreeding), and wasting away the good genetic material.

The generation size experiments (Table 15) suggest that smaller populations ($N_p = 250$ and $N_p = 500$) have an advantage of faster computation time, but perform poorly regarding fitness. Here, insufficient population size leads to stagnation as there remains little genetic variety. Larger populations' ($N_p = 2500$) fitness, on the other hand, perform very similarly to the best case ($N_p = 1000$), require higher computation time, and therefore are not preferred.

The termination condition experiments (Table 16) vary the number of non-improving generations before the algorithm terminates. The objective is to avoid premature termination by allowing sufficient convergence time. Increasing this parameter also increases both the duration of mating and the mutation probability. However, no significant improvement was observed after number of generations ($N_t > 300$). Therefore, we avoid unnecessary computation by selecting $N_t = 300$. Other termination criteria can be further tested

Table 14
Criteria to test EA parameters.

Evaluation criterion	Description
F_b	Best fitness
F_a	Average fitness
C_{as}	Average converge time (seconds)
C_{ag}	Average converge time (no. of unimproved generations)

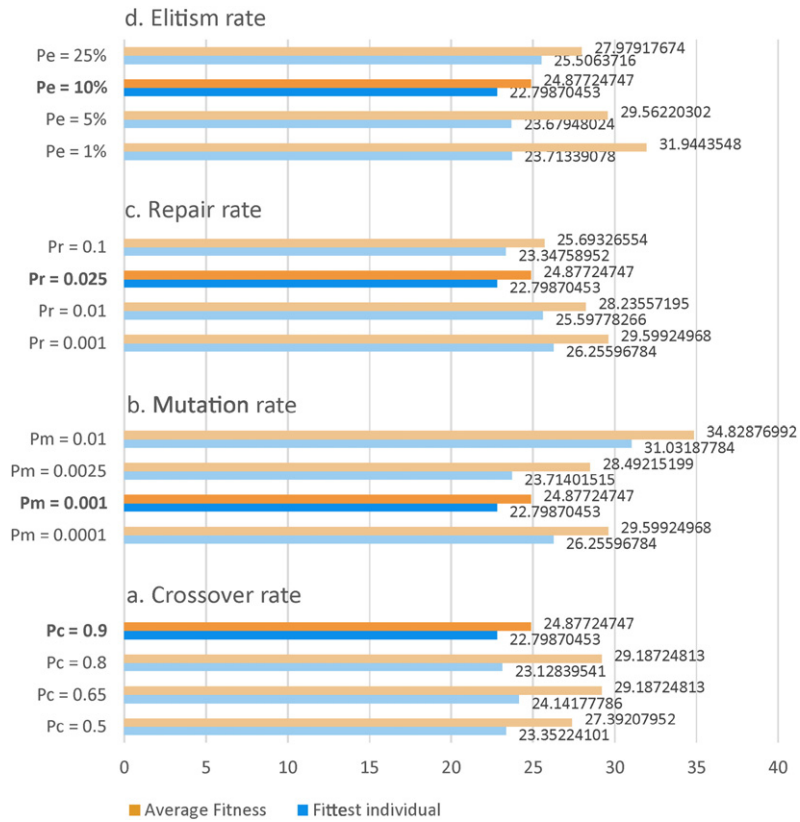


Fig. 11. Best and average fitness values for different values for crossover, elitism, repair and mutation rate parameters. The benchmark values used in the previous section are highlighted.

in the future, such as running mean, standard deviation, Phi, Kappa or best-worst [14].

Having concluded the testing of the EA parameters, it must be noted that varying one parameter at a time provides a limited view of the situation, as these parameters are interdependent and interact with each other non-linearly. Moreover, the large number of parameter configurations and the lack of insight on their effect on fitness can challenge such trial-and-error approaches. In the future, EA parameters can be tuned using methods such as Meta Evolutionary Algorithm, Meta Estimation of Distribution Algorithm or Sequential Parameter Optimization [21].

