Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2012), *26*, 129–142.
© Cambridge University Press 2012 0890-0604/12 $25.00
doi:10.1017/S0890060412000030

Hierarchical structuring of layout problems in an interactive evolutionary layout system

REINHARD KOENIG AND SVEN SCHNEIDER

Department of Computer Science in Architecture, Bauhaus-University Weimar, Weimar, Germany

(RECEIVED March 3, 2011; ACCEPTED October 15, 2011)

Abstract

This paper focuses on computer-based generative methods for layout problems in architecture and urban planning with special regard for the hierarchical structuring of layout elements. The generation of layouts takes place using evolutionary algorithms, which are used to optimize the arrangement of elements in terms of overlapping within a given boundary and the topological relations between them. In this paper, the approach is extended by a data structure that facilitates the hierarchical organization of layout elements making it possible to structure and organize larger layout problems into subsets that can be solved in parallel. An important aspect for the applicability of such a system in the design process is an appropriate means of user interaction. This depends largely on the calculation speed of the system and the variety of viable solutions. These properties are evaluated for hierarchical as well as for nonhierarchical structured layout problems.

Keywords: Automated Layout Design; Evolutionary Algorithms; Hierarchical Structuring

1. INTRODUCTION

Layout is the arrangement of elements, usually within a given area for a particular purpose. In the context of architecture and urban design it relates to the arrangement of spatial elements (e.g., parcels, buildings, rooms) on certain scales. Scales denote the different contexts of a design (e.g., urban neighborhood, building volumes, functional areas, floor plans). Jo and Gero (1998) describe layout as "the assignment of discrete space elements to their corresponding locations while the space elements have relationships among each other."

The design of layouts is of central importance in architecture and urban design. The arrangement, shape, and proportions of elements have a decisive impact on the quality and sustainability of buildings, neighborhoods, or entire cities. Depending on the project and the context a range of different requirements needs to be met. To meet these requirements, the conventional design process involves making assumptions at the beginning of the process, which are then constantly reviewed and refined. A prerequisite for this is a significant reduction in the complexity of a design problem with the aid of guiding images or principles (Lawson, 2006). These guiding principles, which are set at the highest level of scale or abstraction, then define the framework for design decisions at lower levels of scale. Such procedures are known as top-down strategies, whereby the functional requirements are usually subordinated to the guiding image. The aim of the efforts in the development of computer-based methods for solving layout problems is to assist the designer in complex tasks using bottom-up strategies that make it possible to take into account complex functional criteria on different scales more or less equally.

The creative development of a design task usually takes place independently of the optimization of functional considerations. Conversely, the automated generation of designs using computer-based methods is undertaken without taking account of subjective ideas or creative intentions. A central area of potential for improving computer-aided design processes can be seen in the consideration of both creative design and optimization methods. In this paper we will first clarify which generative methods are suitable for such an approach (Section 2) and the need for interaction in such systems (Section 3). We then present a mechanism for solving layout problems (Section 4) and apply it to hierarchical structured and nonhierarchical structured layout problems. Finally, we review the special characteristics, advantages and disadvantages of both types (Section 5).

Reprint requests to: Reinhard Koenig, Department of Computer Science in Architecture, Bauhaus-University Weimar, Belvederer Allee 1, Weimar 99423, Germany. E-mail: reinhard.koenig@uni-weimar.de

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

2. METHODS FOR THE COMPUTER-BASED GENERATION OF LAYOUTS

The solution of layout problems using computer-based methods is a central issue in the application of artificial intelligence in the field of architecture. Layout problems are usually very complex problems, which need to meet a variety of requirements. The number of solutions increases exponentially with each factor to be considered, for example, the number of rooms (March & Steadman, 1971). From the perspective of complexity theory, layout problems fall into the category of so-called nondeterministic polynomial time complete problems.

Since the early 1960s different methods have been developed for the computer-based solution of such problems (Whitehead & Eldars, 1964). Common to all of these methods is a generative mechanism for the production of alternative solutions and an evaluation mechanism to evaluate the variants. The difference we want to take closer look in this section lies in the respective weighting of the two mechanisms.

In terms of design support systems we can distinguish between direct and iterative methods. Direct methods provide an exact solution for a problem within a finite time period. They are usually based on a comprehensive analytical consideration of the problem. Iterative methods, by contrast, usually deliver only approximate solutions to a problem by gradually adapting them to ideal requirements. Another important differentiating factor is the amount of knowledge initially required to obtain useful results. On the one hand, there are methods that require extensive knowledge for the generative mechanism and comparatively little for the evaluation mechanism. This ensures that an appropriate generative system does not deliver useless results to begin with. On the other hand, there are methods that require little knowledge of the problem for the generative mechanism, but require an elaborate evaluation mechanism. Based on the results of the evaluation, useful alternative solutions are improved in an iterative process until they reach a certain quality.

In this context it is important to clarify the difference between problem knowledge and evaluation criteria. Problem knowledge includes a detailed analysis of an actual state as well as a precise specification of the steps to be performed in order to achieve a particular state. Evaluation criteria, however, specify how properties of a solution should be evaluated. Because it is generally easier to expand the evaluation criteria than to adapt a generative mechanism to a new problem, iterative methods that require little problem knowledge are easier to adapt to changing problems than direct methods.

In general, we can state that in approaches that require extensive problem knowledge, the solution of a problem is already largely determined in the specifications, and in procedures that require little problem knowledge, the solution is not explicitly described in the generative mechanism but framed by the evaluation criteria. Approaches that require extensive problem knowledge include logic programming methods (Coyne, 1988), the classic shape grammar method (Stiny & Mitchell, 1978; Duarte, 2000) and the pure constraint-based systems (Li et al., 2000; Medjdoub & Yannou, 2001). These methods are particularly suitable for processing well-defined problems, such as the imitation of certain forms based on shape grammars. Approaches that require little problem knowledge include methods such as cellular automata (Batty & Xie, 1994; Coates et al., 1996), agent-based systems (Coates & Schmid, 2000), and evolutionary algorithms (EAs; Hower, 1997; Rosenman, 1997; Jo & Gero, 1998). In the following section we examine why from our point of view the latter are more suitable for solving design problems.

2.1. Design problems

In dealing with design support methods, it is necessary to distinguish between operational and nonoperational problems. A problem is operational if it can be described so precisely that one can specify the steps needed to resolve it. The concrete, definable, tangible criteria for describing the problem are called operational criteria. In contrast, nonoperational problems, are "vaguely defined, important elements of the task are unknown or not (quantitatively) definable, their solution criterion is not formulated clearly, the decision process is less concerned with finding solutions, but rather with the specification and differentiation of the problem as well as with the closure of open constraints" (Röpke, 1977). For these problems, there is no clear right or wrong solution. Design problems are usually about nonoperational problems, what distinguishes them from most of the problems of scientific research (Simon, 1969). The solution of nonoperational problems always depends on subjective and contextual aspects.

With regard to the aforementioned distinction between methods that require extensive or little problem knowledge one can add that, although both approaches require operational criteria, the latter does not have to include a complete analysis of the complex problem. Generally it is sufficient to specify heuristics that provide reasonable assumptions for solving a problem but do not guarantee a solution. The fact that a problem solution cannot be guaranteed means that the method employed will make mistakes in the search for solutions. At best, these errors can be used productively to identify new, unexpected solutions. Such system characteristics open the possibility of finding creative solutions to problems, which computer systems at present are not very capable of.

2.2. Evolutionary algorithms

A solution can be called creative when it is not apparent from a detailed description of the problem. This situation illustrates why direct methods, which require extensive problem knowledge are not suitable for supporting creative processes. EAs are so-called heuristic methods that do not guarantee the solution of a problem in any particular case, but considerably reduce the time required to solve the problem. Of course, EA, like all computer programs, only function if all the individual steps to be performed are defined. Compared with other

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

algorithms, however, they require very little problem knowledge. Stochastic processes thereby ensure a certain range of possible variants, which can lead to surprising results.

When solving design problems we are usually dealing with several contradictory requirements. A good solution for such multicriteria problems is always a compromise in the form of a balanced account of the various requirements. In the field of computer-based optimization problems such problems are called multiobjective optimization problems (MOOP). EA have the potential to find all the possible compromise solutions, which are known as pareto-optimal or nondominated solutions (Deb, 2001; Coello et al., 2007).

2.3. Interactive systems

Even with EAs there is still the problem that certain requirements, such as the definition of the objective functions (operational criteria) for a design problem, need to be specified explicitly. A common approach for the input of these criteria is to integrate the user. The importance of interactivity in design support systems has already been emphasized in several similar projects such as in Weinzapfel et al. (1971), Ruch (1978), Michalek and Papalambros (2002), Arvin and House (2002), and Elezkurtaj (2004). The aim when developing interactive systems is to achieve an interplay between the generative system and the subjective ideas of the user.

