*Artificial Intelligence for Engineering Design, Analysis and Manufacturing* (2003), *17*, 273–285. Printed in the USA.

DOI: 10.1017/S089006040317401X

# Self-regulatory hierarchical coevolution

MIKE ROSENMAN AND ROB SAUNDERS

Key Centre of Design Computing and Cognition, School of Architecture, Design Science and Planning, Faculty of Architecture, University of Sydney, Sydney, New South Wales, Australia

(RECEIVED January 24, 2003; ACCEPTED August 28, 2003)

**Abstract**

An evolutionary model for nonroutine design is presented, which is called hierarchical coevolution. The requirements for an evolutionary model of nonroutine design are provided, and some of the problems with existing approaches are discussed. Some of the ways in which these problems have been addressed are examined in terms of the design knowledge required by evolutionary processes. Then, a synthesis of these approaches as a hierarchical coevolutionary model of nonroutine design is presented and the manner in which this model addresses the requirements of an evolutionary design model is discussed. An implementation in the domain of space planning provides an example of a hierarchical design problem.

**Keywords:** Evolutionary Design; Hierarchical Coevolution; Nonroutine Design; Self-Regulation

## 1. INTRODUCTION

Nonroutine designing can be characterized by a lack of knowledge about the relationships between given design requirements and possible forms satisfying those requirements (Rosenman, 1997). Note that the term "nonroutine" is relative. The determining characteristic is that, as far as the designing agent is concerned, there is a lack of knowledge between the problem and the solution; hence, it does not merely copy a preexisting solution. As a consequence, computational systems that use general-purpose, knowledge-lean search processes, particularly evolutionary algorithms, have proven popular as models of nonroutine design (Bentley, 1999).

### 1.1. Evolutionary algorithms for nonroutine design

Evolutionary algorithms have proved to be good general-purpose, knowledge-lean search processes for nonroutine design problems. An important feature of evolutionary algorithms is that they efficiently balance exploration and exploitation in the search of design spaces, allowing the effective searching of ill-defined design spaces typical of nonroutine design problems.

Despite these successes, the application of evolutionary algorithms to nonroutine design can still be problematic. A common problem is that traditional evolutionary algorithms (e.g., simple genetic algorithms) evolve complete objects rather than the more naturally described components of the design. The need to evolve holistic design solutions can require complex forms of context-sensitive reasoning or the use of ingenious data structures to represent the evolving designs. This article focuses on the task of specifying design knowledge in an evolutionary algorithm.

## 2. KNOWLEDGE IN EVOLUTIONARY DESIGN SYSTEMS

There are two common ways that a designer can specify design knowledge in the application of an evolutionary algorithm: first, in the specification of a fitness function, and, second, in the design of the data structures to be manipulated and the operators that manipulate them. Alternative ways that a designer can add knowledge to an evolutionary algorithm either require run-time interaction (i.e., supervision) or an intimate knowledge of the workings of the algorithm itself.

The specification of a fitness function is the most common way that a designer encodes design knowledge for an

Reprint requests to: Mike Rosenman, Key Centre of Design Computing and Cognition, School of Architecture, Design Science and Planning, Faculty of Architecture, University of Sydney, NSW 2006, Australia. E-mail: mike@arch.usyd.edu.au

evolutionary algorithm. The fitness function drives the evolutionary process toward satisfactory solutions and thus has a large influence on the efficiency of the process. The proper specification of a fitness function is critical.

Even moderately complex designs often require the specification of multiple criteria as fitness functions. Multiple criteria expressed as separate fitness functions can be problematic, because they require some means of combining fitness measures so that an evolutionary algorithm may use them. This problem is not confined to the application of evolutionary algorithms to design; it has affected all multicriteria optimization methods (e.g., see Cohon, 1978). In evolutionary algorithms several approaches exist, each with their own limitations and benefits (see Fonseca & Fleming, 1995, for an overview of multicriteria optimization research).

Multicriteria fitness functions can pose computational problems for evolutionary systems. As the number of criteria to be evaluated increases, the size of the Pareto optimal set increases exponentially. For complex problems the demands of the increasing population size produces a heavy computational burden. Badly designed multicriteria fitness functions can result in the spread of a population across the fitness landscape away from the desired optima, resulting in an explosion in population size without an improvement in the best designs. Conflicts of this type can be hard to identify and can be particularly difficult to rectify.

The approach taken in this work has been to limit the need for multicriteria fitness functions by restricting the scope and objectives of any one evolutionary process. The pragmatic use of a hierarchy of design problems that mirror the natural decomposition of a problem by a designer permits the number of criteria that must be considered for any particular design problem to be relatively small compared with the number of criteria affecting the complete design. This approach avoids the rapid growth in population size for complex design problems and limits the unintended interaction of fitness functions mentioned above.

Designing domain-specific data structures to be evolved is a more subtle way of adding knowledge to an evolutionary algorithm, but it is just as important as other methods that are used. The design of the data structure can drastically limit the forms that can be evolved. Careful consideration of the design requirements can lead to better representations of the form to be evolved. In particularly complex cases, the design of the data structure may appear to be as difficult and ill defined as the original design problem that the evolutionary algorithm is constructed to solve.

## 3. KNOWLEDGE RICH VERSUS KNOWLEDGE EFFICIENT

From a knowledge-level perspective, there are two ways that evolutionary algorithms could be improved: by incorporating additional knowledge or by increasing the efficiency of the system's use of its knowledge. The first approach is to add domain knowledge to the algorithm outside of the fitness function and data structures. The second approach is to make the algorithm more efficient in its use of the domain knowledge that it already has in its fitness functions and data structures.

### 3.1. Knowledge-rich evolution

Some ways of changing an evolutionary algorithm to add design knowledge include the specification of the selection process used to select individuals for reproduction, operators that perform reproduction, termination criteria to halt an evolutionary run, and the type of evolutionary algorithm.

Knowledge can be added to an evolutionary algorithm by using custom reproductive processes that increase the likelihood that good, or at least viable, individuals are created. This approach is particularly useful if standard evolutionary algorithms prove inefficient because many invalid individuals are created that must still be evaluated, slowing the exploration of useful areas of the design space. New reproductive operators can be as simple as ones that guard against creating invalid data structures, or they can be as complex as local search algorithms that perform local hill climbing, as in the case of memetic algorithms (Moscato, 1989). This approach is quite flexible and can be used to add knowledge that is specific only to the type of data structure being evolved or to the domain.

Some restraint should be exercised when applying domain-specific reproductive operators. One of the advantages of using evolutionary algorithms is that they can discover solutions that are unexpected by the implementer of the evolutionary system. The more knowledge that is added to the evolutionary algorithm in the form of heuristics restricting the search of a design space, the less likely the evolutionary system will discover unexpected solutions.

For problems that require the application of tacit knowledge or aesthetic judgments, a designer may chose to interact with the evolutionary process by providing subjective evaluations instead of a mathematical function (Sims, 1991). This can be accomplished by having the design enter a fitness value on a numerical scale that is translated into a fitness measure by scaling the value to a standard range and optionally combining it with other fitness measures (Witbrock & Reilly, 1999).