5.4. Exploration of 2D layouts

We test the applicability of EASE on an L-shaped 2D building. EASE can easily be converted into a 2D layout solver by limiting the initial building representation (A_{init}) to one level. We also use the same spatial program (Table 3), constraint settings (Tables 4 and 5) and EA parameters (Table 8) as in the previous case study. However, the 2-dimensionality of the problem has an effect on several constraints (Table 17). For instance, the upper limit of the number of corners (cor_{max}) is reduced from 20 to 12, considering that 2D geometries contain fewer corners. Similarly, the neighborhood constraint penalty (c_{neigh}) is decreased, as 2D geometries are less probable to be adjacent

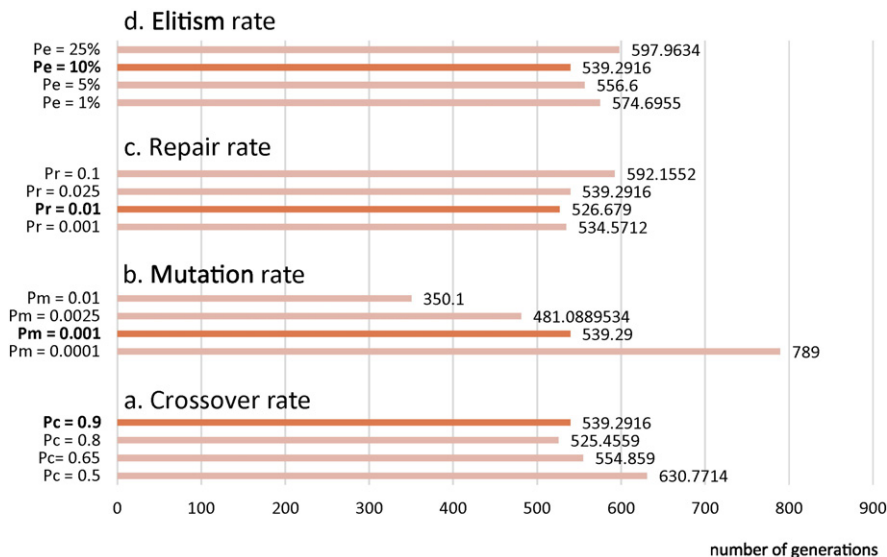


Fig. 12. Average total number of generations for different values for crossover, elitism and mutation rate parameters.

Table 15
Results of generation size testing.

	F_b	F_a	C_{as}	C_{ag}
$N_p = 250$	32.61851	36.6323	72.44	502.75737
$N_p = 500$	23.60606	28.3308	81.7	544.94
$N_p = 1000$	22.79870453	24.87724747	86.7	539.2916205
$N_p = 2500$	24.14178	26.69535	189	563.894

The selected EA parameters are indicated in bold.

Table 16
Results of termination condition testing.

	F_b	F_a	C_{as}	C_{ag}
$N_t = 100$	25.17635281	31.44453351	34.2	217.3
$N_t = 300$	22.79870453	24.87724747	86.7	553.00
$N_t = 600$	23.02984813	27.15810826	142.2	951.1
$N_t = 800$	23.26611186	26.93881892	176.7	1144.1

The selected EA parameters are indicated in bold.

to another geometry. Finally, we set the façade penalty to zero so to deactivate its effect on fitness. We adjust the constraint weights accordingly. In Fig. 13, we present the best 8 layout solutions that had the highest score after 20 runs. The results suggest that EASE can generate meaningful 2D layouts, and more uniform spatial forms as compared

Table 17
The fitness values and raw constraint penalties of the 2D building layouts presented in Fig. 13.

	Weights	2D-1	2D-2	2D-3	2D-4	2D-5	2D-6	2D-7	2D-8
C_{size}	200	0	0	0	0	0	0	0	0
C_{dim}	250	0	0	0	0	0	0	0	0
$C_{compact}$	10	0.186774	0.135939	0.146296	0.151287	0.140585	0.163357	0.119601	0.160613
C_{jag}	750	0	0	0	0	0	0	0	0
C_{convex}	10	0	0	0	0	0	0	0	0
C_{facade}	20	0.416667	0.666667	1	1	0.75	0.375	0.75	0.375
C_{floor}	0	0	0	0	0	0	0	0	0
C_{neigh}	10	0.341667	0.354167	0.508333	0.46875	0.4875	0.508333	0.666667	0.25
C_{sep}	1	0.092593	0.12037	0.148148	0.157407	0.101852	0.055556	0.083333	0.101852
Fitness		5.839965	5.623277	7.435177	7.144815	6.891963	7.050234	8.362681	4.717243

to 3D buildings. However, the satisfaction of C_{neigh} and C_{facade} is harder, due to the above mentioned reasons.