With regard to the aforementioned distinction between direct and iterative process, one can note that interaction with direct systems is limited to changing initial values and running calculations after each change. With iterative systems, by contrast, interventions during the generative process are permitted, either by changing the rules of the generative mechanism or the evaluation criteria.

3. THE REQUIREMENTS OF AN INTERACTIVE DESIGN SYSTEM

Although operational problems are usually straightforward to solve algorithmically, nonoperational problems depend on the interpretation of a problem by a human. The integration of human capabilities is necessary to reduce certain aspects of the design process to operational problems. Because "all that is possible is the conversion of particular problems from ill-structured to well-structured via the one transducer that exists, namely, man" (Ernst & Newell, 1969, p. 363). Requirements for adequately integrating human capabilities into a generative design system have been discussed in Schneider et al. (2010). These requirements are clarified and extended by the following four desirable system properties:

1. Adaptive: A generative system should allow the definition of new constraints/requirements for a problem at any point in time because designing is not simply searching a solution space but also its continual adjustment. This process of ongoing adjustment does not follow a predefined scheme. Adaptive therefore means that the problem definition can be defined by the users themselves and changed during the problem solving process. A prerequisite for such an adaptive system is circularity.
2. Iterative: Designing is an iterative process that brings forth new problems. Once a solution is found new problems emerge that need to be addressed in further iterations. To solve these new problems, it may be necessary to discard old solutions, which can lead to complex dependencies between different solutions. Design problems are therefore not clearly definable in advance but gradually emerge during the design process. Unlike "presented problem situations" one has to deal with "discovered problem situations" (Getzel & Csikszentmihalyi, 1967).
3. Exploratory: Where contradictory criteria are involved, it is important that a generative system offers a broad range of equivalent compromise solutions. This means that the system offers solutions that are distributed as evenly as possible across the so-called Pareto front (which contains all compromise solutions). This ensures that all relevant solutions are provided. The choice between the compromises must be made by a user unless no additional criteria are introduced.
4. Immediacy: The impact of changes to the generative rules and evaluation criteria should be visible as quickly as possible, that is, ideally in a fraction of a second. Immediacy makes it possible to understand the connection between cause and effect during the search process. In this context it is important to choose a suitable representation for the solution variants to facilitate rapid appraisal of the relevant properties of a variant. In general, this should be done in a graphical or diagrammatic way. User interaction is most successfully realized through graphic representation and not separately via text-based parameters.

To consider these properties, it is important to carefully coordinate the various aspects of the layout system. These aspects relate to the graphical representation, user interaction, the layout solver, and management of data. The interplay of these aspects can be considered as a circular process, where interventions at each level have a direct impact on the overall system (see Fig. 1).

In the following section we concentrate on the description of the layout solver and then, based on this solution mechanism, examine two different types of layout problems (hierarchical and nonhierarchical layout) and discuss their properties within the interactive layout system.

4. LAYOUT SOLVER

In this section we present a mechanism for solving layout problems with the aim of generating layout solutions as quickly as possible. For the user interaction, high computational speed is especially important so that the effects of

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

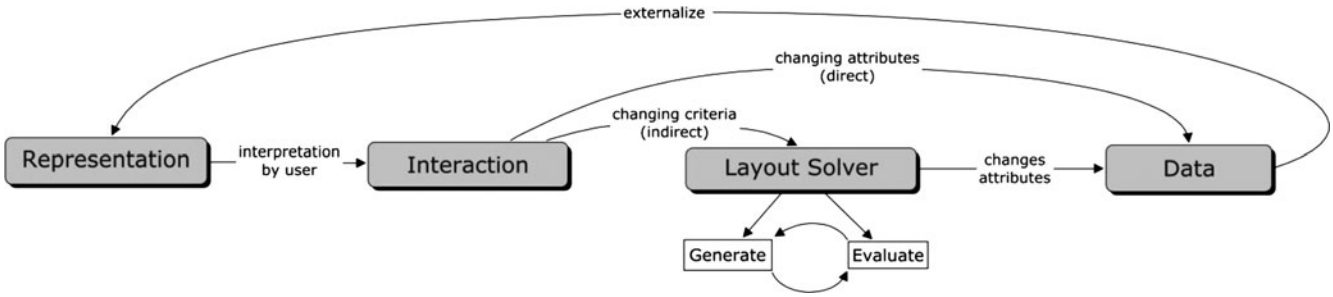


Fig. 1. Partial aspects of the interactive layout system (Schneider et al., 2010).

changes are immediately visible (see point 4 in the previous section). To this end, an additive method is used, which first produces a dense packing of the elements considered, and secondly ensures the topological relations between the elements. For the elements we use rectangles.

4.1. Dense packing

The problem of dense packing occurs when a certain number of spatial elements need to be arranged within a given area without overlapping and with as few gaps as possible. The elements and the bounding area can have fixed values, as, for example, in the case of packing cargo boxes in a truck. In architectural layout design, both the elements to be packed (the rooms) and the surrounding area (the building) can vary within certain limits. In the test scenario discussed in this section, we assume a fixed building outline and deal with the two-dimensional packing of individual rooms that are flexible in their dimensions, within defined minimum and maximum limits, but always have to respect a given area. The sum of the areas to the elements corresponds to the area of the building outline.

The problem to be solved was described formally by Elezkurtaj and Frank (2001, 2002) as follows (Fig. 2): minimize the sum of all intersection areas S_g. This is calculated as the sum of the intersection areas of all elements ($S_i \backslash S_j$) and the weighted sum of the intersection areas arising from the overlapping of the elements with the surrounding area ($S_i \backslash S_u$).

The restriction that rectangles may not extend beyond the limits of the surrounding area, as shown in Figure 2, can be integrated in the calculation of S_g and used for the minimization of S_g. As an alternative to this method overlaps with the bounding rectangle can be avoided in advance by immediately shifting any rectangles that overlap the bounding box within the boundary so that no overlap occurs. This alternative simplifies the calculation of S_g to

$$S_g(X) = f_1^t(X) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (S_i \cap S_j); \quad f_1^t(X) \rightarrow \min, \qquad (1)$$

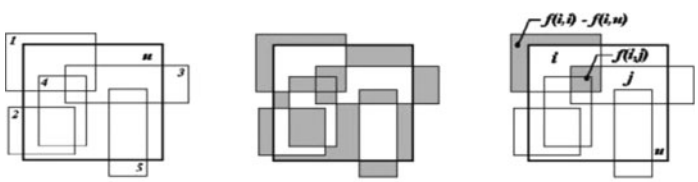
where t indicates the number of the iteration.

4.1.1. Collision detection

For the algorithmic solution of the problem illustrated in Figure 2, the process of collision detection between two geometric elements (rectangles) is useful in order to reduce their degree of overlap by mutually repelling the elements. If two rectangles intersect, as shown in Figure 3, the smallest rebounding vector (v_x or v_y) is chosen, so that each rectangle can be shifted by half of this vector in different directions (v_i and v_j).

The rebounding of the rectangles is an efficient method for reducing the sum of all intersection areas S_g. With this method only the positions of the rectangles are adjusted (location change). For an overlap-free packing of rectangles ($S_g = 0$) it is usually also necessary to change the proportions, that is, length and width, of the rectangles.

For the adjustment of the proportions the same method can be used as used for the rebounding of the rectangles. The only difference is that in this case, the vectors are not used to move the rooms, but to change the width or height of the rectangles.



$$Sg = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (S[i] \cap S[j]) + \lambda \sum_{i=1}^{n} (S[i] \setminus S[u])$$

Fig. 2. A graphical and formal representation of the calculation of the total intersection areas by Elezkurtaj and Franck (2001).

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

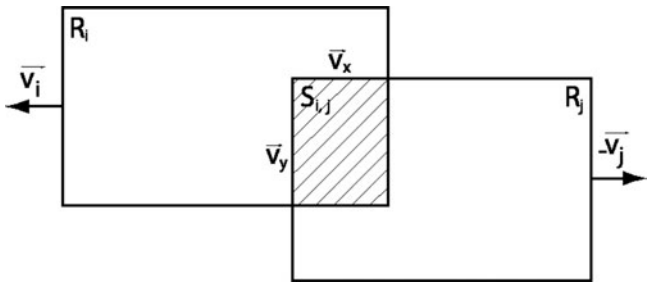


Fig. 3. The calculation of the rebound vectors of two overlapping rectangles.