Alternatively, a designer can interact more directly with an evolutionary process by selecting which individuals in a population will survive to reproduce and form the next generation. Examples of this are the biomorphs of Dawkins (1986) and the selection of art alternatives by Todd and Latham (1992). Rosenman (1996, 1997) has the designer selecting possible candidates for solutions to components to be used in the next level assembly. Although interactive approaches allow subjective judgments, they slow the process and should be replaced by computational evaluation, if possible.

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press

A more direct approach to incorporate knowledge into an evolutionary design system is to use a traditional knowledge-intensive approach to guide the progress of an evolutionary algorithm. In this hybrid model, an expert system, case-based reasoning system, or some other form of knowledge-rich, domain-specific design system would assist in the selection and adjustment of various aspects of the evolutionary design system in response to the particulars of a design problem.

The problem with adding significant amounts of domain knowledge to evolutionary algorithms is that they begin to lose their advantages as knowledge-lean models for non-routine design. Although each piece of domain-specific knowledge that is added has the potential to produce a more powerful specialized problem solver, it also limits the possible forms that the evolutionary design system can generate. The evolution of unexpected solutions to design problems has been one of the hallmarks of evolutionary design to the extent that some observers have proposed that evolutionary design systems appear to be creative (Bentley, 1999; Goldberg, 1999). Adding excessive domain knowledge can strip the evolutionary algorithm of this ability, and the model returns to one of a knowledge-intensive optimization process that is more akin to routine designing.

### 3.2. Knowledge-efficient evolution

A second approach to improving the handling of design knowledge in evolutionary algorithms is to improve the efficiency of their use of existing knowledge. The advantage of this approach is that is allows the evolutionary algorithm to remain knowledge lean, thereby maintaining the greatest scope for evolving innovative solutions. Knowledge-lean evolutionary algorithms do not rule out the possibility of using knowledge-rich techniques at a later date. Increasing the efficiency with which an evolutionary algorithm uses domain knowledge generally involves some sort of adaptation or learning (i.e., the evolutionary algorithm must adapt the domain knowledge it is given to the specifics of the evolutionary run).

Genetic engineering (Schnier & Gero, 1996) is a good example of adapting representations during evolutionary search. Genetic engineering is founded upon the schema analysis of Holland (1975) and the building block hypothesis of Goldberg (1989). Genetic engineering uses statistical analyses of the genotypes evolved to find the most useful "building blocks" that emerge during an evolutionary run. The genetic engineering system finds the building blocks that appear most frequently in fit designs, extracts them from the genotypes, and encapsulates them as evolved genes. These evolved genes are then used to replace existing sequences of genes for the construction of new genotypes. This process of extracting useful building blocks and encapsulating them as evolved genes safeguards the knowledge represented by the schema from future disruption. The genetic engineering process allows the evolutionary system to explicitly code knowledge about useful design forms that is adapted to the specifics of the design problem.

### 3.3. Coevolution

Coevolution is the evolution of individuals within the context of other individuals either in the same population or in another population. Coevolution is pervasive in the natural world, with species being dependent on many other species for survival. A commonly cited case of coevolution is the "arms race" observed between species of predator and prey. Coevolution promises to solve the problem of providing context-sensitive information without the need to add complex context-sensitive fitness functions. The next two subsections give detailed descriptions of two coevolutionary approaches.

Maher and Poon (1996) proposed coevolutionary design models for nonroutine designing that use coevolution to evolve problems and solutions in tandem, permitting the exploration of a wide range of possible forms to a class of problems. They argue that designing is an iterative process of searching the design problem space, as well as the design solution space. The coevolutionary model of design searches both the problem space and the solution (design) space in tandem. The most important aspects of the coevolutionary model of design are that solutions are evaluated in the context of the evolved problems and problems are evaluated in the context of the evolved solutions. As a consequence, design proceeds as a compromise between solving the criteria for the design problem and altering the problem to suit the solutions at hand. In this way the coevolutionary process transforms an ill-defined problem that lacks the knowledge necessary to determine the relative importance of the criteria into a well-defined problem that can be solved using the resources available. The problem and solution search processes cooperate to solve an ill-defined problem in an efficient manner.

Potter and De Jong (1994) have proposed a quite different type of cooperative coevolution to evolve solutions by splitting up the evolutionary search of a single complex solution space into separate evolutionary processes that each search simpler solution spaces and cooperate to produce a complete solution.

In the cooperative coevolution genetic algorithm (CCGA), a problem is decomposed into a set of simpler problems, each of which is tackled by a separate evolutionary process. Information is implicitly communicated between evolutionary processes through the use of a shared fitness function. The shared fitness function provides a measure of how well each component works within the context of the other components evolved to solve the complete problem. This is accomplished by selecting representatives from each population and combining them into a single composite structure that can be evaluated by the single fitness function against the top level goal. Credit from evaluating the composite structure flows back to the individual subcompo-

nents to reflect how well they collaborate with the other subcomponents to achieve the top level goal. Each individual solution is therefore evaluated within the context of the representative best solutions at the current time. As a consequence, the best solutions have a great deal of influence on the evolution of other components.

Like the coevolutionary design algorithm, the CCGA uses representatives, or *collaborators*, from each population to cooperate in solving the design problem. However, in this model, the representatives collaborate to form a complete solution rather than forming a problem–solution pairing. Selecting representatives in either approach can be done in many ways, but it is most easily done by selecting the member with the highest fitness value from each collaborating population.

Potter and De Jong (1994) have demonstrated their approach to evolutionary design in a variety of domains and have shown it to be effective. However, the problems tackled thus far with the CCGA have not been decompositions in the same sense that a designer might decompose a design problem in terms of subcomponents of different kinds that have their own specific requirements; instead, they are in terms of subsets of a set of similar elements that have the same requirements as the composite structure. The composite structure is an aggregate of like atomic elements, not "functional" components with differing requirements.

Chen and Brown (2002) have proposed a two-layered system in which components are generated and evaluated according to their success in making up a successful configuration. This work is demonstrated in the configuration of pipes on a Cartesian plane. It is limited to two levels.

## 4. HIERARCHICAL EVOLUTION

Specifying a fitness function that captures the important aspects of a set of design requirements for a complex design in a single quantitative value is a nontrivial problem. The problems become even more complex as multiple criteria are introduced, not only in the specification of fitness functions but also in the unintended ways that the encoded requirements can interact. For example, the work of Jo and Gero (1995) using a single fitness function in the generation of house plans showed that conflicting requirements did not allow the process to converge on satisfactory solutions. Rosenman (1996) presented a hierarchical evolutionary model for nonroutine design. This model was based on recursively decomposing a complex artifact into subproblems of designing components with lower levels of complexity until a level is reached at which the design problem becomes a routine one of generating a simple component.

This approach can significantly ease the burden of specifying data structures and fitness functions for evolutionary algorithms because these structures and functions can be made local to the component in question. There are two potential advantages of decomposing a complex design problem into a hierarchy of simpler problems. The first is that the decomposed problems are simpler and hence are more likely to be easily solved. The search of multiple design spaces that represent simple problems is more efficient than searching a single design space that represents a complex problem. As the number of design variables increases the number of design spaces increases linearly, but each design space remains small. In other words, the size of the genotypes used by each evolutionary algorithm will never become larger than is strictly necessary to solve the functional requirements of the subcomponent. As discussed above, in traditional evolutionary algorithms the size of the design space will necessarily grow exponentially as the number of design variables increases. This approach is comparable to the advantage gained by stage–state approaches such as dynamic programming (Bellman, 1957).

