Utility of Evolutionary Design in Architectural Form Finding: An Investigation into Constraint Handling Strategies

Likai Wang, Patrick Janssen and Guohua Ji

Evolutionary design allows complex design search spaces to be explored, potentially leading to the discovery of novel design alternatives. As generative models have become more complex, constraint handling has been found to be an effective approach to limit the size of the search space. However, constraint handling can significantly affect the overall utility of evolutionary design. This paper investigates the utility of evolutionary design under different constraint handling strategies. The utility is divided into three major factors: search efficiency, program complexity, and design novelty. To analyze these factors systematically, a series of generative models are constructed, and populations of designs are evolved. The utility factors are then analyzed and compared for each of the generative models.

Introduction

In the last decade, the use of evolutionary design (ED) has been gaining popularity in architecture as a strategy for architects to improve building designs. By defining generative models (GM) for the building design and evaluative models (EM) for the building performance, designers are able to use evolutionary algorithms to explore complex design search spaces and discover design alternatives for different objectives [1]. In some cases, novel design alternatives can be discovered that not only break conventional rules of thumb but also lead to innovative solutions that are able to resolve complex design challenges [2, 3].

L. Wang (✉) · G. Ji
Nanjing University, Nanjing, China
e-mail: dg1436002@smail.nju.edu.cn

P. Janssen
National University of Singapore, Singapore, Singapore

J. S. Gero (ed.), *Design Computing and Cognition '18*,
https://doi.org/10.1007/978-3-030-05363-5_10

Among the three major components in ED (the evolutionary algorithm, the GM, and the EM), the GM has the most direct impact on outcomes of ED as well as the overall utility of the ED. This research focuses primarily on GMs for generating building geometries with a specific emphasis on constraint handling.

In architecture, GMs have become an important subdomain within ED research. Various innovative form-finding approaches have been explored by Frazer [4], Bentley and Kumar [5, 6], and others. Following these pioneers, other researchers have explored GMs with wide-ranging diversity [1]. Theoretically, the ED based on such GMs is useful for architects to explore design space and find solutions with excellent performance.

However, when such EDs are applied to real-world architectural designs it is often inapplicable to use since viable solutions are difficult to be found within reasonable time frames or deadlines set by practice. On the one hand, the process of ED is often prolonged by detailed performance simulations, the time cost by which can range from seconds to hours per design solution [7]. On the other hand, as GMs have become more complex to describe a detailed building design, the associated number of parameters and the resulting dimensions of the search space have also increased rapidly, which leads to an exponentially expanding size of design search spaces [8, 9]. Since there is usually a high proportion of invalid solutions in the search spaces [10], the expansion of the search spaces may also result in an increase in the number of invalid design solutions [11]. Therefore, the convergence of the evolutionary process is hindered due to the need to exclude large numbers of invalid design solutions.

Detailed simulations and large search spaces directly result in long running times of the evolutionary process which can severely weaken the utility of ED. Aside from the performance simulations which is out of designers' control, the search space is a critical factor in reducing the running times. Therefore, when constructing the GM for complex building designs, designers can incorporate constraint handling strategies in the GM in order to compress the search space [2, 12]. Such strategies can improve search efficiency by preventing computational resources from being spent on invalid design solutions.

In general, constraint handling can be categorized into two major classes: indirect and direct approaches. The main difference is that the indirect approach embeds constraints in the EM, while the direct approach embeds constraints in the GM [12]. These two approaches have different impacts on the evolutionary process. With indirect constraints handling, invalid solutions are identified and downgraded by the EM, which will lead to them becoming excluded during the evolutionary process. Direct constraint handling, in contrast, uses the GM to filter out invalid design solutions by including explicit or implicit rules [2, 13].

In general, the direct approach is preferred due to its ability to reduce the size of the search space, thereby improving the overall search process. However, with regard to overall ED utility, three main drawbacks have been identified, relating to *search efficiency*, *program complexity*, and *design novelty*.

First, search efficiency may be negatively impacted due to the introduction of disruptive nonlinearities into the genotype-to-phenotype mapping, which will result in a more irregular fitness landscape. The irregular change in fitness will make the evolutionary search process more difficult to extract information to predict promising design subspaces [14].

The second drawback of embedding constraint handling into GMs is that it results in more complex control flows, which makes the program implementation and maintenance more difficult [5]. These characteristics are particularly demanding for architects who are mostly not good at programming.

The third drawback of embedding constraint handling into GMs is the fact that it may reduce design novelty. For architects, design novelty is a critically important factor. GMs must be able to generate designs that vary significantly in terms of their form and configuration.

Direct constraints handling, therefore, is a double-edged sword for ED. The resulting conflict between search efficiency, program complexity, and design novelty is a complex trade-off. However, current understanding of these factors and the trade-off between them in the field of ED is not well understood. Taking this as the point of departure, this study investigates the relationships between constraint handling and the three abovementioned factors. A series alternative GMs based on different constraints are constructed and populations of designs are evolved. The quality of these factors for each GM is then analyzed and compared.

Method: A Framework for Analyzing Utility

The aim of this study is to develop approaches that can help designers to evaluate which GM constraint handling strategies are suitable for design scenarios in terms of the three proposed utility factors. Even though absolute metrics are hard to come by, there are still various relative measures that can be used.

Search Efficiency

Search efficiency refers to the extent to which the GM enables the evolutionary search process to converge on viable design solutions. In practice, the search efficiency is typically one of the most pragmatic factors.