In the example in Figure 3, the width of R_i would be reduced by the amount of the vector v_i, while simultaneously increasing the height. Thus, the area F_i remains constant, despite the change of proportions. The same is done with the second rectangle R_j.

For the implementation of the previously described procedure the following additional operators are introduced: the factor β for weighting the rebound vectors, the probability ρ_L of executing the rebound function when two rectangles overlap, the factor α for weighting the proportion change vectors and ρ_G, which describes the probability of executing the proportion change function. To summarize, we can represent the two functions for the adjustment of the rectangles as follows:

$$\begin{bmatrix} \Delta G_i^t = \vec{v}_{iG}^{\,t} = \alpha \times \vec{v}_i^{\,t} & \text{if } \rho_G > U \\ \Delta G_i^t = 0 & \text{else} \\ \Delta L_i^t = \vec{v}_{iL}^{\,t} = \beta \times \vec{v}_i^{\,t} & \text{if } \rho_L > U \\ \Delta L_i^t = 0 & \text{else} \end{bmatrix}$$

$$\vec{v}_i' = -\vec{v}_j' = \begin{cases} \begin{pmatrix} 0.5 \times v_{xi}' \\ 0 \end{pmatrix} & \text{if } v_{xi}' > v_{yi}' \\ \begin{pmatrix} 0 \\ 0.5 \times v_{yi}' \end{pmatrix} & \text{else} \end{cases}, \tag{2}$$

where ΔG_i specifies the proportion change and ΔL_i specifies the localization change at the iteration step t and U is a uniformly distributed random number on the interval [0, 1], which is redrawn at every call.

Because the collision process itself does not serve as a satisfactory solution algorithm for dense packing, the following section introduces the technique of evolutionary strategy (ES) and demonstrates how collision detection can be combined with an ES.

4.1.2. Evolutionary strategy

The simplest form of the ES is the so-called two-tier (1 + 1)-ES (Rechenberg, 1994). (μ + λ)-ES represents the general notation for an ES with a (+)-selection, where μ displays the number of parents and λ the number of children in a population *P*. (+)-Selection means that for the next generation the parents remains unchanged and are selected together with the children. As a result, good solutions will not be lost through mutation or crossover, but can be replaced only by better solutions. The (+)-selection is relative continuous because there are no abrupt changes between different solutions. With the (,)-selection, by contrast, the pool for the next generation is only selected from the children. This can lead to a degradation of the fitness values because the best individuals may be changed by mutation and recombination. The advantage of the (,)-selection is that the ES does not get stuck so easily in local optima and can adapt better to changing targets.

The quality of an individual is calculated on the basis of the evaluation function *f*. From λ individuals μ individuals with the highest quality are taken to the next generation while the others are deleted. The evaluation function used in our case is the function for the calculation of S_g given in formula (1). The smaller S_g is, the higher the quality of an individual.

In the simplest form of an ES, an individual a_k with the index k includes with the specific set of object parameters X_k and an evaluation function $f(\boldsymbol{X}_k)$:

$$a_k := (\boldsymbol{X}_k, f(\boldsymbol{X}_k)). \tag{3}$$

A specific object parameter $\boldsymbol{X}_k$ for an individual a_k, in turn, includes the parameters of a single element $\boldsymbol{X}_k = \{X_1, X_2, \ldots, X_i\}$. Here, X_i is also referred to as a gene that encodes the x and y position (px and py), height h and breadth b of a rectangle:

$$X_i = (px_i, py_i, h_i, b_i). \tag{4}$$

The iterative scheme of the ES can be applied to this set of parameters. The mutation operations, which are used to randomly change the object parameters, are specified with

$$f(X_i^t + Z) = (px_i^t + Z, py_i^t + Z, b_i^t + Z). \tag{5}$$

The parameter h_i is calculated automatically after changing b_i to fulfill the condition of keeping the area F_i of the rectangle constant: $h_i = F_i/b_i$.

For the ES, the functions for changing the proportion and location (2) are used as mutation operators. To allow purely random changes, further mutation operators are introduced that cause a random change in the position or proportion with a probability of ρ_Z:

$$px_i'^t = \begin{cases} px_i^t + N_0(\sigma) \times \kappa & \text{if } \rho_{z1} > \mathcal{U} \\ px_i^t + N_0(\sigma) & \text{if } \rho_{z2} > \mathcal{U} \\ px_i^t & \text{else} \end{cases},$$

$$py_i'^t = \begin{cases} py_i^t + N_0(\sigma) \times \kappa & \text{if } \rho_{z1} > \mathcal{U} \\ py_i^t + N_0(\sigma) & \text{if } \rho_{z2} > \mathcal{U} \\ py_i^t & \text{else} \end{cases},$$

$$b_i'^t = \begin{cases} b_i^t + N_0(\sigma) & \text{if } \rho_{z3} > \mathcal{U} \\ b_i^t & \text{else} \end{cases}. \tag{6}$$

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

N_0 denotes a normal distributed random number; the factor κ allows greater changes.

Because of the combination of specific proportion and localization changes (2) and purely random changes, the method developed so far can be termed a hybrid algorithm or hybrid ES.

4.1.3. *Hybrid ($\mu + \lambda$)-ES with recombination operators*

When using several parents per generation $\mu > 1$, ρ parents can be involved in the production of a child. Accordingly the notation of an ES is supplemented by the parameter ρ: ($\mu/\rho + \lambda$)-ES. For our purposes, we consider the discrete recombination (Bäck et al., 1991) in which each object parameter is randomly chosen from one of the ρ parents (Deb, 2001, p. 137; $\rho = 3$):

$$\begin{array}{rcccccc} \textit{parent}1: & X_1^{(1)} & X_2^{(1)} & X_3^{(1)} & X_4^{(1)} & X_5^{(1)} & X_6^{(1)} \\ \textit{parent}2: & X_1^{(2)} & X_2^{(2)} & X_3^{(2)} & X_4^{(2)} & X_5^{(2)} & X_6^{(2)} \\ \textit{parent}3: & X_1^{(3)} & X_2^{(3)} & X_3^{(3)} & X_4^{(3)} & X_5^{(3)} & X_6^{(3)} \\ \textit{recombinant } X': & X_1^{(3)} & X_2^{(1)} & X_3^{(2)} & X_4^{(2)} & X_5^{(1)} & X_6^{(3)} \end{array} \quad (7)$$

Furthermore, the recombination operator r is introduced. For $\rho = 2$, the recombination of two individuals can be described as follows:

$$r(P^t) = a' = X' \qquad X' \in \mathbb{R}^n \qquad \rho = 2$$

$$X_i' = \begin{cases} X_{b,i}, & \mathcal{U}_1 \leq 0.5 \\ X_{c,i}, & \mathcal{U}_i > 0.5 \end{cases} \quad \text{if } \rho_r < \mathcal{U}_2 \quad \forall_i \in \{1, \ldots, n\} \quad (8)$$

where X_b and X_c represent two randomly selected parents selected via the recombination operator r.

At this point it is important to realize that, as noted in (4) above, a gene X_i includes the complete set of properties of a rectangle, which should not be destroyed by recombination. This fact is relevant because we do not want to change predefined area of a rectangle. If we were to subject the properties of the rectangles to recombination, new rectangles would arise as a result of new values for heights and widths, which would necessarily entail new surface areas. If the surface area is allowed to vary, the population would rapidly be dominated by small elements because they cause fewer overlaps. The sum of the area of the rectangles must therefore always be equal to the area of the surrounding rectangle.

After (8), the vectors X_i of two parents are randomly combined for the generation of X_i'. Here, U denotes a uniformly distributed random number on the interval [0, 1], which is redrawn on every call. The recombination operator is not used for the production of every child, but only according to a certain probability ρr. The best system properties, that is, the best possible continuous improvement of solutions along with a fast response to changing conditions can be achieved by using a (+)-selection, where the parents can be mutated but to a lesser extent than their children.

Before we look at methods for taking into account topological relationships, the algorithm developed so far (referred to as standard ES) can be summarized as follows:

Step 1. Initialize a population P with μ individuals.

Step 2. Create λ children by

a. mutation of the parents [purely random mutations (6) and collision detection (2)]
b. recombination of ρ parents for the production of children [(7)–(8) with $\rho r = 0.75$]
c. mutation of the children (recombinants) [(6) and (2)].

Step 3. Select the μ best individuals from parent and children population for the next generation of parents.

Step 4. If the termination criterion is met, terminate the algorithm otherwise go to step 2.