The second advantage of specifying a set of simpler problems is that it becomes easier for the implementer of an evolutionary design system to specify the data structures and fitness functions, as they can be kept as simple as possible for each component problem. Evolutionary problems solving different aspects of the design problem can use different data structures. This can result in a great simplification of the specification task.

A complex artifact design problem is first broken down into a hierarchy of increasingly simpler subcomponent design problems. Each component within a hierarchy has an associated evolutionary design process that must produce solutions according to its own set of evaluation criteria. The advantages of the hierarchical approach are that only those factors relevant to the design of a component are considered, and factors relevant to the relationships among components are treated at their assembly level.

A problem with this bottom-up hierarchical evolutionary approach is that it does not have any mechanism for the coordination of evolutionary processes through the sharing of design knowledge provided in advance or discovered during the design process. All of the subcomponents are evolved before they are assembled into higher level components. This requirement places a great deal of importance on defining the correct fitness functions for subcomponents in advance, because there is no opportunity to review the evaluation criteria other than to start a completely new evolutionary search. There is an assumption that by evolving a sufficient number of suitable alternative solutions for each component, a suitable solution will be available at the next level. This hierarchical evolutionary model of designing is therefore not well suited to tackling design problems that have ill-defined requirements that must be propagated throughout the component hierarchy.

Early implementations of hierarchical evolution (Rosenman, 1996) have included an interactive component to allow the designer to specify which components are used in the assembly of higher level components. This allows the designer the opportunity to exercise control over the evolutionary process while at the same time addressing the problem of coordinating evolutionary processes. The type

of knowledge provided by the designer as selections of preferred components is particularly difficult to encode in fitness functions in advance of an evolutionary run, because it is not local to a single component design problem. In addition, this type of knowledge tends to be highly context sensitive, which means that knowledge encoded in fitness functions must be applied in a manner that depends on the context of the component provided by other components that may potentially include all of the components in the hierarchy. Conventional approaches to the provision of domain knowledge would require the augmentation of fitness functions with potentially complex context-sensitive decision mechanisms to determine whether a particular piece of knowledge is applicable.

## 5. HIERARCHICAL COEVOLUTION

This section introduces an evolutionary algorithm, which attempts to be knowledge lean and relatively simple, to apply to complex problems. It presents a synthesis of the evolutionary approaches to nonroutine designing as described above. Hierarchical coevolution combines the pragmatic decomposition of design problems and the knowledge clarity of *hierarchical evolution* together with the efficient use of design knowledge demonstrated by cooperative coevolution and the coevolutionary model of design. Hierarchical coevolution aims to provide the necessary interrelationship knowledge between the various components and levels of assembly to produce a self-regulatory evolutionary system.

### 5.1. Intrinsic and extrinsic fitness

Hierarchical evolution and cooperative coevolution share the concept of problem decomposition, although they differ to the degree to which the decomposition is applied. However, they have quite different means of defining evaluation criteria: hierarchical evolution evaluates components according to criteria defined at the component level, whereas cooperative coevolution evaluates components only at the superior problem level. The coevolutionary model of design developed by Maher and Poon (1996) is closer to hierarchical evolution in this regard. Even though it appears to evaluate both problem and solution at the same time, evolutionary searches are treated as independent "components" of the design situation and are evolved using quite separate fitness functions.

Hierarchical coevolution combines these two means of evaluating individuals by defining two types of fitness: *intrinsic* and *extrinsic*. Intrinsic fitness, as used by the evaluation criteria for hierarchical evolution, measures the success of a component to satisfy the criteria unique to its function. Evaluation criteria that define intrinsic fitness deal only with those aspects of the problem that affect a component directly. Extrinsic fitness, similar to the fitness evaluation in cooperative coevolution, measures the ability of components to cooperate with other components at the same level within the *design situation* of the immediately superior problem. Extrinsic fitness is the fitness of an individual with respect to its immediately superior assembly.

Representatives from the lower level component solution population are used to evolve new higher level problems by contributing to the intrinsic fitness evaluation of the higher level component solutions. In this way, the coevolutionary model of design exploration is mirrored in hierarchical coevolution between adjacent levels of the component hierarchy. Like the coevolutionary model of design exploration, hierarchical coevolution treats immediately superior solutions in the hierarchy of components as a population of potential problem definitions that define the design problem for lower level solution populations. The hierarchical coevolutionary model uses representatives from the problem population to define the extrinsic fitness function of lower level component by evaluating them within the context of the representative problems. At each level of the hierarchy the cooperative coevolution of components is guided by the evaluations of the assembly of the components evolved at the immediately superior level. The coevolution of components with a common fitness function at a higher level allows the communication of context-sensitive knowledge about the relationships between components. Knowledge is transferred implicitly from one evolutionary process to another through the shared evaluation of the two components at the assembly level. Intrinsic evaluations cover aspects of the design problem that are limited to the scope of a component and the subcomponents that make it up. Extrinsic evaluations cover aspects of the design problem that relate to its ability to function within the context of an assembled superior level component. As a consequence, intrinsic and extrinsic evaluations are related, and in general, the intrinsic evaluation of a component at one level is the extrinsic evaluation of its subcomponents.

The separation of intrinsic and extrinsic evaluations allows the context of a component to be limited to avoid the problems of having to evaluate a component within the context of all other components. The use of higher level intrinsic evaluations as lower level extrinsic evaluations over two levels of a hierarchy allows some information to be shared vertically without a need to reevaluate all of the other components that go to make up the tree. The determination of an overall fitness for an individual from the intrinsic and extrinsic fitness measures can be done in many ways. However, a simple means of combining intrinsic and extrinsic fitness functions is used in which the fitness evaluations are combined using a weighted sum to produce a fitness value that reflects the relative importance given to the intrinsic and extrinsic evaluation criteria. In this work, different weights for the intrinsic fitness and the extrinsic fitness were used.

During the initialization phase, only the intrinsic fitness is used to define the fitness of individuals. This is used as a bootstrapping process to provide some initial components for the coevolutionary processes to use.

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press

Comparing hierarchical evolution and hierarchical coevolution, we can observe some important differences between the two approaches. Most important, the hierarchical coevolutionary model does not block the evolution of new solutions at any level of the component hierarchy until the lower levels have been completed. In the previous hierarchical evolution approach (Rosenman, 1996), each component population at the lower level was fully evolved over a number of generations, and potentially suitable components were selected before the next level of components (assemblies) was, in turn, evolved. In hierarchical coevolution, for each generation, components are evolved and used to generate the next level of component assemblies. The former approach was a breadthwise bottom-up approach, whereas the hierarchical coevolution approach is a depthwise bottom-up and then top-down approach.

## 6. IMPLEMENTATION OF HIERARCHICAL COEVOLUTION

To illustrate the behavior of the hierarchical coevolutionary an implementation has been tested against a set of simple layout problems. Although this problem uses domain-specific data structures and fitness functions, the focus of this work is the hierarchical nature of the problem. This continues the experiments done by Rosenman (1996) evolving house layouts using hierarchical evolutionary systems. In Rosenman's experiments, a user guided the evolutionary process by selecting components interactively, allowing the evolutionary system to be applied to complex component design problems. In particular, Rosenman used a cellular growth model to combine atomic units to form rooms and used the same model to assemble rooms together into larger components. The representations involved allow great flexibility in the shape of the rooms evolved and in close packing of nonrectangular rooms to form assemblies.