6. Future work and conclusions

In this paper, we present a design tool in support of 3D SLD based on evolutionary algorithms. Considering that EA is an already well-established optimization method, the main modeling effort lies in the development of a layout construction heuristic, the mathematical formulation of the constraints and the designation of the adequate representational structures. Consequently, we propose a novel heuristic method, P-LCH, that generates 3D layouts without overlapping or empty areas, and a Constraint Checker that quantifies spatial constraints.

EASE's effectiveness has been tested on the design process of a library building. While EASE could generate feasible layouts regarding both the fitness values and the physical configurations for all form alternatives, it performs better on compact and well-connected forms as they better satisfy topology and spatial form constraints. Moreover, we found that some constraints may be in conflict with each other and with the initial settings from the beginning. In this case, weights need to be relaxed to reduce the evolutionary pressure. Eventually, we experimented with several EA parameters, and found that a good balance between divergent and convergent search can be found with very high elitism and mutation rates.

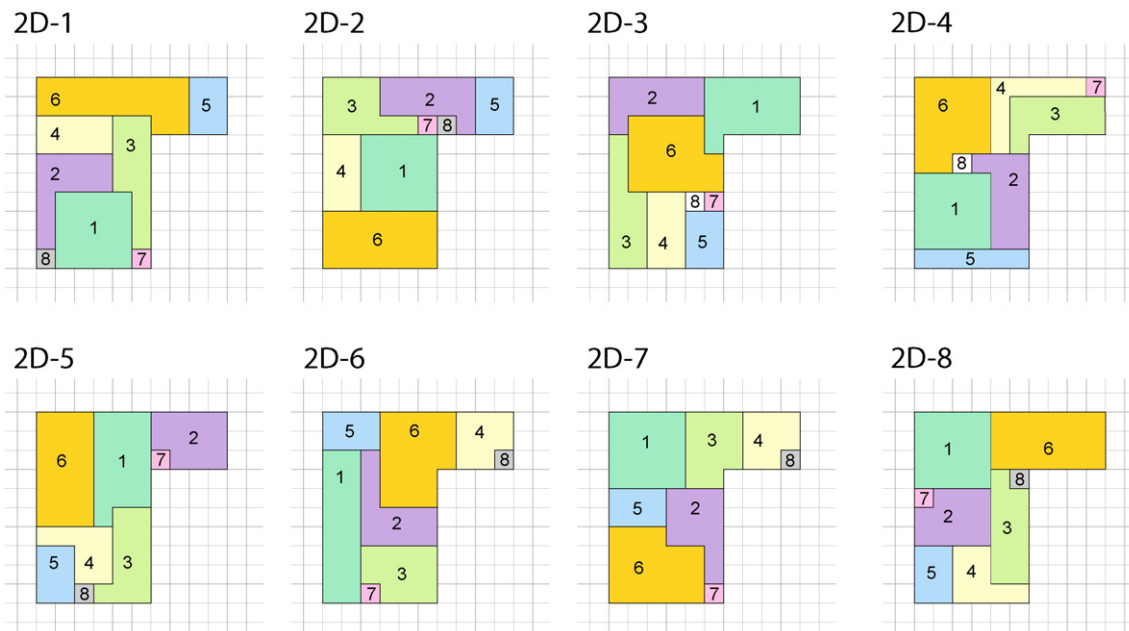


Fig. 13. 2D layout solutions.

The most determinant factor that influences the fitness of the layouts generated by EASE are the constraint weights. Currently, the constraints are weighed manually by the designer in a trial-and-error way. Although such an interaction can provide the designer with valuable insight about the design problem, it nevertheless is a tedious task, especially in case of independent constraints. The constraint weighing process can benefit from more systematic approaches that apply additional optimization procedures that seeks to find the best weights. While this process cannot fully replace designer interaction, it can nevertheless semi-automatically guide the user towards configurations that yield higher fitness. Therefore, our future work include the development of such procedures that can increase the efficiency and usability of the Constraint Checker.