4.2. Topological relations

Topological relation refers to the adjacency of individual rooms. Two rooms are adjacent to each other when there is no distance between them and they touch on one edge each for a certain length. If we interpret a layout as a plan with individual rooms, this requirement should ensure a minimal passage between two rooms. We consider only cases in which rooms are adjacent to each other or not. Other restrictions that specify, for example, how close a room should be to another or that two rooms should be as far apart as possible are not considered here. In the authors' opinion, such an inclusion makes little sense as in the user's perception two mutually adjacent spaces may occur relatively distant from each other when there is no connection (passage) between them.

Formally, the topological constraint can be expressed as follows:

$$f_2^1(X) = \sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij} + c_{ij}); \quad f_2^1(X) \rightarrow \min. \quad (9)$$

For the calculation of the function f the distance d between two rectangles with the indices i and j and the contact lengths c of the rectangles are added. To create the desired topological relationships, the objective function f_2 needs to be minimized.

In this section we will examine three ways to achieve the required topological relationships. In the first, the proportions of the rooms are changed (Fig. 4, right), in the second, the spaces are interchanged (permutation; Fig. 4, center) and in the third virtual springs are drawn between rectangles with an existing topological relationship.

The function for changing proportions of the rooms has already been expressed with the equation ΔG_i (2) and needs no further detailing.

4.2.1. *Permutation*

To achieve a topological relation between elements, one possible method is to exchange elements. The interchange

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

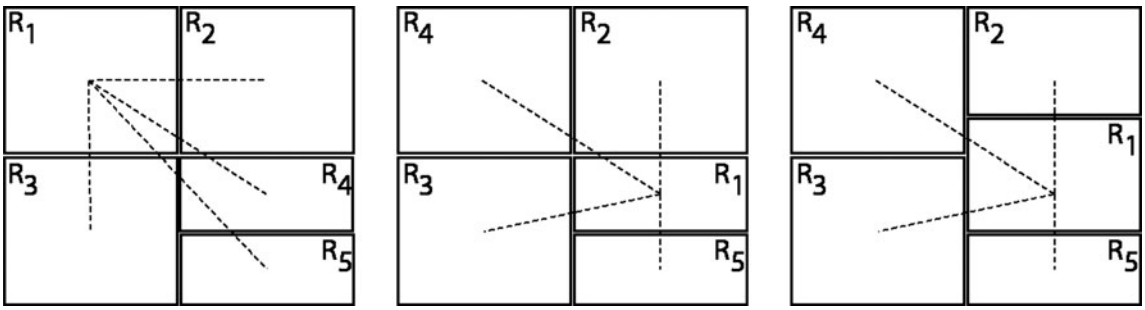


Fig. 4. Successive approximations of the required topological relations (dashed lines) by permutation (see left and center layout configuration) and proportion change (see center and right layout configuration). The representation does not consider the consistency of the surface area of the individual rooms.

of elements and the consequent change in their arrangement is called a permutation.

Each room or each gene X has an index i. Using this index, particular functions can be allocated to each room and the topological relationships can be defined. When using the recombination and mutation operators, we must now ensure that an individual who represents a layout solution has only one particular index for one particular room. For recombination this requirement is ensured using the "partially mapped crossover" method developed by Goldberg and Lingle (1985). The mutation of an individual, either swaps two genes with one another, or a randomly chosen gene order is inverted.

4.2.2. *Attractions using virtual springs*

The calculation of the attraction vectors between two adjacent rectangles is done by means of virtual springs between these rectangles (Arvin & House, 2002). Their tension is calculated in a similar way to the repelling vectors in Section 4.1.1. First, the distance $d_{i,j}$ between the two rectangles is established (Fig. 5). If $d_{i,j} > 0$, that is, they are not directly adjacent, the x or y vector component is checked to see which is greater (10). The attraction of the two rectangles occurs in the direction of the larger vector component. Consequently, the resulting attraction vectors either follow the x or y direction, so that the rectangles only move perpendicular to each other.

$$\vec{d}_i^{\,t} = \gamma \times \vec{d}_i^{\,t}$$

$$\vec{d}_i = -\vec{d}_j = \begin{cases} \begin{pmatrix} 0.5 \times d_{xi} \\ 0 \end{pmatrix} & \text{if } d_{xi} > d_{yi} \\ \begin{pmatrix} 0 \\ 0.5 \times d_{yi} \end{pmatrix} & \text{else} \end{cases} . \qquad (10)$$

The greatest difficulty now is to combine all the methods presented so far in such a way that several objective functions can be optimized simultaneously. How this issue can be handled is explained in the next section.

4.2.3. *Multiobjective optimization problem*

With the introduction of topological relationships as a further restriction, we now have two objective functions f_1 (1) and f_2 (9), which both need to be optimized by the layout system. We are therefore dealing with a MOOP (see Section 2.2). These problems are distinguished into conflicting and nonconflicting objective functions. With conflicting objectives not every criterion can be fulfilled completely; the best that is possible is a compromise. For such a compromise, there are usually many different variants that are described as pareto-optimal. A Pareto-optimal state is characterized by the fact that it is not possible to improve a solution without simultaneously degrading another. In the case of nonconflicting objective functions, each criterion can be fully met. In the two objective functions f_1 and f_2 we are usually dealing with nonconflicting objective functions. This is valid as long as we do not stipulate a topological configuration for f_2, for which there is no overlap-free layout of the elements for f_1.

The classical method of solving a MOOP is to reduce it to simple optimization problems by combining all the criteria into one combined criterion (Deb, 2001, pp. 13 and 46). For the reduction to one criterion, there are several methods of which the weighted sum is the most common and easiest to implement. Because of the difficulties associated with classical methods, such as the problem of weighting each criterion (Deb, 2001, pp. 49–80), these were not used in this case for the MOOP. Rather, we examine an approach to solving the MOOP by combining the vector evaluated genetic algorithm (VEGA) after Schaffer (1985) and the vector-optimized evolution strategy after Kursawe (1990). The two basic methods are described in detail in Deb (2001, pp. 179–189). To differentiate this from other methods, we refer to the proposed technique as Layout-MOES or L-MOES.

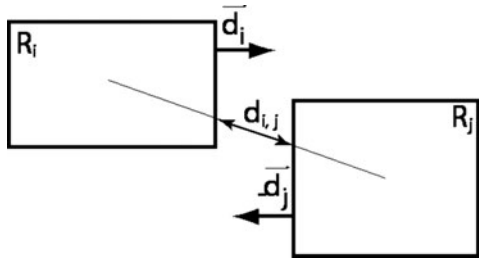


Fig. 5. The calculation of the attraction vectors of two rectangles that are adjacent to each other.

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

In the context of the present problem, ES for MOOP based on the concept of nondominated solutions, such as the Pareto Envelope-Based Selection Algorithm (Corne et al., 2000), have not shown any improvement in system performance. This was mainly due to the choice of selecting the best solutions so far (nondominated set) as the parent population (elite-preserving). This property has led to difficulties in overcoming the local pareto front.

For the EA we use the $(\mu/\rho + \lambda)$-ES, as defined in Section 4.1.3. The significant extension of the standard ES in this instance is how individuals are selected for the next generation. We still create λ recombined and mutated children from μ parents. Afterward, parents and children are selected together by (+)-selection. For this we calculate two fitness values for each of the two objective functions f_1 and f_2 for each individual. Then the child population is sorted according to the first objective function. A certain percentage of individuals is selected for the next generation of parents. Similarly, we proceed for the second objective function. Relation $V = (f_1/f_2)$ defines the number of individuals selected for the two objective functions f_1 and f_2. During the initialization of $V = (\mu*0.1/\mu*0.9)$ was chosen. This means, for example, for $\mu = 10$ that only one individual is selected for the minimum overlapping area and nine are selected for the minimum distance. For lower values of μ at least one individual is selected for f_1. If the number of individuals selected for f_1 is increased from the beginning, the system converges more easily at a local pareto front, which approximately corresponds to a local optimum of optimization problems with only one criterion.

Once a layout has been found that exhibits after f_2 a very small distance sum, we introduce a threshold Ψ_V for changing the ratio V in order to minimize its overlapping areas as soon as possible. It has proven effective to alter the threshold as soon as the individual with the lowest value for f_1 (overlapping areas) falls below a value for f_2 (distances) that is less than twice the number of rooms N_R:

$$V^t = \begin{cases} (\mu \times 0.4/\mu \times 0.6) & \text{if } f_2^t(X_{(f_1 \min)}) < \Psi_v; \quad \Psi_v = 2 \times N_R. \\ (\mu \times 0.1/\mu \times 0.9) & \text{else} \end{cases} \tag{11}$$

The definition of the threshold by the number of rooms works well in the test system, but has to be changed with the scale of the system. We use a similar rule for the selection method. As long as a parent individual a has a value for f_2, which is greater than the threshold Ψ_V, a (,)-selection is applied, otherwise a (+)-selection is applied with collision detection and random movement of parents, which we already introduced in the standard-ES:

$$\text{Selection } s = \begin{cases} (\,,\,) & \text{if } \; f_2^t(X_k) < \Psi_s; \quad \Psi_s = 4 \times N_R. \\ (+) & \text{else} \end{cases} \tag{12}$$

To summarize, the algorithm for the L-MOES can be specified as the following:

Step 1. Initialize a population P with μ individuals.