The search efficiency of alternative GMs can be objectively compared by analyzing the evolutionary search process. It is closely related to the size of the search space. Smaller search spaces will typically result in evolutionary processes that are able to find viable solutions in the short term and converge rapidly.

As additional constraints are embedded into the GM, the search space will continuously shrink, and search efficiency may be progressively improved. However, the negative impact of the more irregular fitness landscape also needs to be taken into account. Therefore, the overall effect of constraint handing on search efficiency remains an open research question.

Program Complexity

Program complexity refers to the complexity of the control flow of the GM code. In practice, the coding of complex constraint handling control flows can present significant technical difficulties for architects who are not good at programming.

The rising complexity of a program and the associated degradation of its maintainability cannot be measured by reference to the number of lines of program code. An alternative approach to measuring code complexity is *cyclomatic complexity* [15]. By counting the numbers of nodes and edges in the control flow graph of a program, the cyclomatic complexity measures the number of all linearly independent paths. This index has a close relationship with maintainability of programs. As the cyclomatic complexity increases, the control flow becomes more complex. This will typically result in extra coding effort and time that have to be spent on debugging and refactoring.

Lower program complexity, however, cannot ensure that the overall effort for ED implementation will be reduced. Since simple control flows are usually unable to avoid invalid solutions being generated, the more coding effort may have to be spent on implementing indirect constraint handling strategies, such as penalty functions, in the EM for downgrading undesired design solutions. As the result, coding effort saved by the simple GM control flow will, in many circumstances, be offset by additional coding effort in the EM.

Design Novelty

Design novelty refers to the ability of the GM to generate viable solutions that are unexpected. Discovering novel design alternatives is one of the main aims of ED. Therefore, the significance of design novelty may outweigh the search efficiency and the program complexity when it comes to the overall utility [16].

In most architectural design cases, the potential to discover novel design solutions has a close relationship with the formal variability of the phenotype space. If the phenotype space is overly restricted by constraints, it becomes more difficult for the evolutionary search process to find novel design solutions. Thus, although search efficiency can be improved by constraint handling, the overall utility may still be weakened, or even exhausted if fewer or no novel design solutions can be found.

Design novelty, however, is hard to evaluate objectively. Some measures have been developed [3, 17, 18], but these are themselves somewhat subjective and are hard to implement. In general, the degree of design novelty is highly subjective and largely determined by the needs of architects and projects. However, a visual appraisal of formal variability can provide a rudimentary way of differentiating the amount of design novelty from the architectural perspective. In reverse, GMs with low design novelty usually generate solutions that are visually very similar.

Case Study

In order to systematically investigate the evolutionary design utility factors for architectural designs, a case study high-rise office building design with an atrium and vertical gardens is introduced. The combination of atriums and vertical gardens is widely used as an effective strategy for improving environmental performance in many regions, from tropical to temperate climate zones. Examples include Commerzbank Tower in Frankfurt, Germany, and the Tongji University Multi-Functional Building in Shanghai, China [19].

In recent years, many GMs representing such building designs have been constructed for various design optimization problems. Based on these GMs, ED then can be used to explore possible configurations of these vertical gardens. The configuration of vertical gardens can be categorized as a subdomain of *facility layout problem*, which mainly addresses the various layout problems from the perspective of material handling costs, spatial efficiency, etc. [8, 20, 21]. However, in most cases, the adopted generative rules controlling combinations and allocations of vertical gardens are not properly constrained to avoid invalid solutions being generated. Thus, finding an appropriate trade-off between conflicting performance criteria requires atriums and vertical gardens to be carefully controlled and configured within the building volume [20].

Different constraint handling strategies can be incorporated in GMs to regulate the configurations of vertical gardens. In order to investigate the impact of constraint handling strategies on overall utility, four alternative GMs were implemented and tested. Each GM incorporated incrementally more constraint handling.

A GM with basic constraints was first constructed, referred to as the *naïve* GM (N-GM) and represents an elementary approach for generating the building. To compare the effects of constraint handling on evolutionary designs, three GMs with incrementally more constraints were constructed based on the same structure frame as in the N-GM. These three GMs, respectively, referred to as the *constrained* GM (C-GM), the *constrained-repaired* GM (CR-GM), and the *constrained-confined* GM (CC-GM), literally reflect their constraint handling strategies.

Figure 1 presents the random sampling generated by these four GMs. In general, the formal variability decreases with more constraints embedded. The distinct formal variability will result in significant effects on different utility factors of the ED.

For the case study, a fixed structural frame is used, consisting of a rectangular plan office floor with an open atrium in the center rising up through the whole building, flanked by two structural cores on both sides. The core-column structure is taken as the structural prototype as it has been widely applied in practice for its spatial and constructional efficiency. The size of each column grid is 8.4 by 8.4 m, which is proved can achieve a desirable balance between spatial and structural efficiency [22].