Much simpler representations and construction processes have been used in these experiments to facilitate hierarchical coevolution without the need for human guidance. The experiments described below use rectangular-shaped rooms and constrain higher level assemblies to be joined along the edges of their bounding boxes. Although the domain problems are in house planning, the process is applicable generally to all complex problems capable of being decomposed hierarchically.

### 6.1. Facility layout problems

Layout problems of the type described below are also known as facility layout problems and have been extensively studied in design computing, artificial intelligence, and computer science. The facility layout problem is applicable to several fields, including the layout of space in a building, the layout of components on printed circuit boards, and the packaging of goods in containers. Computer systems capable of solving layout problems have been developed using both knowledge-rich and knowledge-lean approaches. Different evolutionary algorithms (e.g., GA, genetic programming, evolutionary strategy, etc.) have been applied to layout problems using different representation schemes and have been shown to be good general-purpose methods when matched with appropriate representation schemes.

In general, the facility layout problem deals with a set of $n$ rectangular facilities $F = \{ f_1, f_2, \ldots , f_n\}$ that have to be located within a planar site. A connectivity matrix $M = [m_{ij}]i, j = 1, \ldots , n$ defines the connection weights between each pair of facilities $f_i$ and $f_j$. The objective is to find a nonoverlapping arrangement of the facilities with minimal connectivity costs, calculated as $\sum_{i,j=1}^{n} d_{ij}\, m_{ij}$, where $d_{ij}$ is the distance between facilities $f_i$ and $f_j$. Variants of the facility layout problem have been introduced. For example, facility layout problems have been devised for irregular sites, sites with preoccupied regions, or multiple sites (Kuziak & Heragu, 1987; Meller & Gau, 1996).

### 6.2. The house layout problem

The house floor plan design problem set for the evolutionary algorithms below differs from the typical facility layout problems in several ways. The first is that the problem involves laying out a set of nonidentical, flexible facilities, in this case rooms. The size of the rooms are not predetermined; instead, constraints are supplied as ranges of valid areas and desired length to width ratios. The evolutionary algorithm must determine the most appropriate sizes and shapes of the rooms. The problem then naturally decomposes into a two-level hierarchy of subproblems (i.e., a standard facility layout problem to assemble rooms and a set of subproblems to determine size and shape of each room). In other words, this is both a topology and geometry problem.

The second difference in the layout problem is that the rooms can be grouped into zones in advance of an evolutionary run, such that each zone contains rooms that share related functions. The introduction of zones allows a designer some additional control over the placement of related rooms, beyond the specification of connection weights. The addition of a zone level also allows complex facility layout problems to be broken down into a set of simpler facility layout problems for the sake of efficiency. It provides a way for designers to add knowledge about the design process in a natural way.

As a consequence, the layout problem breaks down into a hierarchy of at least three levels. First, at the room level, an appropriate size and shape of each room must be determined. Second, at the zone level, related rooms must be arranged to minimize the connectivity costs of rooms within the zone. Third, at the house level, zones must be arranged to minimize the connectivity costs of rooms in different zones. The requirements of a house layout problem can be represented with the diagram shown in Figure 1. The diagram clearly illustrates the connectivity relationships between rooms and how rooms that have many connectivity require-

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press

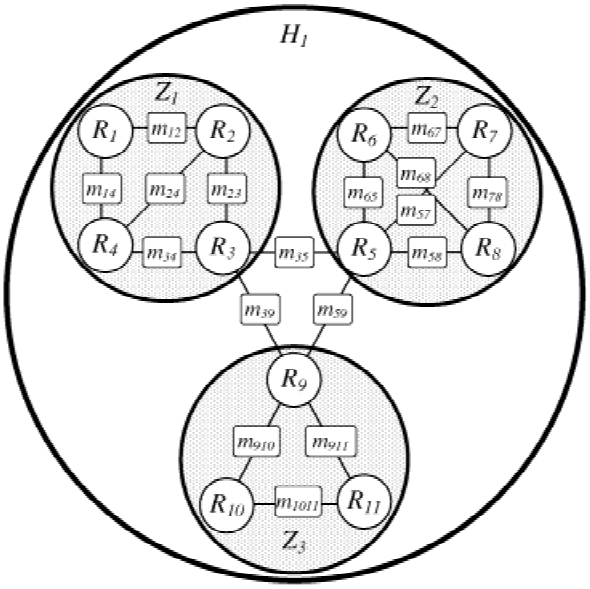



**Fig. 1.** A diagrammatic representation of the relationships between rooms, zones, and the house with three zones.

ments are grouped together into zones. Although for this implementation example the decomposition structure is given, in a full design situation, different evolutionary runs could be implemented with different decomposition structures.

The subproblems at each level of the problem hierarchy have different objectives, and the fitness functions used to evolve possible solutions reflect this. The details of the evolutionary processes, representations, and evaluation functions used to evolve rooms, zones, and houses are given in the next subsection. At each level, the weighted function for combining intrinsic and extrinsic fitnesses used in the examples was a 2:1 ratio. That is, the intrinsic fitness was weighted twice as much as the extrinsic fitness.

## 6.3. Hierarchical coevolution of house layouts

### *6.3.1. Room evolution*

Rooms are represented using integer genotypes with two genes. The first gene is used to encode the width of the room, and the second gene is used to encode the length. The actual length and width of a room are calculated by multiplying the integer value of each gene by 0.1 m. Expressed genotypes are represented as rectangular shapes that can be rotated and positioned within higher level assemblies as required.

Rooms are evaluated using an intrinsic fitness function that compares each room's area and length to width ratio with predefined ranges of desired areas and aspect ratios. The fitness functions for a room are defined by specifying minimum and maximum values for area and width to length ratio, $\text{Area}_{\text{min}}$, $\text{Area}_{\text{max}}$, $\text{Ratio}_{\text{min}}$, and $\text{Ratio}_{\text{max}}$. The evaluation functions take a common approach in evolutionary computing by turning the hard constraints that would disallow rooms that fall outside the minimum and maximum values into soft constraints that allow rooms to have areas and ratios outside these ranges but with increasing penalties, incurred as loss of fitness. The shape of the fitness function used for evaluating the area of a room is illustrated in Figure 2; a similar function is used for evaluating the shape of a room.

Some of the evaluation functions for rooms in the following examples have been made deliberately vague by specifying wide ranges of acceptable areas and ratios. As a result, the evaluation functions define a large plateau in the intrinsic fitness landscape for rooms that are equally "good" from the perspective of a room's requirements. Vague descriptions of requirements are common in design when the acceptability of a design relies on contingent factors. The objective of using them in the following experiments has been to show that hierarchical coevolution can help resolve vague requirements by allowing different levels of the hierarchy to interact.