Step 2. Create λ children by

a. only for (+) selection, see (12): mutation of the parents [purely random mutations (6) and collision detection (2)]
b. Recombination of ρ parents to produce the children [(7)–(8) $\rho_r = 0.75$]
c. Mutation of the children (recombinants) [using (6) and (2), permutation (see Section 4.2.1) and virtual springs (10)].

Step 3. Select after (3) and (9), the μ best individuals according to the rules (11) and (12) for the next generation of parents.

Step 4. If the termination criterion is met, terminate the algorithm otherwise go to step 2.

5. HIERARCHICAL STRUCTURED LAYOUT PROBLEMS

In the following, the layout solver presented above is applied to two different types of layout problems. A distinction is drawn between hierarchical and nonhierarchical layout problems. It is shown that for hierarchical layout problems, it is necessary to define relations across hierarchical boundaries. Finally, the advantages and disadvantages for the calculation of both types of layout problems are discussed.

5.1. Hierarchical structuring of elements and the need for cross-border relations

In the previous sections we dealt with the arrangement of elements on a certain level of scale under the premise of ensuring their topological relations. This type of layout problem is referred to as a simply structured or nonhierarchical structured layout problem (NHLP). Another approach lies in the hierarchical structuring of a layout problem (HLP). A layout is termed hierarchically structured when its elements are nested into each other several times. This means that elements can be located within other elements and itself be part of other elements. As an example of an HLP, we can consider a residential building where the corridor and several apartments are arranged on one floor. Within each of the flats, several rooms have to be accommodated and inside these rooms the furniture needs to be arranged. It is worth noting, that the means of hierarchical structuring depends on the situation and can be defined freely by the designer. It is possible to describe a house either as single apartments, or to structure it in certain zones (such as private and public). The concept of HLP was also considered by Flemming et al. (1992, p. 20), among others.

With regard to the relations between the elements, for HLP it is necessary to establish cross-border relations. Cross-border designates the crossing of hierarchical levels or boundaries. Relations refer to the topological relations, which were described in Section 4.2. The need for cross-border relations is obvious since, as shown in Figure 6, elements of one hier-

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

archical level may have relations with elements outside the level. For example, in a residential house, a cross-border relationship is required to connect the corridor of the apartment with the corridor of the house. Consequently, cross-border relations are indispensable for adequately describing layout problems with hierarchically nested elements.

Thus, on the one hand, HLPs include nested elements. On the other hand, they can include relationships between elements that are located on different levels of hierarchy. The extent to which HLPs with cross-border relations can be solved with the layout solver and what needs to be considered will be clarified below.

5.2. Explicit and implicit internal and external relations

Considering the interaction of a user with the layout system we can distinguish between explicit and implicit requirements (constraints) for a layout. Explicit requirements are requirements that a user explicitly specifies via inputs. Implicit requirements however are requirements that result from the explicit requirements and were not explicitly defined. In the example of defining topological relations in a HLP, this becomes obvious: if two rooms are linked together, there are basically two different types of relations: internal and external. Internal relations are relations of elements within a certain hierarchy level, such as the topological relationship of rooms within an apartment. External relations are relations of elements from different hierarchical levels, for example, the relation of a corridor in an apartment to the access hallway of the building in which the apartment is located. In this case, the apartment and the house's access hallway lie on the same hierarchical level, but the house hallway and the apartment's corridor lie on different levels. This means that the house's access hallway must be adjacent to both: to the corridor of the apartment and to the apartment itself. Because the spatial relation house access hallway–corridor was not explicitly given, this means that this relation has to be created implicitly by linking the apartment corridor with the house's access hallway. In a HLP, two different types of relations exist: internal and external. Relations between elements on the same hierarchical level are called internal; relations between elements on different hierarchical levels are called external. Both internal and external relations can be explicitly specified (by the user). Implicit relations arise only indirectly through the specification of external relations. Here, both internal implicit, as well as external implicit relations, which arise in deeply nested layout problems, can occur (see Fig. 7).

If the layout solver presented in Section 4 is applied to HLPs, various cases have to be considered. For layout problems where there are only internal relations, the mechanism is readily applicable. For layout problems in which external relationships are present, these must be treated differently. The previously established distinction between implicit and explicit relations is solely important for the user interaction because the user only considers the explicit requirements he or she stipulates. The subsequent automatically created implicit requirements are invisible to the user. The layout solver treats implicit and explicit requirements equally.

To better understand these special cases, it helps to clarify the procedure of the layout solvers when solving HLPs. Once an element contains other elements, the solver tries to arrange the subelements within the superior element. If there is more than one superior element, these are considered simultaneously. Several optimization processes then run in parallel. The HLP is divided into several NHLP. By way of example, this is shown in Figure 8 and Figure 9. The complex problem in Figure 8 is divided in into two problems (Fig. 9): the arrangement of $R_{1.1}$, $R_{1.2}$, and $R_{1.3}$ in R_1 and the arrangement of $R_{1.2.1}$, $R_{1.2.2}$, and $R_{1.2.3}$ in $R_{1.2}$.

With regard to achieving dense packing (see Section 4.1), both problems shown in Figure 9 (R_1 and $R_{1.2}$) are solvable separately. The creation of dense packing in R_1 can take place independently of the one in $R_{1.2}$, because for the subelements $R_{1.2.1}$, $R_{1.2.2}$, and $R_{1.2.3}$ it only means that the height h and width w of the surrounding area $R_{1.2}$ can change while packing.

When considering the topological relations, we have to distinguish between internal and external relations. Although internal relations are treated as described in Section 4.2, external relations must be considered separately. First, it is important that external relations impact in only one direction, from the deeper to the higher hierarchical level. On the one hand, this is related to the fact that an implicit link (from $R_{1.1}$ and $R_{1.2}$) already ensures the requirement for corresponding exter-

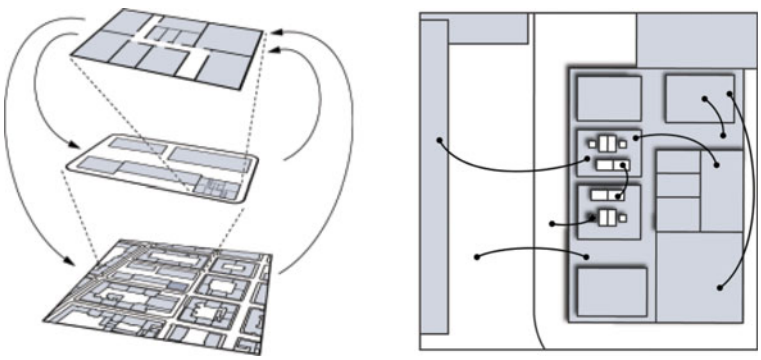


Fig. 6. The hierarchical structure (left) and cross-border relations (right).

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

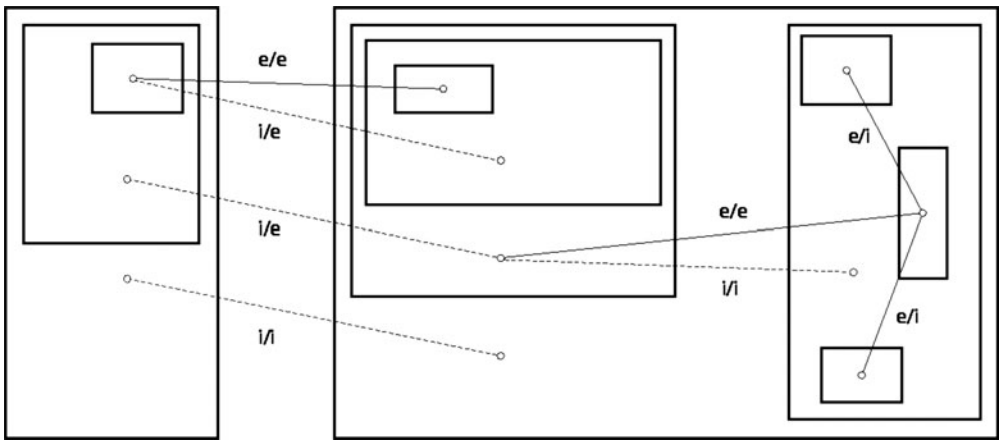


Fig. 7. Different types of relations in a problem description of hierarchical structuring of a layout problem; e/i, explicit-internal; e/e, explicit-external; i/i, implicit-internal; i/e, implicit-external.