A *modular approach* is applied in the GMs, which partitions the floor plan into multiple fixed-size modules [20, 21]. Thus, each floor is divided into 11 cells based on this structural frame (Fig. 2). Except for the cells representing the structural cores

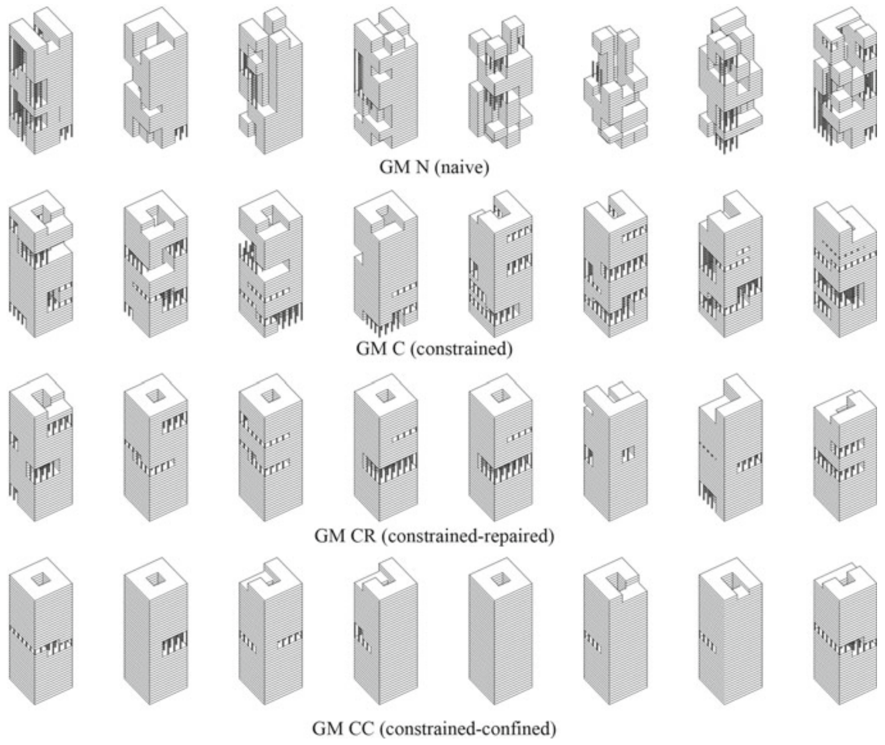

Fig. 1 Random sampling (not evolved) based on the presented GMs

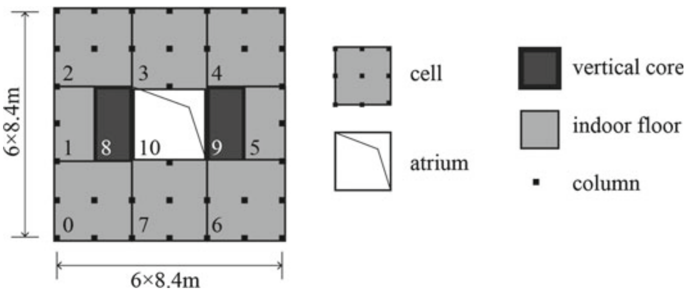

Fig. 2 The structural frame

(#8 and #9 in Fig. 2) which are fixed under all circumstances, all perimeter cells (#0–7 in Fig. 2) can be switched from solid to void, thereby creating complex patterns of interlocking indoor and outdoor spaces. In the presented case study, the tower is assumed to be 40 stories tall.

The evaluation model will first be briefly introduced, followed by the alternative generative models, each with varying levels of constraints.

Evaluation Model

A detailed simulation of environmental performance is beyond the scope of this study, and running relevant simulations would be also too time-consuming. In order to evaluate the generated solutions, a simplified EM based on an economic index is used. This index has the advantage that it is fast to calculate.

For each floor, the fitness function calculates the potential profit that can result from the rentable floor area and subtracts three construction cost factors: the core cost, the slab cost, and the facade cost.

- Potential profit: Rentable floor area multiplied by a factor that gives preference to south facing spaces and spaces on the upper or lower floors (due to the better view or accessibility).
- Core cost: The area of the structural cores in plan multiplied by a factor that increases with the rise of the floor level (due to the difficulty of construction on high).
- Slab cost: Slab area (excluding core but including outside spaces) multiplied by a factor that increases with the rise of the floor level (due to the same reason as the core cost).
- Facade cost: Facade area multiplied by a constant cost factor (due to façade cost mostly related to the material).

In reality, the gross area of buildings is regulated by urban planning codes. As the result, the EM also defines an upper limit of the gross area for the whole building. A solution whose gross indoor floor area surpasses a predefined limit (70,000 m^2 in this EM) will have its potential profit proportionally scaled back according to the excess area.

Based on the above EM, every design solution will have a distinct fitness. An analysis of randomly generated designs confirmed that the solutions that intuitively seem to be desirable also received high fitness values.

Unconstrained GM

As a comparative baseline for the four presented GMs, a GM called *unconstrained* GM (U-GM) is first introduced. This GM is not actually implemented and also not evolved for testing. No constraints are implemented in this GM, so the number of constraints is 0. The constraint-free mapping allows all possible solid-void combinations to be generated. The number of combinations (search space) is 8^40 ≈ 1.329e+36. This search space is actually impossible to be searched through under current computational capacity. This GM represents the simplest way of generating the target design solutions; therefore, it can be used as the baseline to reveal the impact of different constraint handling strategies has on compressing the design space.

Naive GM

As to N-GM, floors are grouped into ranges with 2-to-5 consecutive floors, and each group has the same layout of the vertical garden (solid-void patterns). Applying floor groups is not only for reducing the number of parameters but also for the reason that single-floor vertical gardens are uneconomic and too dark.