The evaluation of the area and width to length ratio of each room provides the hierarchical coevolutionary algorithm with a measure of the intrinsic value of the room (i.e., the ability of the room to support its assigned function). Each room is also evaluated within the context of the zone to which it belongs, providing a measure of its extrinsic fitness. The details of the zone evaluation functions are given in the next subsection. When a zone is being evaluated, it uses the best rooms currently available. However, when a room is being evaluated in the context of a zone, it replaces the current best room of its type.

### *6.3.2. Zone evolution*

Zones are represented as slicing tree structures (STSs; see Schnecke & Vornberger, 1997). An STS for $n$ rooms is a binary expression tree in which each leaf node is a label indicating a type of room and each internal node represents the spatial relationship between its two child substructures. Using the STS, zones are constructed using a sequence of join operations between pairs of rooms or between pairs of partially constructed zones that place the second substructure directly north, south, east, or west of the first substructure, as illustrated in Figure 3.

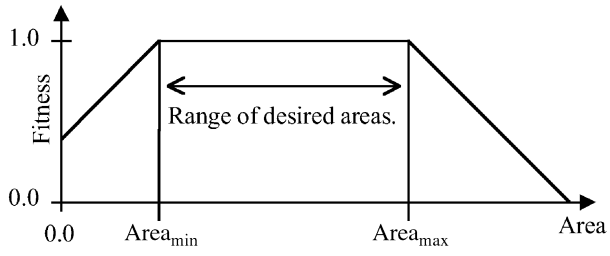



**Fig. 2.** The shape of the fitness function for the evaluation of the area of a room.

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press

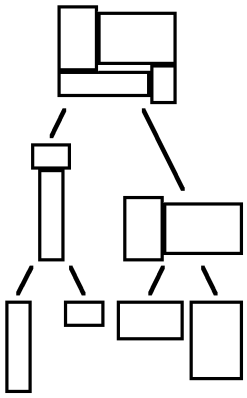


**Fig. 3.** The bottom-up construction of a zone using a slicing tree structure (STS).

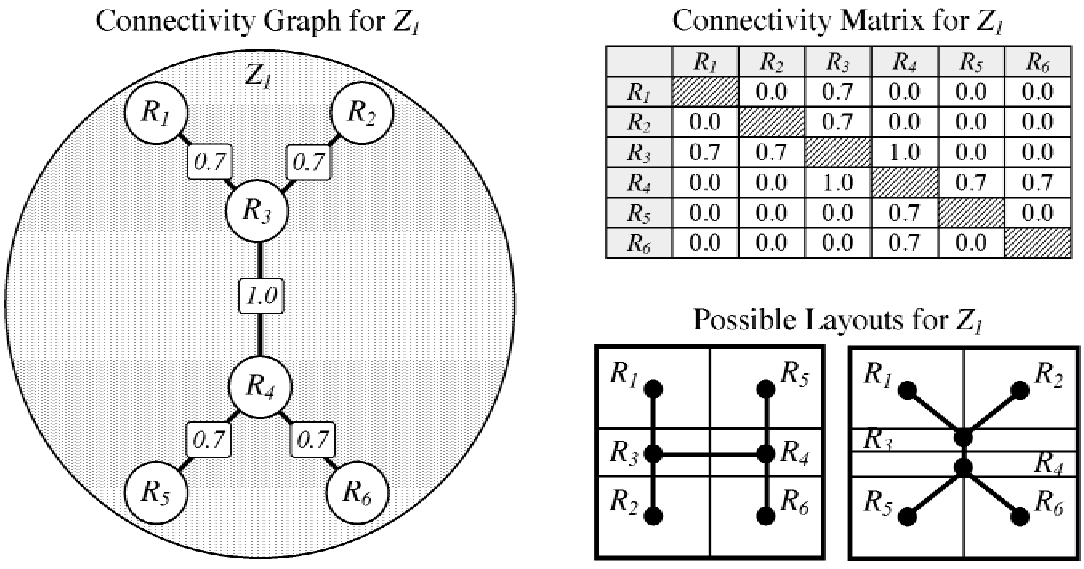
As well as being illustrated as a tree, STSs can also be written as symbolic expressions, using Polish notation. For example, the STS expression (N $R_1R_2$) places room $R_2$ directly north of room $R_1$, and the STS expression (N (W $R_1R_2$)$R_3$) places room $R_2$ west of room $R_1$ and room $R_3$ north of the substructure ($R_1 \cdot R_2$) that contains rooms $R_1$ and $R_2$. Each join operation also specifies whether either of the substructures should be rotated and if so by how much. Rotations are specified clockwise in 90° increments. The result of joining two rooms is a partially constructed zone. Zones are combined in much the same way as rooms. When zones are joined, they may be rotated. The Polish notation represents the genotype for zones and houses. The bounding box of each zone is used to calculate the relative positions of the zones such that the bounding boxes of the rotated zones touch.

The intrinsic fitness of each zone is evaluated in terms of the connectivity between the rooms it contains. The fitness measures for a zone are assessed by substituting the labels in the STS with the current best rooms available from the lower level. To evaluate the connectivity between rooms, a matrix of connection strengths between rooms is used. A simple connectivity matrix together with the graph of connections that it defines between rooms is shown in Figure 4. Each zone is also assigned an extrinsic fitness based on how well the zone works with other zones in the current best house. The zone being evaluated is constructed with the current best rooms. It is then used in place of the best zone of its type in the evaluation of the best house. This method of evaluating the extrinsic fitness of a zone works in much the same way that rooms are evaluated in the context of zones.

### *6.3.3. House evolution*

Houses are represented in similar way to zones (i.e., using a STS genotype), but they contain zone identifiers. Houses are constructed in a very similar way to zones. Houses are composed by joining and rotating zones, or partially constructed houses, according to the operators at the internal nodes of the tree. Figure 5 illustrates the construction process for a house consisting of three zones and shows how a complex house can be built with only two join operations.

Each house is evaluated by assessing the connectivity between rooms in different zones and the compactness of the house. In addition, each house can also be evaluated in terms of its fit to a predetermined site. The connectivity of rooms in different zones is used to evaluate the connectivity of the house. Only the interzone connections between rooms in different zones are considered because connections between rooms in the same zone are evaluated by the




| | $R_1$ | $R_2$ | $R_3$ | $R_4$ | $R_5$ | $R_6$ |
|---|---|---|---|---|---|---|
| $R_1$ | | 0.0 | 0.7 | 0.0 | 0.0 | 0.0 |
| $R_2$ | 0.0 | | 0.7 | 0.0 | 0.0 | 0.0 |
| $R_3$ | 0.7 | 0.7 | | 1.0 | 0.0 | 0.0 |
| $R_4$ | 0.0 | 0.0 | 1.0 | | 0.7 | 0.7 |
| $R_5$ | 0.0 | 0.0 | 0.0 | 0.7 | | 0.0 |
| $R_6$ | 0.0 | 0.0 | 0.0 | 0.7 | 0.0 | |

**Fig. 4.** A connectivity matrix for a zone $Z_1$ and two possible solutions for the room layout.

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press

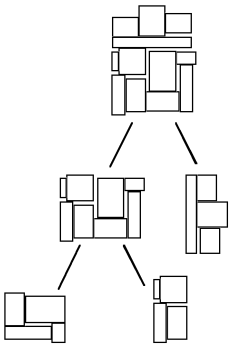


**Fig. 5.** The bottom-up construction of a house using a slicing tree structure (STS) to represent the zones to join at leaves and the join operations at internal nodes.

appropriate zone. By assessing the connectivity of two rooms in different zones, a pressure to rotate one of the zones may be exerted if the zones are adjacent but the rooms are separated. For example, although we could define that a utility zone should be connected to a living zone, it does not say how these two zones should be connected; instead, by defining that the laundry should be connected to the kitchen, we can provide additional domain knowledge. Commonly, interzone connections are defined between hallways in different rooms thereby specifying their function as a means of getting between various parts of a house.