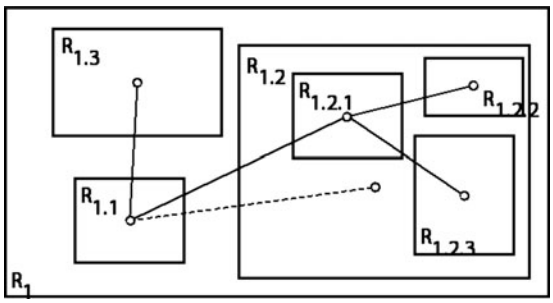


Fig. 8. The implicit and explicit relations in a simple hierarchical structuring of a layout problem.

nal adjacency (of $R_{1.1}$ and $R_{1.2.1}$). On the other hand, potential feedbacks between the subproblems are avoided by considering only one direction. For the solution of R_1 the topological relations between $R_{1.1}$ and $R_{1.2}$, and $R_{1.1}$ and $R_{1.3}$ have to be taken into account. For solving $R_{1.2}$, however, it is also necessary to consider the external relation to $R_{1.1}$ in addition to the internal relations. During the solution of $R_{1.2}$ the external element $R_{1.1}$ is locked for changes in location and size. The changes to $R_{1.1}$ take place in R_1. The solver for $R_{1.2}$ treats $R_{1.1}$ passively, trying to place the item $R_{1.2.1}$ as close as possible to $R_{1.1}$.

For the calculation of the distance between the elements that are connected by an external relation, the sum of the distances of the parent element to the external elements needs to be subtracted from the sum of the all the considered distances. In our example, this means that for the calculation of the distance between $R_{1.2.1}$ and $R_{1.1}$, the distance between $R_{1.2}$ and $R_{1.1}$ needs to be subtracted.

5.3. Performance comparison of HLP versus NHLP

In this section HLP and NHLP are compared with each other in terms of their properties when solving a given test scenario. This scenario consists of generating a layout for a residential building with three apartments (Fig. 10). For the formulation of an HLP, the first level of the hierarchy involves the arrangement of three apartments and one hallway. On the second level, the individual rooms need to be arranged within each apartment. The internal relations on the hierarchical level of each apartment are shaped radially. The corridor within in each apartment has an external relation to the hallway of the house. These external relations generate implicit internal relations from the apartments to the corridor of the building (Fig. 10, left). The formulation of the NHLP consists in the arrangement of individual elements for the rooms of the flats

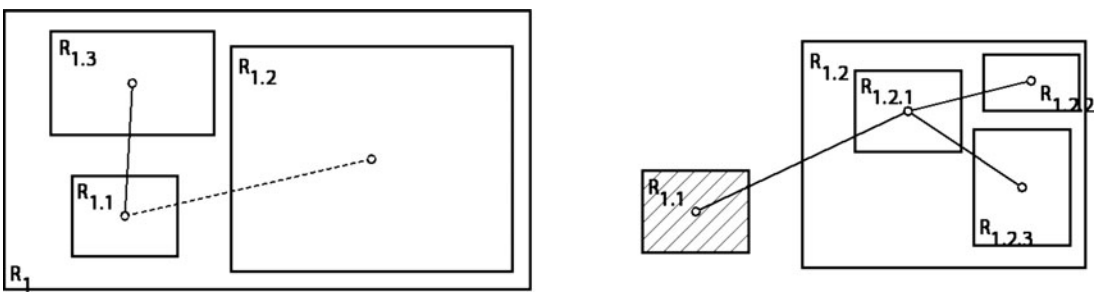


Fig. 9. Different kinds of relations connecting different level hierarchies for (left) the partial problem R_1 and (right) the partial problem $R_{1.2}$.

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

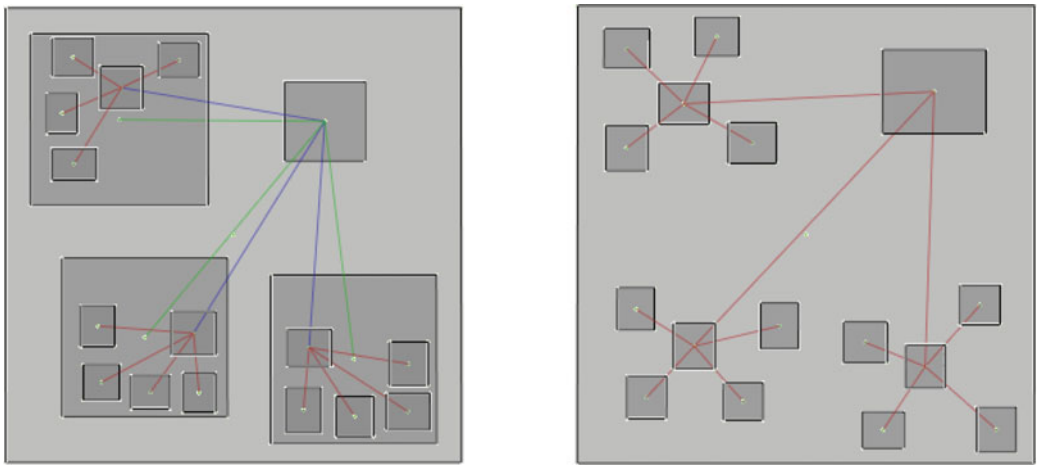


Fig. 10. A schematic representation of the test scenarios each with 16 rooms: (left) hierarchical structuring of a layout problem and (right) nonhierarchical structuring of a layout problem. Red lines are internal relationships, blue lines are external relationships, and green lines are implicit relationships. [A color version of this figure can be viewed online at http://journals.cambridge.org/aie]

and the house corridor to a single level. Consequently, only internal relations are required (Fig. 10, right).

The focus of the comparison between the two test scenarios (HLP vs. NHLP) lies on the speed of convergence through which the system finds a useful solution. This is crucial to ensure good user interaction with the system. For a definition of what can be understood as a useful layout solution, the fitness values for f_1 (overlapping areas) and f_2 (distances) are normalized to the interval [0, 1]. This normalization is carried out by assuming maximum values for f_1 and f_2 that lead to a scaling of the fitness values, which primarily covers the value range of useful solutions. For the following analysis and the diagrams in Figure 11 the following scaling functions were used:

$$\begin{aligned} f_1(X)_{\text{norm}} &= f_1(\boldsymbol{X})/(b_p \times h_p \times 1/3), \\ f_2(X)_{\text{norm}} &= f_2(\boldsymbol{X})/N_{\text{rel}}(b_p \times h_p \times 1/10). \end{aligned} \tag{13}$$

where b_p and h_p specify the breadth and height of the superior rectangle and N_{rel} describes the number of relations in a layout problem. Based on this normalization it is now possible to specify that the layout solutions can be described as useful when the corresponding fitness values fall below certain values:

$$\begin{aligned} f_1(\boldsymbol{X})_{\text{norm}}^{\text{accept}} &< 0.3, \\ f_2(\boldsymbol{X})_{\text{norm}}^{\text{accept}} &< 0.1. \end{aligned} \tag{14}$$

The speed of convergence with which the system provides useful solutions, is counted in generations t of the L-MOES. For rapid user interaction values in (14) should be reached after $t = 20$ generations [on a computer with average performance (2.4 GHz) this requires ~5 s]. The system should respond with roughly the same speed to interactions by a user, which can be interpreted as interferences to a once optimized layout.

With the definitions made so far we can now analyze the diagrams in Figure 11. The diagrams represent measurements that were collected during 100 repetitions of the layout system based on the two test scenarios in Figure 10. In both graphs in Figure 11 the data points indicate the fitness values for f_1 (overlap) of each individual run. The green solid line represents the average values for f_1, and the blue dotted line represents the mean values for f_2 after 100 runs.

Comparing the two diagrams in Figure 11 it can be seen that in HLP (left diagram) the mean curve for both f_1 and f_2 falls steeply and reaches lower levels. According to the previous definitions, this means that by using HLP good solutions can be generated in an acceptable speed for appropriate user interaction. In contrast, the two mean curves for NHLP (Fig. 11, right diagram) are much flatter and do not at any time reach values as low as those for HLP. Furthermore, viable solutions are only found for f_1 after an average of about $t = 40$ generations, which is too slow for responsive user interaction. In addition, the distribution of data points exhibits a much wider spread in NHLPs, which indicates that the system often finds no viable solutions in the considered range of generations. In both diagrams in Figure 11, the average curve for f_2 (distances) returns good results quickly, which can be traced back to rules (11) and (12) of the L-MOES, which first prefer the optimization of f_2.