The tower is divided into 10 groups. As the ten groups will each have variable floors, it may result in either too many or too few floors. Some simple rules are therefore applied in order to ensure that the correct number of floors is achieved. If the total floors are less than 40, then the topmost floor layout will be taken to fill the rest floors. If the total floors are greater than 40, then extra floors will be culled.

In this GM, the solid-void condition of every cell is defined by a binary switch. This results in a genotype–phenotype mapping that is straightforward (without conditional statements, iteration, or subroutines). This simple control flow is easy to implement and often applied to these types of optimization problems.

The genotype defines the layout for ten floor groups. For each group, the genotype contains two parameters. The first parameter is an integer between 2 and 5, defining the number of floors in that group ($p1$ in Fig. 3). The second parameter is a string containing 8 binary switches, defining the solid-void pattern for the eight perimeter cells in that floor group ($p2$ in Fig. 3). As a result, the design space of this GM is $(4 \times 2\hat{}8)\ \hat{}10 \approx 1.268e{+}30$. The constraint handling strategies of floor groups and numbers of floors within one group impart N-GM with two constraints compared with the U-GM.

At the same time, the simple mapping process results in N-GM having the most regular fitness landscape compared with the other GMs (aside from the U-GM). The independent binary combinations allow a wide range of possible design solutions to be generated. However, this unrestricted diversity also allows many invalid design solutions to be generated as stochastic combinations of vertical gardens can result in unbuildable designs. For example, solutions may have very large or disproportionate voids or many separated small voids on the facades, or, in some other cases, suspended or large overhanging cells (see the first line in Fig. 1). Such problematic features will result in the expensive construction cost or make it hard to rent due to poor spatial accessibility or connectivity [20].

$$[\ldots\ \underbrace{3}_{\mathrm{p1}},\ \ \underbrace{11101001}_{\mathrm{p2}},\ \ \underbrace{2}_{\mathrm{p1}},\ \ \underbrace{01011001}_{\mathrm{p2}},\ \ \underbrace{4}_{\mathrm{p1}},\ \ \underbrace{10011110}_{\mathrm{p2}}\ \ldots]$$

Fig. 3 Example of genotype data structure of N-GM

Constrained GM

The C-GM limits the number and size of vertical gardens. First, the number of vertical gardens is limited to one per floor, as multiple small vertical gardens result in a huge façade area which is costly in building materials. Second, the size of a vertical garden should be controlled and should not be significantly larger than that of the indoor space for the rental profitability. In addition, vertical gardens should be connected to the atrium to facilitate natural ventilation [19].

For C-GM, the genotype still defines the layout for ten floor groups. However, in order to constrain the GM to the above rules, certain modifications were introduced into the mapping process. For each floor group, the genotype now contains three parameters on different decision levels. The first parameter is the same as the N-GM and defined the number of floors in that group ($p1$ in Fig. 4).

The second and third parameters ($p2, p3$ in Fig. 4) replace the binary string ($p2$ in Fig. 3). These parameters are used to create voids through a mapping process with conditional statements. Since there are only two cells directly connecting the atrium, the vertical garden must include one of these two cells. Thus, the second parameter ($p2$ in Fig. 4) is either 0, 1, or 2. If the value is 0, then it indicates that there will be no void, in which case the third parameter can be ignored. If the value is 1 or 2, then it indicates which one of the two cells adjacent to the atrium will be included in the vertical garden. Finally, the third parameter ($p3$ in Fig. 4) is an integer that assigns a solid-void pattern from a predefined set for the vertical garden. To restrict the size of the vertical garden, the number of cells in each void pattern is limited to a maximum of 5, which results in a total of 13 unique patterns (Fig. 5). As the result, the search space of this GM is $(4 \times 3 \times 13)\,\hat{}\,10 \approx 8.536\text{e}{+}21$. The two extra constraints on floor layouts make the C-GM with two more constraints compared with that of N-GM (totally four constraints).

By excluding most stochastic combinations of small voids in the building volume, the rationality of the generated design solution of C-GM is improved considerably (see the second line in Fig. 1). At the same time, there is a significant compression of the design search space compared with that of N-GM.

However, the explicit constraint rules also result in a more irregular fitness landscape, due to the introduction of conditional statements into the mapping. These statements result in discontinuities and *neutral mutations* in the genotype–phenotype–fitness mappings. Neutral mutation refers to genotypic mutations that have no effect on the phenotype and the fitness. (For example, in Fig. 5, $p2$ is a higher order decision level that will have a more significant impact on the design fitness. $p3$ becomes neutral when $p2$ defines that no vertical gardens are generated.) Such neutral mutations introduce many-to-one mappings in the GM, resulting in numerous

Fig. 4 Example of genotype data structure of C-GM

$$[\ldots\underbrace{3,}_{\text{p1}}\ \underbrace{2,}_{\text{p2}}\ \underbrace{0,}_{\text{p3}}\quad \underbrace{2,}_{\text{p1}}\ \underbrace{2,}_{\text{p2}}\ \underbrace{12,}_{\text{p3}}\quad \underbrace{4,}_{\text{p1}}\ \underbrace{1,}_{\text{p2}}\ \underbrace{7,}_{\text{p3}}\ldots]$$