The decision variables and constraints of creative design problems are usually qualitative, and cannot exactly be expressed in metrics without sublimating the intrinsic qualities they carry. Therefore, the design criteria formalized by early design tools by no means can represent the whole complexity of design. This shortcoming inherent in automated design tools can be alleviated through exploratory and iterative interfaces they provide to the designers. The designers' continuous access to the design parameters and working mechanisms of a tool can help her/him play a more decisive role in design. EASE currently provides such interaction at the beginning, when the constraint settings and weights are determined. Future work includes the development of a method that integrates automated optimization with continuous human judgment to allow the designers operationalize it as a more integral component of her/his design process.

Acknowledgments

This work was supported by Middle East Technical University BAP-1 Grant number BAP-08-11-2013-070. The author would like to express her sincere gratitude to Prof. Göktürk Üçoluk for his support, Tarik Kaya for the implementation, and Ekodenge A.S. for their contributions on the case study.

References

- [1] S. Abdinnour-Helm, S.W. Hadley, Tabu search based heuristics for multi-floor facility layout, *Int. J. Prod. Res.* 38 (2) (2000) 365–383.
- [2] R.K. Ahuja, J.B. Orlin, A. Tiwari, A greedy genetic algorithm for the quadratic assignment problem, *Comput. Oper. Res.* 27 (10) (2000) 917–934.
- [3] G. Aiello, M. Enea, G. Galante, A multi-objective approach to facility layout problem by genetic search algorithm and Electre method, *Robot. Comput. Integr. Manuf.* 22 (5) (2006) 447–455.
- [4] G. Aiello, G. La Scalia, M. Enea, A multi objective genetic algorithm for the facility layout problem based upon slicing structure encoding, *Expert Syst. Appl.* 39 (12) (2012) 10352–10358.
- [5] O. Akin, H. Moustapha, Strategic use of representation in architectural massing, *Des. Stud.* 25 (1) (2004) 31–50.
- [6] L. Al-Hakim, On solving facility layout problems using genetic algorithms, *Int. J. Prod. Res.* 38 (11) (2000) 2573–2582.
- [7] A.A. Alhusban, What Does the Architectural Creative Leap Look like through a Conceptual Design Phase in the Undergraduate Architectural Design Studio?(Doctoral Dissertation) Washington State University, Pullman, WA, 2012.
- [8] F. Azadivar, J. Wang, Facility layout optimization using simulation and genetic algorithms, *Int. J. Prod. Res.* 38 (17) (2000) 4369–4383.
- [9] J. Balakrishnan, C.H. Cheng, Genetic search and the dynamic layout problem, *Comput. Oper. Res.* 27 (6) (2000) 587–593.
- [10] A. Banerjee, J.C. Quiroz, S.J. Louis, A Model of Creative Design Using Collaborative Interactive Genetic Algorithms, in: J. Gero, A. Goel (Eds.), *Design Computing and Cognition'08: Proceedings of the Third International Conference on Design Computing and Cognition*, Springer, Atlanta, USA 2008, pp. 397–416.
- [11] R. Baušys, I. Pankrašovaite, Optimization of architectural layout by the improved genetic algorithm, *J. Civ. Eng. Manag.* 11 (1) (2005) 13–21.
- [12] A. Baykasoglu, T. Dereli, I. Sabuncu, An ant colony algorithm for solving budget constrained and unconstrained dynamic facility layout problems, *Omega* 34 (4) (2006) 385–396.
- [13] A. Baykasoglu, N.N. Gindy, A simulated annealing algorithm for dynamic layout problem, *Comput. Oper. Res.* 28 (14) (2001) 1403–1426.
- [14] D. Bhandari, C. Murthy, S.K. Pal, Variance as a stopping criterion for genetic algorithms with elitist model, *Fundamenta Informaticae* 120 (2) (2012) 145–164.
- [15] S.G. Boswell, TESSA—a new greedy heuristic for facilities layout planning, *Int. J. Prod. Res.* 30 (8) (1992) 1957–1968.
- [16] G. Bullock, M. Denham, I.C. Parmee, J. Wade, Developments in the use of the genetic algorithm in engineering design, *Des. Stud.* 16 (4) (1995) 507–524.
- [17] U. Buscher, B. Mayer, T. Ehrig, A Genetic Algorithm for the Unequal Area Facility Layout Problem, in: S. Helber, M. Breitter, D. Rösch, C. Schön, J.-M. Graf von der Schulenburg, P. Sibbertsen, M. Steinbach, S. Weber, A. Wolter (Eds.), *Operations Research Proceedings 2012: Selected Papers of the International Annual Conference of the German Operations Research Society (GOR)*, Leibniz University of Hannover, Germany, September 5–7, 2012, Springer International Publishing, Cham 2014, pp. 109–114.
- [18] W.-C. Chiang, P. Kouvelis, An improved tabu search heuristic for solving facility layout design problems, *Int. J. Prod. Res.* 34 (9) (1996) 2565–2585.
- [19] L. Chwif, M.R.P. Barretto, L.A. Moscato, A solution to the facility layout problem using simulated annealing, *Comput. Ind. Eng.* 36 (1) (1998) 125–132.
- [20] T. Dunker, G. Radons, E. Westkämper, A coevolutionary algorithm for a facility layout problem, *Int. J. Prod. Res.* 41 (15) (2003) 3479–3500.