The compactness of a house is evaluated by penalties for not using all of the space within its bounding box. In addition to the compactness measure, a house can be evaluated within the context of a site. The site evaluation function penalizes a house for extending beyond the bounds of a site or for underutilizing the available area in the site. Figure 6 illustrates the calculation of site evaluation.

### 6.4. Example 1: Evolving a hallway

The aim of this experiment is to show that the desired topology of rooms, specified in the connectivity matrix for a zone, can have a significant effect on the development of rooms. This simple experiment is again limited to the coevolution of rooms within a single zone. There are five rooms in this example, four of which have been constrained to evolve toward squarelike shapes, and the fifth of which has been constrained only in its size. Figure 7 shows the connectivity graph and the area and aspect ratios for this example.

The connectivity graph shows that four of the rooms, $R_1$, $R_2$, $R_4$, and $R_5$ are connected to the fifth room $R_3$. The fifth room is thereby defined to act as a hallway connecting all of the other rooms. The zone must arrange the rooms $R_1$, $R_2$, $R_4$, and $R_5$ around $R_3$ to minimize connectivity costs. Figure 8 illustrates three stages of the evolution of a zone, including the genotype representation of the zones.

The first STS shown in Figure 8 shows an early solution to the problem of connecting all of the other rooms to $R_5$ after all of the rooms have adapted to the arrangement evolved by the zone. This solution is good, but it is suboptimal, which allows the continued search for better zone layouts. The second STS in Figure 8 shows a later stage in the evolutionary process, when the zone has rearranged the rooms into a zone with a better connectivity evaluation at the expense of compactness.

The room evaluations for this stage are also lower in general than for the earlier stage because the rooms are adapting to the requirements of the new zone arrangement. One of the most interesting observations to make about the evolutionary process illustrated in Figure 8 is the way that $R_3$ elongates from the second to the third STS to fill all of the space left between the two pairs of rooms: $R_1$ and $R_2$ and $R_4$ and $R_5$. The elongation of $R_5$ is entirely caused by the extrinsic evaluations of the room at the zone level in the context of its function as a hallway.

This example shows the effectiveness of the coevolutionary process at influencing the evolution of rooms as the extrinsic zone evaluations change radically. We can also observe a shift in the style of the evolutionary design process in early and late stages of the design process. The style of the design process shifts from the evolution of approximate room shapes in the early stages of design to a more detailed design process once a good zone has been found. In the early stages of the evolutionary process, when many different room arrangements compete in the zone evolution-

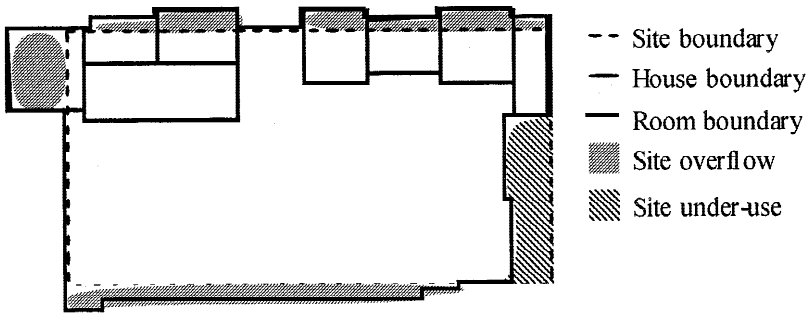
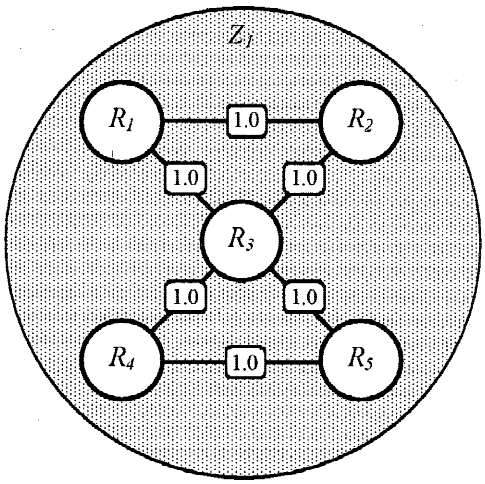



**Fig. 6.** An illustration of the evaluation of a house in the context of a site.

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press




| | $A_{min}$ | $A_{max}$ | $R_{min}$ | $R_{max}$ |
|---|---|---|---|---|
| $R_1$ | 8 | 16 | 0.5 | 2.0 |
| $R_2$ | 8 | 16 | 0.5 | 2.0 |
| $R_3$ | 8 | 16 | 0.5 | 2.0 |
| $R_4$ | 8 | 16 | 0.5 | 2.0 |
| $R_5$ | 8 | 16 | 0.1 | 10.0 |

**Fig. 7.** The connectivity graph for the zone and the room evaluation parameters used to evolve a hallway $R_3$ that connects rooms $R_1$, $R_2$, $R_4$, and $R_5$.

ary process, the room shapes have to alter radically from one generation to the next. As the zone-level evolutionary process settles on a good arrangement of rooms, it provides a more stable environment for the room evolutionary processes to evaluate new rooms. The style of the evolutionary processes shifts from the development of approximate room shapes to a process of fitting rooms into the stable zone. This example shows the coevolutionary process of reformulating intrinsic fitness in a two-level hierarchy. In the next example, it will be shown how this process can be extended across multiple levels.

### 6.5. Example 2: Evolving a small house

This example shows how a hierarchy of coevolving processes exchanges information that reduces the complexity of specifying the design problem a priori. In this experiment the hierarchy was extended to all three levels to demonstrate that evolutionary pressure (i.e., information) can flow from one level to another, even when those levels are not adjacent and the pressures must be transmitted through an intermediate level. In this example, a small house layout is evolved. The small house (SH) contains two zones: a living zone (LZ) and a sleeping zone (SZ). The living zone contains a living room (LR), a dining room (DR), a kitchen (Ki), and an entrance (En). The sleeping zone contains two small bedrooms (B1 and B2), a master bedroom (MB), and a bathroom (Ba). The constraints placed on each room and the connectivity relationships between them are illustrated in Figure 9.