To give an idea of the layout results that lies behind the diagrammatic representation in Figure 11, Figure 12 shows geometric solutions for an HLP (left) and NHLP (right). The effects of the use of different strategies are shown by the fact that in the HLP (left), the apartments are organized within the parent rectangles that were inserted by applying a hierarchy to the problem. In contrast, in the NHLP the rooms are not attached to superior boundaries, resulting in a spatial entanglement of the three apartments.

Finally, the advantages and the disadvantages of the compared strategies (NHLP and HLP) can be summarized as fol-

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

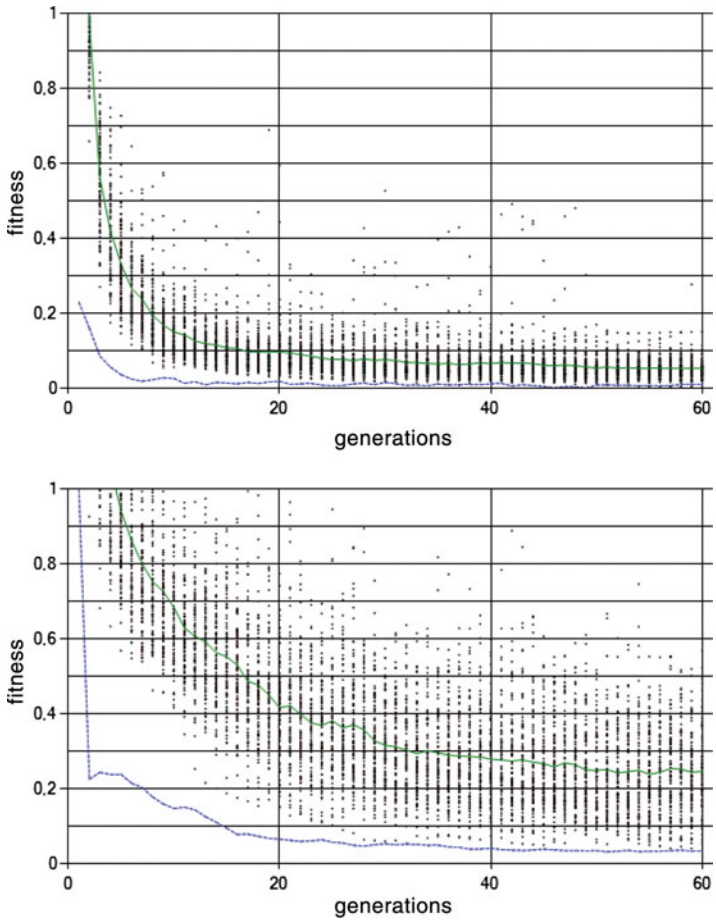


Fig. 11. Analysis diagrams for the generation of layouts via L-MOES after 100 runs of the program, each with $t = 60$ generations. The green solid line represents the average values for f_1, and the blue dotted line represents the mean values for f_2. The diagrams show the analysis of the (top) hierarchical structuring of a layout problem and (bottom) that of the nonhierarchical structuring of a layout problem. The system was initialized with the corresponding scenario from Figure 10. [A color version of this figure can be viewed online at http://journals.cambridge.org/aie]

lows. The main weakness of NHLPs lies in the slow generation of viable solutions. This is especially problematic when a large amount of elements with many relations needs to be arranged. The main advantage of NHLP is that there is no prior hierarchy applied to the elements, resulting in solutions that remain unaffected by certain conventions, which are otherwise given in the indication of hierarchies. In the example shown, such a convention represents the grouping of rooms of an apartment within a rectangular superior element (Fig. 12).

With regard to the interactive layout system presented in this paper, the usage of HLPs delivers two significant advantages. First, in terms of user interaction the use of hierarchies provides a better overview of the elements to be arranged. This is advantageous because a user can usually only consider a limited number of elements simultaneously (Miller, 1956). Second, complex problems with many elements and relationships are solved much faster compared with NHLPs. A disadvantage is that because of the *a priori* definition of hierarchies, certain solutions are excluded which compromises the requirement in Section 3 that a system should be as exploratory as possible. Consequently, the spectrum of equivalent compromise solutions in HLP is restricted by reducing the search and solution space. The smaller search space also partially explains the higher speed of convergence in HLPs.

It was mentioned in Section 5.1 that the idea of HLPs is not new. For example Coyne and Flemming (1990) presented it in the ABLOOS framework for layout design. ABLOOS is a hierarchical extension of a system called LOOS (Flemming, 1989), which introduces orthogonal structures for the representation of loosely packed arrangements of rectangles. Although these systems use a different generative mechanism, we can approximately compare their performance with the system presented in this paper.

Flemming et al. (1992, p. 18) evaluate the performance of the LOOS system. It generates 26 solutions for a test problem with 8 rooms and 11 topological relations in 188 s. Of course, this system would perform faster on current computers, but there is reason to doubt that the generation time for a more

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

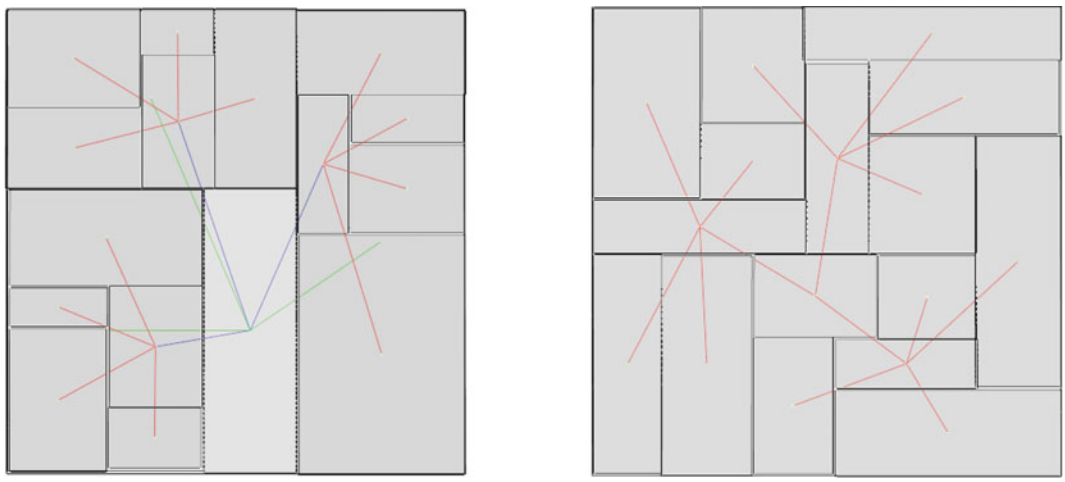


Fig. 12. Typical solutions in the calculation of (left) a hierarchical structuring of a layout problem and (right) a nonhierarchical structuring of a layout problem via L-MOES after $t = 60$ generations. The corresponding fitness values of both solutions fall below the thresholds defined in (14); accordingly both solutions are defined as useful. [A color version of this figure can be viewed online at http://journals.cambridge.org/aie]

complex problem similar to that presented in Section 5.3 (Fig. 10) with 16 rooms and 15 topological relations, would be faster than 5 s, because LOOS calculates all possible solutions using its hierarchical generate and test method. The L-MOES presented in Section 4.2.3 needs approximately 5 s to generate one solution for a given problem. Although the generation of only one solution may be considered as a disadvantage in other contexts, for the requirements of interaction time it can be regarded as suitable strategy.

Harada et al. (1995) have shown a method for fluid user interaction with discrete constrained layout problems (LOOS can be classified in this category). They trigger a local discrete search, during which certain transformations may be performed. The transition to a new state is visualized using animated visual effects. Such techniques show, that one can also use other generative mechanisms to create a layout system with appropriate user interaction. The advantage of the L-MOES presented here is that it works with continuous parameters that can be manipulated directly by the user, so we do not need further techniques to achieve proper user interaction.

6. CONCLUSION AND OUTLOOK

The present work was developed within a research project that deals with computer-based design methods to support the creative design process. The project aims to develop a system to solve layout problems in architecture, which quickly provides viable results without initially requiring extensive problem knowledge. By gradually adding problem knowledge (in the form of constraints) these results can be elaborated interactively. This approach is contrary to those that require extensive problem knowledge at the very beginning of the solving process and are thereby less able to adapt to constantly changing requirements during the design process.