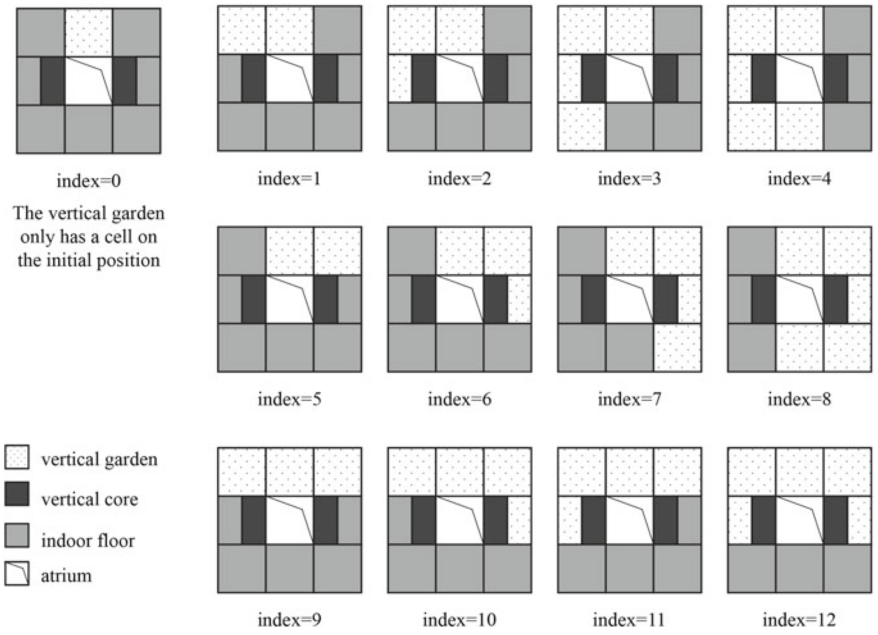

Fig. 5 Floor layout patterns

Table 1 The frequency of neutral mutation of the presented GMs

GM	N-GM	C-GM	CR-GM	CC-GM
Rental profit	7	30	35	20
Constructional cost	6	30	34	22
Gross profit	6	27	32	14

fitness plateaus. Such plateaus can trap the evolutionary process into subspaces with local optimals, thereby resulting in premature convergence [14, 23].

To analyze the frequency of neutral mutations in the presented GMs, 100 pairs of randomly sampled solutions from separated neighboring genotype subspaces were selected and evaluated, and pairs sharing the same fitness values were then counted (Table 1). As shown in Table 1, the additional constraints of the conditional statement result in a significant rise in the frequency of neutral mutations of C-GM compared with that of N-GM.

Although the configuration of the vertical gardens with C-GM has become more rational, the independence between floors layouts can still create certain types of voids that may be problematic. Two key types of problematic voids are identified: oversized voids in cases where two voids meet above one another and become merged, or cross-diagonal voids in cases where two voids meet at a point on the diagonal. Such voids are hard to avoid within the mapping process of C-GM.

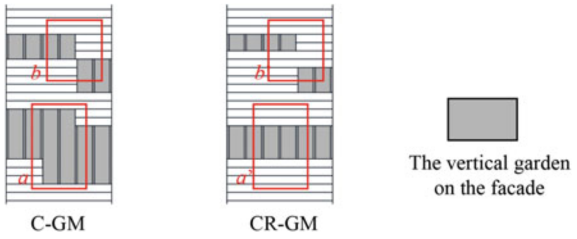

Fig. 6 Example of the repair operators

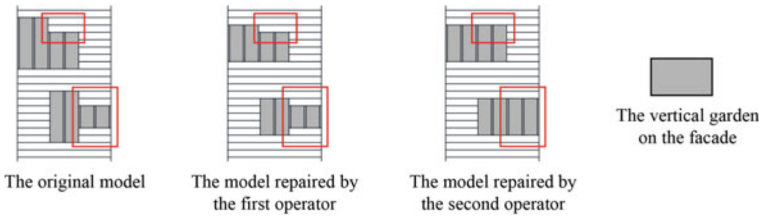
The original model
The model repaired by the first operator
The model repaired by the second operator
The vertical garden on the facade

Fig. 7 Example of the first and the second repair operators

Constrained-Repaired GM

The CR-GM uses the same control flow as the C-GM. However, in order to correct the oversized voids and cross-diagonal voids generated by the C-GM, implicit repair operators are added. The repair operators will fix the oversized voids and cross-diagonal voids by switching cells on selected floors to become non-void.

In the case of the oversized void (larger than five floors), floors are iteratively removed from the top and the bottom of the void, until a suitable height is reached (*a*-*a*′ in Fig. 6). In the case of the cross-diagonal void, all cells on the floor in the middle will be assigned non-void, so that the two voids become disconnected (*b*-*b*′ in Fig. 6).

These repair operators may, however, result in additional problematic conditions being generated. In particular, inserting non-void floors in certain groups can result in many single-floor pendulous cells which are hard to rent or construct. Hence, an extra repair operator is defined in order to correct these conditions. This repair operator will identify isolated or pendulous cells and will switch them to the opposite solid-void condition (Fig. 7). Due to the fact that additional problematic conditions can continuously emerge after the execution of the first and the second repair operators, these operators are run in a loop until all infeasible conditions have been eliminated or the number of iterations reaches a predefined limit. The number of 30 is set as the limit in this GM.

These repair operators are able to filter out most invalid design solutions from the C-GM by further restricting the variability of vertical garden combinations (see the third line in Fig. 1). Including the first and the second repair operators, three more constraints are implemented in the CR-GM compared with that of the C-GM (totally seven constraints). However, as the implicit rule has no effect on the data structure of the genotype, the genotype space remains intact.