The zones are evaluated according to the connectivity requirement on their respective rooms and do not have a

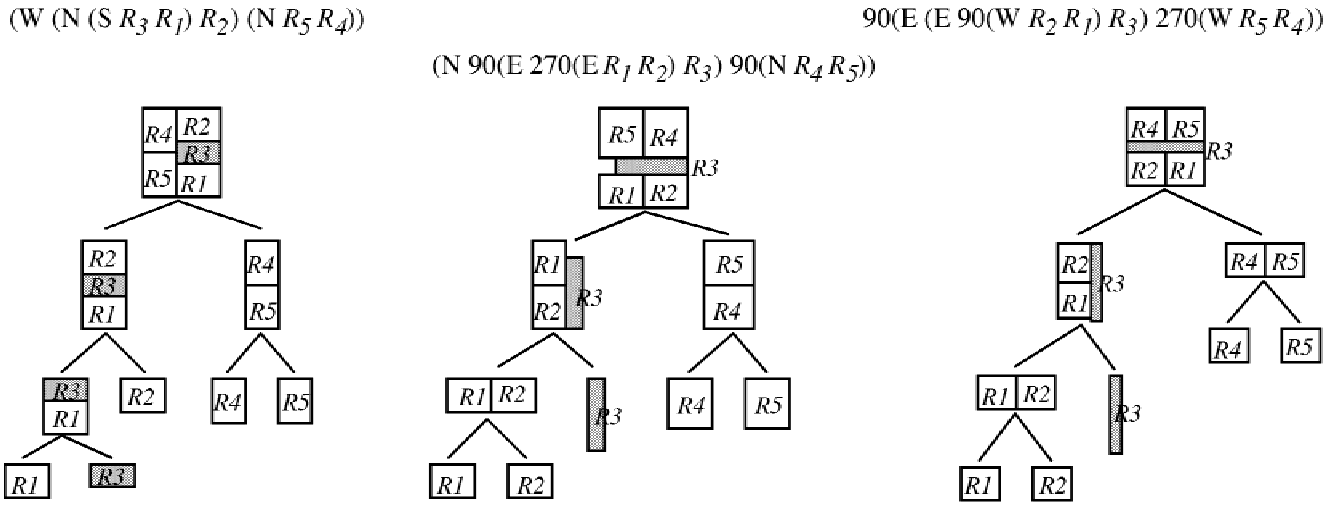
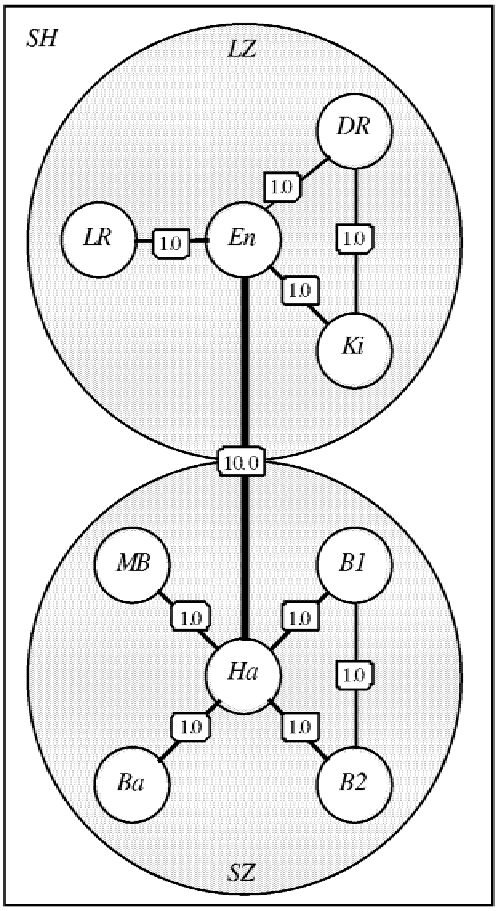



**Fig. 8.** Three stages in the evolution of a zone in hierarchical evolution, showing the evolution of a hallway. (Rotations of 0° have been omitted from the descriptions of the tree structures for brevity.)

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press




| | $A_{min}$ | $A_{max}$ | $R_{min}$ | $R_{max}$ |
|---|---|---|---|---|
| *LR* | 12 | 24 | 0.25 | 4.0 |
| *DR* | 9 | 20 | 0.25 | 4.0 |
| *Ki* | 8 | 16 | 0.25 | 4.0 |
| *En* | 4 | 20 | 0.05 | 20.0 |
| *B1* | 8 | 16 | 0.50 | 2.0 |
| *B2* | 8 | 16 | 0.50 | 2.0 |
| *MB* | 12 | 20 | 0.25 | 4.0 |
| *Ba* | 8 | 16 | 0.50 | 2.0 |
| *Ha* | 4 | 20 | 0.05 | 20.0 |

**Fig. 9.** The design problem specification for a small house.

compactness evaluation function. Without this criterion, the zones are free to evolve into a wide range of possible configurations.

At the house level, the entrance and the hallway must connect. This requirement forms part of the extrinsic fitness measure for the zones. Therefore, between the hallway in the sleeping zone and the entrance in the living zone, the weight of the connection is 10.0 because the connectivity between these two spaces is essential. The house is also evaluated on the compactness of the overall shape. The compactness of house designs is the result of an evaluation function defined at that level rather than of the careful manipulation of compactness evaluation functions throughout the hierarchy. The entrance has an additional requirement over previously presented hallways that it must touch the boundary of the house. This requirement is implemented as an evaluation function of evolved houses. The addition of the requirements that En touches one of the outside edges of the house and connects with Ha in the sleeping zone means that En must touch two outside edges of LZ.

Figure 10 shows the results of the best run out of 15 runs of the hierarchical coevolution on the above problem. Each run was implemented for 300 generations. Five of the 15 runs generated satisfactory designs: 6 were reasonably satisfactory, and the other 4 solutions were not satisfactory.

Figure 10a–f shows the best house for each of the generations shown. At generation 10, the room sizes and room connectivities have been satisfied, but the house compactness is still poor, as there is some amount of "white space" in the bounding box. In addition, the entrance and hallway are not connected. Note, however, that the pressure at the house level for the entrance to have external access has resulted in a change of shape of the entrance. At generation 50, the compactness is increased; at generation 200 the house is fully compacted but now the entrance and hallway are out of line; at generation 250 all the room size constraints are satisfied, the zone room connectivities are maximized, the house is fully compacted, the entrance and hallway connectivity is maximized, and the entrance access to the exterior is satisfied.

## 7. SUMMARY

This article has presented an approach to the nonroutine design of complex objects based on hierarchical coevolution. It has shown that whereas the local design of lower