To solve layout problems, an iterative approach based on EAs was chosen. The method ensures a nonoverlapping arrangement of elements and their topological relationships. Afterward the hierarchical structuring for describing layout problems was discussed. This type of layout description was compared with a nonhierarchical description. Their advantages and disadvantages with regard to the interactive layout system were discussed. It was shown that HLP delivers good results with regard to the speed of convergence. The system is capable of finding viable solutions, even in complex problems after a reasonable time.

In this study we have used an abstract test scenario with a configuration of rectangles to focus on the basic functionality of the system presented. The results shown in Figure 12 can be considered as first design sketches, for example, for the layout of a residential building. Of course, some important aspects are missing, for example, the proportions of rooms need to be considered, the orientation of the rooms in certain directions has to be defined, and the placement of windows needs to be added. From our point of view, however, these are comparatively small problems that can be solved relatively easily in a next development stage based on a flexible and robust layout system.

In this exploration, we have concentrated on the use of only rectangular elements. The use of free forms would increase the search and solution space considerably and thus reduce the convergence speed significantly. A fast generation of solutions for fluent user interaction would not at present be possible. Future work will, however, examine ways for dealing with nonrectangular forms (Schneider et al., 2011).

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press

ACKNOWLEDGMENT

This project is funded by the German Research Council (DFG) under Funding Number DO-551/19-1.

REFERENCES

Arvin, S.A., & House, D.H. (2002). Modeling architectural design objectives in physically based space planning. *Automation in Construction 11*, 213–225.

Bäck, T., Hoffmeister, F., & Schwefel, H.-P. (1991). A survey of evolution strategies. *Proc. 4th int. Conf. Genetic Algorithms*.

Batty, M., & Xie, Y. (1994). From cells to cities. *Environment and Planning B: Planning and Design 21*(*7*), 31–48.

Coates, P., Healy, N., Lamb, C., & Voon, W.L. (1996). The use of cellular automata to explore bottom up architectonic rules. In *Eurographics '96* (Rossignac, J., & Sillion, F., Eds.). Poitiers: Blackwell.

Coates, P., & Schmid, C. (2000). *Agent based modelling*. London: University of East London School of Architecture, Centre for Computing & Environment in Architecture. Accessed at http://uelceca.net/research/ECAADE/agentnotes%20the%20paper398liverpool%201999.pdf

Coello, C.A., Lamont, G.B., & Veldhuizen, D.A.V. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems* (2nd ed.). Berlin: Springer.

Corne, D.W., Knowles, J.D., & Martin, J.O. (2000). The Pareto envelope-based selection algorithm for multiobjective optimization. *Proc. Nature VI Conference Parallel Problem Solving*.

Coyne, R.F., & Flemming, U. (1990). Planning in design synthesis: abstraction-based LOOS. In *Artificial Intelligence in Engineering V* (Gero, J.S., Ed.), Vol. 1, pp. 91–111. New York: Springer.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. New York: Wiley.

Duarte, J.P. (2000). *Customizing Mass Housing: A Discursive Grammar for Siza's Malagueira Houses*. Cambridge, MA: Massachusetts Institute of Technology.

Elezkurtaj, T. (2004). *Evolutionäre Algorithmen zur Unterstützung des kreativen architektonischen Entwerfens*. Wien: TU-Wien.

Elezkurtaj, T., & Franck, G. (2001). Evolutionary algorithm in urban planning. *Proc. CORP 2001, Information Technology in Urban and Spatial Planning*, Vienna.

Elezkurtaj, T., & Franck, G. (2002). Algorithmic support of creative architectural design. *Umbau 19*, 129–137.

Ernst, G., & Newell, A. (1969). *GPS: A Case Study in Generality and Problem Solving*. New York: Academic Press.

Flemming, U. (1989). More on the representation and generation of loosely packed arrangements of rectangles. *Environment and Planning B: Planning and Design 16(3)*, 327–359.

Flemming, U., Baykan, C.A., Coyne, R.F., & Fox, M.S. (1992). Hierarchical generate-and-test vs. constraint-directed search: a comparison in the context of layout synthesis. In *Artificial Intelligence in Design* (Gero, J.S., Ed.), pp. 817–838. Boston: Kluwer Academic.

Getzels, J.W., & Csikszentmihalyi, M. (1967). Scientific creativity. *Science Journal 3*, 80–84.

Goldberg, D.E., & Lingle, R. (1985). Alleles, loci and the traveling salesman problem. *Proc. int. Conf. Genetic Algorithms*.

Harada, M., Witkin, A., & Baraff, D. (1995). Interactive physically-based manipulation of discrete/continuous models. *Proc. 22nd Annual Conf. Computer Graphics and Interactive Techniques*.

Hower, W. (1997). Placing computations by adaptive procedures. *Artificial Intelligence in Engineering 11*, 307–317.

Jo, J.H., & Gero, J.S. (1998). Space layout planning using an evolutionary approach. *Artificial Intelligence in Engineering 12*, 149–162.

Kursawe, F. (1990). A variant of evolution strategies for vector optimization. In *Parallel Problem Solving from Nature*, Vol. 1, pp. 193–197. Berlin: Springer.

Lawson, B. (2006). *How Designers Think: The Design Process Demystified* (4th ed.). Oxford: Architectural Press.

Li, S.-P., Frazer, J.H., & Tang, M.-X. (2000). A constraint based generative system for floor layouts. *Proc. 5th Conf. Computer Aided Architectural Design Research in Asia (CAADRIA)*, Singapore.

March, L., & Steadman, P. (1971). *The Geometry of Environment: An Introduction to Spatial Organization in Design*. London: R.I.B.A. Press.

Medjdoub, B., & Yannou, B. (2001). Dynamic space ordering at a topological level in space planning. *Artificial Intelligence in Engineering 15*, 47–60.

Michalek, J., & Papalambros, P. (2002). Interactive design optimisation of architectural layouts. *Engineering Optimization 34*, 485–501.

Miller, G.A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review 63*, 81–97.

Rechenberg, I. (1994). *Evolutionsstrategie '94*. Stuttgart: Frommann Holzboog.

Röpke, J. (1977). *Die Strategie der Innovation: Eine systemtheoretische Untersuchung der Interaktion von Individuum Organisation und Markt im Neuerungsprozess*. Tübingen: Mohr Siebeck.

Rosenman, M.A. (1997). The generation of form using an evolutionary approach. In *Evolutionary Algorithms in Engineering Applications* (Dasgupta, D., & Michalewicz, Z., Eds.). New York: Springer.

Ruch, J. (1978). Interactive space layout: a graph theoretical approach. *Proc. 15th Design Automation Conf.*

Schaffer, J.D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. *Proc. 1st Int. Conf. Genetic Algorithms*.

Schneider, S., Fischer, J.-R., & König, R. (2010). Rethinking automated layout design—developing a creative evolutionary design method for the layout problems in architecture and urban design. *Proc. Design Computing and Cognition DCC10*, Stuttgart, Germany.

Schneider, S., Koenig, R., & Pohle, R. (2011). Who cares about right angles? Overcoming barriers in creating rectangularity in layout structures. *Proc. eCAADe 2011: Respecting Fragile Places*, Ljubljana, Slovenia.

Simon, H.A. (1969). *The Sciences of the Artificial* (1st ed.). Cambridge, MA: MIT Press.

Stiny, G., & Mitchell, W.J. (1978). *The Palladian grammar. Environment and Planning B: Planning and Design 5(1)*, 5–18.

Weinzapfel, G., Johnson, T.E., & Perkins, J. (1971). IMAGE: an interactive computer system for multi-constrained spatial synthesis. *Proc. 8th Design Automation Workshop*.

Whitehead, B., & Eldars, M.Z. (1964). An approach to the optimum layout of single-storey buildings. *Architects Journal* 1373–1380.

Reinhard Koenig studied architecture and urban planning and completed his PhD thesis in 2009 at the University of Karlsruhe. He has worked as a Research Assistant at the Chair for Computer Science in Architecture (InfAR) at the Bauhaus-University Weimar since 2007 and heads research projects on the complexity of urban systems as well as the development of evolutionary design methods. In October 2009 he was appointed Interim Professor of the Chair for Computer Science in Architecture.

Sven Schneider studied media informatics and architecture at several universities, completing his diploma in architecture in 2009 at the Bauhaus-University Weimar, where he now works as a Research and Teaching Assistant. His primary research interests lie in supporting design processes with computational tools for generative as well as analytic purposes. Currently he is writing his PhD dissertation, investigating ways in which spatial characteristics (measured, e.g., using space syntax methods) can be integrated into the automatic generation of layouts.

https://doi.org/10.1017/S0890060412000030 Published online by Cambridge University Press