With more constraints being embedded, the fitness landscape of CR-GM is further degraded, as the repair operators lead to additional neutral mutations in the phenotype space (Table 1), which makes the fitness landscape more irregular [24]. However, the number of possible combinations can be reduced remarkably by these neutral mutations. Lastly, a significant additional coding effort was required for implementing the more sophisticated mapping process.

Constrained-Confined GM

The CC-GM also uses the same control flow as the C-GM, but compared to the CR-GM, the search space is further compressed by only keeping parameters that have a mostly positive impact on overall fitness. (See the CC-GM solutions in the fourth line in Fig. 1.)

For CC-GM, vertical gardens are only allowed to be inserted in middle to upper floors, and there is a terrace on the roof. To ensure these features can be fully represented, CC-GM only has three floor groups (as opposed to the C-GM, which has ten floor groups).

The first two groups are assigned to floors ranging from 15 to 30 stories, and the third one defines the terrace on the roof. Therefore, the regularity of the fitness landscape of CC-GM is similar to that of C-GM, but the size of the genotype space is much smaller, which is $(4 \times 3 \times 13)$ ^3 $\approx$ 3.796e+6. Compared with C-GM, seven more constraints are defined to disable the change of the remaining seven floor groups (totally 11 constraints).

Evolutionary Run

In order to further investigate the impacts of different constraint handling strategies on search efficiency, program complexity, design novelty, and overall utility, the evolutionary search processes based on the four presented GMs were run.

The evolutionary algorithm was executed using the Rhino–Grasshopper environment, and the standard genetic algorithm in the Galapagos was applied [25]. The population size was set to 100. Due to the large genotype space for some of the presented GMs, the population of the initial generation was raised to 1000. Meanwhile, to avoid the premature convergence, a higher mutation rate and a lower selection pressure were used. (In Galapagos, the settings are 25% for *maintain* and 25% for *inbreeding*.) At the same time, the number of 25 consecutive generations without new improvement solutions is set as the terminated threshold for the evolutionary process. Last but not the least, in order to reduce the impact of stochastic variation, the evolutionary process based on each GM was repeated five times.

Results

For the presented case study, different constraint handling strategies impart each GM with distinct constraint numbers and genotype search space. Table 2 summarizes the number of constraints and the size of the design search space for all five GMs. In general, the size of search space decreases along with more constraints being embedded. Compared with the size of the search space of U-GM, the effect of the constraint handling strategies of C-GM on compressing the search space is more significant than that of any other GM.

Figure 8 shows the fitness progression trend lines of the evolutionary processes. For each GM, five trend lines are shown. The graphs show the best two solutions over time. The reason for recording the best two is that focusing only on the best solution can conceal the overall progress of the whole population. By recording the best two solutions, the improvement of the population can be revealed more subtly and precisely.

The tendency of the trend lines corresponds to the regularity of the fitness landscape. The smoother the landscape (such as N-GM), the more gently and smoothly the trend line grows, which visually reveals the correlation between the constraint handling and the utility of the evolutionary process. Except for N-GM, the fitness landscapes of the other three GMs are irregular to different extents. As a result, the trend lines also become correspondingly more irregular.

From the graph, it can be found that smoothness of the fitness progression trend lines has a strong correlation with the frequency of neutral mutations. The result in Table 1 shows that the frequency of the neutral mutations of both the C-GM and CR-GM are very high. Around or over 30% of the samples share the same fitness values, followed by CC-GM with about 20%, and N-GM with about 7%. As neutral mutations become more frequent, the trend lines grow more irregularly.

Aside from the neutral mutations, the repair operators also have significant impacts on the evolutionary process. Despite the fact that the search space of CR-GM is much bigger than that of CC-GM, the sophisticated repair operators of CR-GM not only facilitate the evolutionary process to converge faster but also result in the discovery

Table 2 The number of constraints and the size of design search space

GM	U-GM	N-GM	C-GM	CR-GM	CC-GM
The number of constraint	0	2	4	7	11
Design search space	1.33e+36	1.27e+30	2.06e+14	2.06e+14	3.796e+6

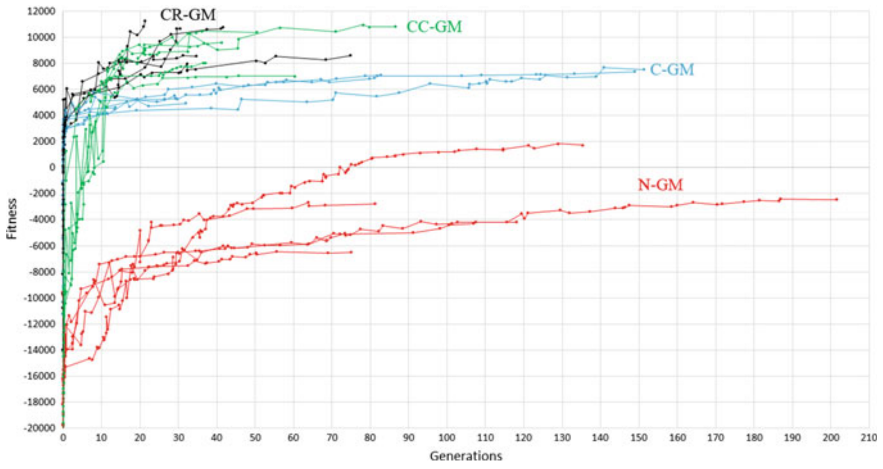

Fig. 8 Fitness progression trend lines for the alternative GMs. The number of generations is plotted along the x-axis, and fitness values on the y-axis

of better solutions. As the result, it can be assumed that the actual size of phenotype space s of CR-GM, which is compressed by the repair operator, is similar to that of CC-GM.