- [21] A.E. Eiben, S.K. Smit, Parameter tuning for configuring and analyzing evolutionary algorithms, *Swarm Evol. Comput.* 1 (1) (2011) 19–31.
- [22] M.A. El-Baz, A genetic algorithm for facility layout problems of different manufacturing environments, *Comput. Ind. Eng.* 47 (2) (2004) 233–246.
- [23] R.W.J. Flack, B.J. Ross, Evolution of Architectural Floor Plans, in: C. Chio, A. Brabazon, G.A. Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, C. Prins, J. Romero, G. Squillero, E. Tarantino, A.G.B. Tettamanzi, N. Urquhart, A.Ş. Uyar (Eds.), *Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG*, Torino, Italy, April 27–29, 2011, Proceedings, Part II, Springer Berlin Heidelberg, Berlin, Heidelberg 2011, pp. 313–322.
- [24] U. Flemming, C.A. Baykan, R.F. Coyne, M.S. Fox, Hierarchical Generate-and-Test vs Constraint-Directed Search, in: J.S. Gero, F. Sudweeks (Eds.), *Artificial Intelligence in Design '92*, Springer Netherlands, Dordrecht 1992, pp. 817–838.
- [25] T. Honiden, Tree structure modeling and genetic algorithm-based approach to unequal-area facility layout problem, *Ind. Eng. Manag. Syst.* 3 (2) (2004) 123–128.
- [26] D. Iclanzan, D. Dumitrescu, Overcoming Hierarchical Difficulty by Hill-Climbing the Building Block Structure, in: D. Thierens (Ed.), *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO-2007)*, ACM, London, England 2007, pp. 1256–1263.
- [27] M. Inoue, M. Unehara, K. Yamada, M. Hiramoto, H. Takagi, Evaluation of Hybrid Optimization with EMO and IEC for Architectural Floor Planning, *Soft Computing and Intelligent Systems (SCIS)*, 2014 Joint 7th International Conference on and Advanced Intelligent Systems (ISIS), 15th International Symposium on Advanced Intelligent Systems (ISIS) IEEE, Kitakyushu, Japan, 2014 54–61, <http://dx.doi.org/10.1109/SCIS-ISIS.2014.7044784>.
- [28] A. Islier, A genetic algorithm approach for multiple criteria facility layout design, *Int. J. Prod. Res.* 36 (6) (1998) 1549–1569.
- [29] J.H. Jo, J.S. Gero, Space layout planning using an evolutionary approach, *Artif. Intell. Eng.* 12 (3) (1998) 149–162.
- [30] J.C. Jones, *Design Methods*, John Wiley & Sons, 1992.
- [31] J. Kim, Y. Kim, A branch and bound algorithm for locating input and output points of departments on the block layout, *J. Oper. Res. Soc.* 50 (5) (1999) 517–525.
- [32] J.S. Kochhar, B.T. Foster, S.S. Heragu, HOPE: a genetic algorithm for the unequal area facility layout problem, *Comput. Oper. Res.* 25 (7) (1998) 583–594.
- [33] R. Koenig, K. Knecht, Comparing two evolutionary algorithm based methods for layout generation: dense packing versus subdivision, *Art. Intell. Eng. Des. Anal. Manuf.* 28 (03) (2014) 285–299.
- [34] R. Koenig, S. Schneider, Hierarchical structuring of layout problems in an interactive evolutionary layout system, *Art. Intell. Eng. Des. Anal. Manuf.* 26 (02) (2012) 129–142.
- [35] K.-Y. Lee, M.-I. Roh, H.-S. Jeong, An improved genetic algorithm for multi-floor facility layout problems having inner structure walls and passages, *Comput. Oper. Res.* 32 (4) (2005) 879–899.
- [36] H. Li, P.E. Love, Genetic search for solving construction site-level unequal-area facility layout problems, *Autom. Constr.* 9 (2) (2000) 217–226.
- [37] S. Luke, *Essentials of Metaheuristics*, Lulu, second ed., 2013 (available at <http://cs.gmu.edu/~sean/book/metaheuristics/>).
- [38] K. Mak, Y. Wong, F. Chan, A genetic algorithm for facility layout problems, *Comput. Integr. Manuf. Syst.* 11 (1) (1998) 113–127.
- [39] A.R. McKendall, J. Shang, S. Kuppusamy, Simulated annealing heuristics for the dynamic facility layout problem, *Comput. Oper. Res.* 33 (8) (2006) 2431–2444.
- [40] Z. Michalewicz, D.B. Fogel, *How to Solve it: Modern Heuristics*, Springer Science & Business Media, 2013.
- [41] M. Rajasekharan, B.A. Peters, T. Yang, A genetic algorithm for facility layout design in flexible manufacturing systems, *Int. J. Prod. Res.* 36 (1) (1998) 95–110.
- [42] H.W. Rittel, M.M. Webber, Dilemmas in a general theory of planning, *Policy. Sci.* 4 (2) (1973) 155–169.
- [43] E. Rodrigues, A.R. Gaspar, Á. Gomes, An approach to the multi-level space allocation problem in architecture using a hybrid evolutionary technique, *Autom. Constr.* 35 (2013) 482–498.
- [44] M.J. Rosenblatt, The dynamics of plant layout, *Manag. Sci.* 32 (1) (1986) 76–86.
- [45] M. Rosenman, J. Gero, Evolving Designs by Generating Useful Complex Gene Structures, in: P.J. Bentley (Ed.), *Evolutionary Design by Computers*, Morgan Kaufman, San Francisco 1999, pp. 345–364.
- [46] R. Saunders, J.S. Gero, Artificial Creativity: a Synthetic Approach to the Study of Creative Behaviour, in: J.S. Gero, M.L. Maher (Eds.), *Computational and Cognitive Models of Creative Design*, Key Centre of Design Computing and Cognition, Sydney 2001, pp. 113–139.
- [47] E. Shayan, A. Chittilappilly, Genetic algorithm for facilities layout problems based on slicing tree structure, *Int. J. Prod. Res.* 42 (19) (2004) 4055–4067.