Search Efficiency

As demonstrated in Fig. 8, the search efficiency of the four presented GMs varies significantly. In general, adding more constraints both reduces the number of generations required to find viable solutions and improves the quality of the solutions that are found. The conclusion that can, therefore, be drawn is that, for this case study, the positive effects of a smaller search space outweigh the negative effects of an irregular fitness landscape.

Program Complexity

Constraint handling did not result in significant expansion in the programs' physical sizes of the GMs in this study (500–900 lines). However, the actual increase in the coding effort and time spent on the more complex control flows were considerable. By analyzing the cyclomatic complexity (M) based on the below formula [15], the latent effects brought by the complex control flows are revealed more precisely.

$$M = E - N + 2$$

Table 3 The cyclomatic complexity of the presented GMs

GM	N-GM	C and CC-GM	CR-GM
Nodes	2	6	11
Edges	2	8	16
Complexity	2	4	7

where E is the number of edges in the control flow, and N is the number of nodes.

As shown in Table 3, the complexity of the four GMs roughly increases exponentially, and the numerical differences of the values generally match the actual differences between the amount of coding effort and time spent. As the control flow becomes more complex, much more effort on debugging and refactoring has to be spent to maintain the program. However, the complex control flow and the associated effort can be offset or even outnumbered by the time saved in the evolutionary process.

Design Novelty

Figure 9 lists the results from the evolutionary processes. Similar to Fig. 1, the solutions become more rational as more constraints are embedded in the GM. However, improper use of constraint handling can make the evolutionary results suffer from poor design novelty. The design solutions generated by CR-GM and CC-GM are mostly predictable and lack *design surprise* which means that "the design is unexpected for the domain given previous experience [9, 10]."

In contrast, the solutions generated by N-GM have greater formal diversity but still cannot be regarded as having the desirable design novelty since they cannot be seen as being feasible architectural solutions. This is reflected in their low fitness values, which suggest that these solutions are not economical.

The results from C-GM suggest that there is a possible balance between the need for design novelty and design fitness. Although the designs are topologically similar to that form CC-GM and CR-GM (with similar locations and numbers of the vertical gardens), the less stringent constraint handling allows more unexpected solutions to be discovered. Furthermore, the regularity of the fitness landscape also facilitates the evolutionary process to search the design space more thoroughly, allowing a greater number of alternative design solutions to be evaluated. As the result, the C-GM solutions have more distinct formal features than the CR-GM and CC-GM solutions.

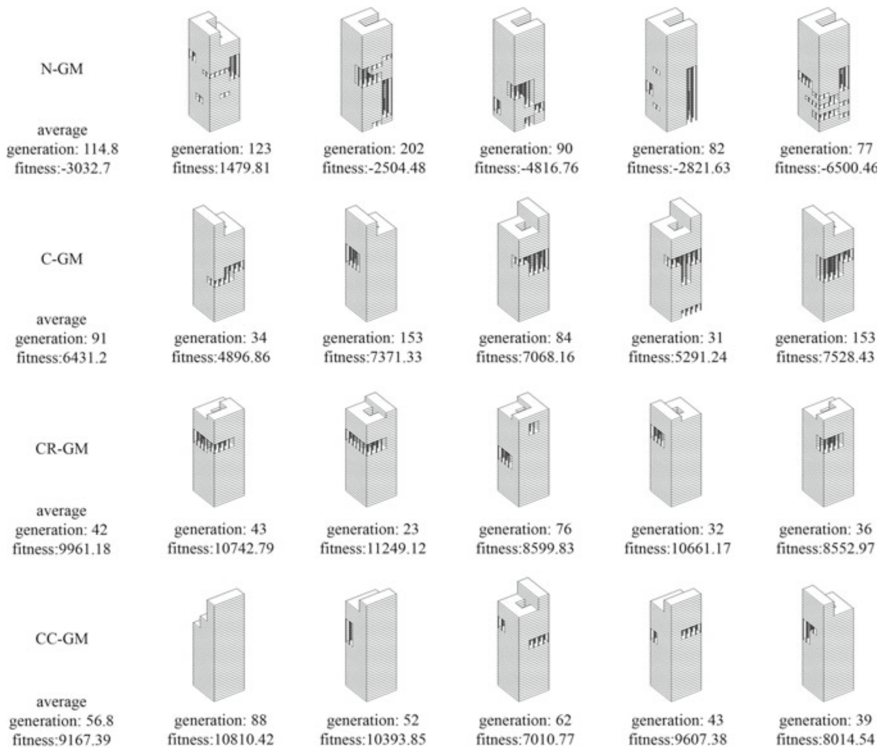

Fig. 9 Evolved design solutions based on the presented GMs

Table 4 Overall qualitative description of the presented GMs

GM	Constraint handling	Search efficiency	Design novelty	Program complexity	Utility
N-GM	Loose	Low	Very high	Low	Low
C-GM	Medium	Fair	High	Fair	High
CR-GM	Tight	Very high	Fair	High	High
CC-GM	Very tight	High	Low	Fair	Low

Utility

By summarizing the utility factors of the four presented GMs, a qualitative conclusion is drawn, as shown in Table 4. Due to the extremely poor search efficiency or low design novelty, it is fair to consider that N-GM and CC-GM are least useful for real-world scenarios.