- [48] H.A. Simon, Rational choice and the structure of the environment, *Psychol. Rev.* 63 (2) (1956) 129.
- [49] W.M. Spears, K.D. De Jong, On the Virtues of Parameterized Uniform Crossover, in: R. Belew, L. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA 1991, pp. 230–236.
- [50] T. Starkweather, S. McDaniel, K.E. Mathias, L.D. Whitley, C. Whitley, A Comparison of Genetic Sequencing Operators, in: R.K. Belew, L.B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA 1991, pp. 69–76.
- [51] E.-G. Talbi, O. Roux, C. Fonlupt, D. Robillard, Parallel ant colonies for the quadratic assignment problem, *Futur. Gener. Comput. Syst.* 17 (4) (2001) 441–449.
- [52] K. Tam, Solving facility layout problems with geometric constraints using parallel genetic algorithms: experimentation and findings, *Int. J. Prod. Res.* 36 (12) (1998) 3253–3272.
- [53] M.-j. Wang, M.H. Hu, M.-Y. Ku, A solution to the unequal area facilities layout problem by genetic algorithm, *Comput. Ind.* 56 (2) (2005) 207–220.
- [54] Y. Wu, E. Appleton, The optimisation of block layout and aisle structure by a genetic algorithm, *Comput. Ind. Eng.* 41 (4) (2002) 371–387.
- [55] W. Xie, N.V. Sahinidis, A branch-and-bound algorithm for the continuous facility layout problem, *Comput. Chem. Eng.* 32 (4) (2008) 1016–1028.