The utility of the other two GMs, in contrast, is recognized as much better, but it is also affected by external conditions. For CR-GM, the ability to quickly discover viable solutions minimizes the number of evaluations that are required. This allows it to be used in ED systems incorporating computational expensive simulations. However, the limited design variability may make this GM only effective for well-defined design problems.

On the contrary, if the simulation is relatively inexpensive or the design problem is ill-defined, C-GM is likely to be the better choice. The relatively regular fitness landscape of the C-GM facilitates the evolutionary process to search the design space more completely, and the potential to discover novel design alternatives is also higher.

Conclusion

In this study, the utility of constraint handling in GMs has been researched. For the case study investigated in this research, utility factors vary significantly when different constraint handling strategies are applied in the GM. On the one hand, constraint handling has a positive impact on both the search efficiency and design fitness. However, on the other hand, overly stringent constraint handlings can significantly weaken the other utility factors, especially design novelty.

Exclusively focusing on search efficiency by embedding evermore constraints in the GM is unlikely to be an effective strategy, as it will result in low design novelty and complex control flows which are hard to implement and maintain. As the result, a more balanced approach to constraint handling is critical to achieving effective and efficient evolutionary processes. For architects, in order to ensure that the resulting ED system is applicable for the purpose, the different utility factors should be carefully considered before constructing a GM.

Last but not the least, the impacts of constraint handling on the utility of ED may vary considerably across different GMs, and it is therefore not possible to draw generalized conclusions until more research under different design scenarios has been conducted. However, the importance of the overall utility is clearly revealed in this study, and further research will facilitate architects to carry out ED more efficiently and effectively in the future.

Acknowledgements This paper was supported by the National Natural Science Foundation of China (51378248) and the China Scholarship Council (201706190203).

References

1. Caldas L (2008) Generation of energy-efficient architecture solutions applying GENE_ARCH: an evolution-based generative design system. Adv Eng Inform 22:59–70
2. Eiben AE, Smith JE (2004) Introduction to evolutionary computing. New York
3. Gero JS (2006) Computational models of creative designing based on situated cognition. In: Hewett T, Kavanagh T (eds) Creativity and cognition 2002. ACM Press, New York, NY, pp 3–10
4. Frazer J (1995) An evolutionary architecture. Architectural Association, London

5. Bentley P, Kumar S (2003) Three ways to grow designs: a comparison of embryogenies for an evolutionary design problem. In: Proceedings of the 1st annual conference on genetic and evolutionary computation, vol 1, pp 35–43
6. Kumar S, Bentley P (2003) Computational embryology: past, present and future. Adv Evol Comput:1–16
7. Zhou L, Haghighat F (2009) Optimization of ventilation systems in office environment part II: results and discussions. Build Environ 44:657–665
8. Jo JH, Gero JS (1998) Space layout planning using an evolutionary approach. Artif Intell Eng 12:149–162
9. Chen S, Montgomery J, Bolufé-Röhler A (2015) Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution. Appl Intell 42:514–526
10. Rasheed KM (1998) GADO: a genetic algorithm for continuous design optimization, Ph.D. dissertation, Rutgers University, New Jersey
11. Rasheed K, Ni X, Vattam S (2005) Comparison of methods for developing dynamic reduced models for design optimization. Soft Comput 9:29–37
12. Banzhaf W (1994) Genotype-phenotype-mapping and neutral variation—a case study in genetic programming. Parallel Probl Solving Nat III 866:322–332
13. Janssen P, Kaushik V (2014) Evolving Lego, rethinking comprehensive design: speculative counterculture. In: Proceedings of the 19th international conference on computer-aided architectural design research in Asia:523–532
14. Rothlauf F (2006) Representations for genetic and evolutionary algorithms. Springer, Berlin, Heidelberg
15. McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng SE- 2:308–320
16. Rebhuhn C, Gilchrist B, Oman S, Tumer I, Stone R, Tumer K (2015) A multiagent approach to identifying innovative component selection. In Des Comput Cogn 14:227–244
17. Brown DC (2015) Computational design creativity evaluation. Des Comput Cogn 14:207–224
18. Grace K, Maher ML, Fisher D, Brady K (2015) Modeling expectation for evaluating surprise in design creativity. Des Comput Cogn 14:189–206
19. Wood A, Salib R (2013) Guide to natural ventilation in high rise office buildings. Routledge
20. Liggett RS (2000) Automated facilities layout: past present and future. Autom Constr 9:197–215
21. Dino IG (2016) An evolutionary approach for 3D architectural space layout design exploration. Autom Constr 69:131–150
22. Ayşin SEV, Özgen A (2009) Space efficiency in high-rise office buildings. Metu Jfa:2
23. Vanneschi L, Clergue M, Collard P, Tomassini M, Vérel S (2004) Fitness clouds and problem hardness in genetic programming. Genet Evol Comput GECCO2004 Part II 3103:690–701
24. Rothlauf F, Goldberg DE (2003) Redundant representations in evolutionary computation. Evol Comput 11:381–415
25. Rutten D (2013) Galapagos: on the logic and limitations of generic solvers. Archit Des 83:132–135