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press

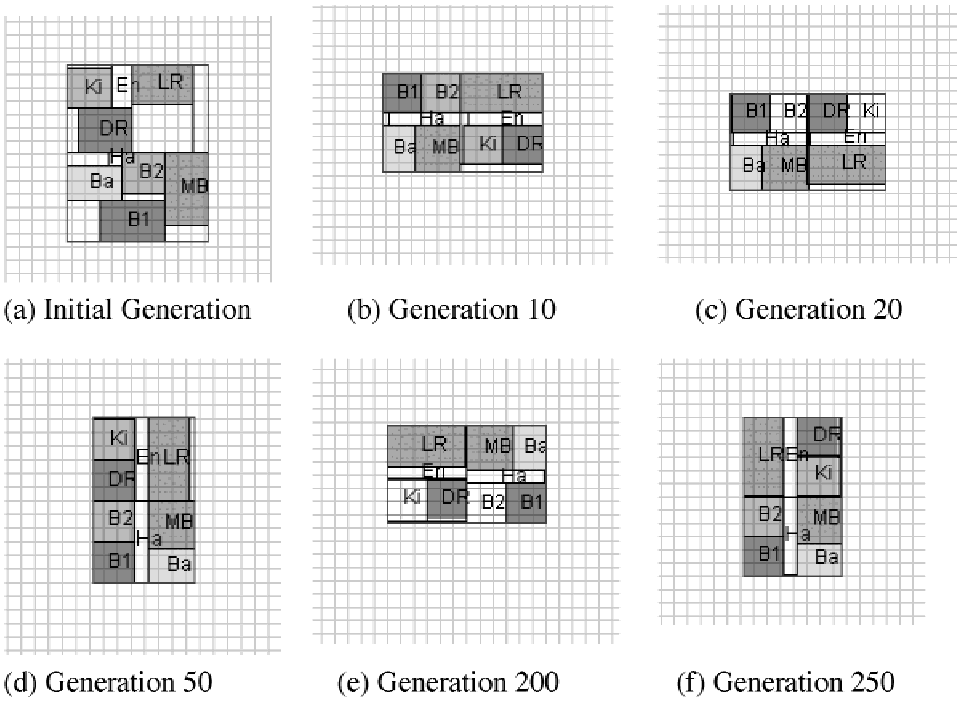



**Fig. 10.** The results of the hierarchical coevolution of a small house.

level components can simplify the design of a complex object, the self-regulating mechanism can provide the necessary transfer of information necessary across the various components and levels of the hierarchy. Whereas the implementation shown is in the domain of space layout problems, the focus is on the hierarchical nature of the problem and the decomposition and integration issues. The coevolution approach reformulates the given or intrinsic fitnesses by formulating extrinsic fitnesses that take into account a component's contribution to the assembly that uses it. By working with two levels at a time, it is possible to propagate requirements at a given level to all the lower levels. The automatic reformulation of the fitness function based initially on intrinsic fitnesses to take into account the extrinsic fitnesses provides for the necessary relationships between components and the assemblies at different levels. This has been demonstrated in the house example by the pressure exerted by the house requirements on the zones and rooms. In the example, knowledge emerges that shapes the hallway and the surrounding rooms so that not only are they arranged satisfactorily according to the zone fitness but they are sized to meet the requirements coming from the house level. The generation of new knowledge that is appropriate to the problem at hand is fundamental to creative design. The knowledge generated (i.e., regarding the interrelationships between the components at different levels) is not present in any of the given evaluation functions but emerges as a result of the evolutionary process for that situation. Because this knowledge is specific to the particular situation, it would be impossible to specify it a priori. It is this ability of the hierarchical evolutionary process that demonstrates its suitability for nonroutine design. Compared with the previous work in hierarchical evolution, no interaction was present, eliminating the need for including run-time implicit knowledge.

Although the solution achieved may not seem innovative (to a human viewer) in the wider context of house design, it can be considered nonroutine to the extent that the designing agent, the program, has generated this design from basic components without any explicit knowledge of the forms and relationships required to satisfy the requirements.

At present, the algorithm is based on a predetermined weighting for intrinsic and extrinsic fitnesses. The weighting ratios have a large influence on the results of the process. Too large a weight on the extrinsic fitness and the intrinsic fitness has little, if any, influence. Too large a weight on the intrinsic fitness and the important context criteria can be ignored. Current work is aimed at automating the influence of the extrinsic fitness on the overall fitness function.

## ACKNOWLEDGMENTS

This work is supported by Australian Research Council Large Grant A8970053.

## REFERENCES

Bellman, R. (1957). *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Bentley, P. (Ed.) (1999). *Evolutionary Design by Computers*. San Francisco, CA: Morgan Kaufmann.

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press

Chen, Z.F., & Brown, D.C. (2002). Explorations of a two-layered A-Design system. *Int. Workshop Agents in Design: WAID'02*. Cambridge, MA: MIT.
Cohon, J.L. (1978). *Multiobjective Programming and Planning*. New York: Academic.
Dawkins, R. (1986). *The Blind Watchmaker*. Harlow, UK: Longman Scientific and Technical.
Fonseca, C.M., & Fleming, P.J. (1995). An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation, 3(1)*, 1–16.
Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA: Addison–Wesley.
Goldberg, D.E. (1999). The race, the hurdle and the sweet spot: lessons from genetic algorithms for the automation of design innovation and creativity. In *Evolutionary Design by Computers* (Bentley, P.J., Ed.). San Francisco, CA: Morgan Kaufmann.
Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.
Jo, J.H., & Gero, J.S. (1995). A genetic approach to space layout planning. *Architectural Science Review, 38(1)*, 37–46.
Kuziak, A., & Heragu, S. (1987). The facility layout problem. *European Journal of Operational Research, 29*, 229–251.
Maher, M.L., & Poon, J. (1996). Modelling design exploration as co-evolution. *Microcomputers in Civil Engineering, 11*, 195–210.
Meller, R.D., & Gau, K.-Y. (1996). The facility layout problem: recent trends and perspectives. *European Journal of Operational Research, 57*, 351–366.
Moscato, P. (1989). *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*. Technical Report No. 790. Pasadena, CA: California Institute of Technology, Caltech Concurrent Computation Program.
Potter, M., & De Jong, K.A. (1994). A cooperative coevolutionary approach to function optimization. In *Lecture Notes in Computer Science*. *Proc. Third Conf. Parallel Problem Solving from Nature 2* (Davidor, Y., Schwefel, H.-P., and Manner, R., Eds.), Vol. 866, pp. 249–257. New York: Springer–Verlag.
Rosenman, M.A. (1996). A growth model for form generation using a hierarchical evolutionary approach. *Microcomputers in Civil Engineering, 11*, 161–172.
Rosenman, M.A. (1997). The generation of form using an evolutionary approach. In *Evolutionary Algorithms in Engineering Applications* (Dasgupta, D., & Michalewicz, Z., Eds.), pp. 69–85. New York: Springer.
Schnecke, V., & Vornberger, O. (1997). Hybrid genetic algorithms for constrained placement problems. *IEEE Transactions on Evolutionary Computation, 1(4)*, 266–277.
Schnier, T., & Gero, J.S. (1996). Learning genetic representations as alternative to hand-coded shape grammars. In *Artificial Intelligence in Design '96* (Gero, J.S., & Sudweeks, F., Eds.), pp. 39–57. Dordrecht: Kluwer.
Sims, K. (1991). Artificial evolution for computer graphics. *Computer Graphics, 25(4)*, 319–328.
Todd, S., & Latham, W. (1992). *Evolutionary Art and Computers*. New York: Academic.
Witbrock, M., & Reilly, S.N. (1999). Evolving genetic art. In *Evolutionary Design by Computers* (Bentley, P.J., Ed.), pp. 251–259. San Francisco, CA: Morgan Kaufmann.

---

**Mike Rosenman** is a Senior Lecturer at the Key Centre of Design Computing and Cognition, School of Architecture, Design Science, and Planning, University of Sydney. He is an architect with an interest in the general processes involved in design. His research, which spans over more than 35 years, has included design optimization, design methods, and artificial intelligence in the design domain. Dr. Rosenman has presented work at a number of international conferences and workshops and is the author of over 80 publications. He has been conducting research into hierarchical evolutionary systems for design since 1995.

**Rob Saunders** completed his PhD at the University of Sydney, where he developed a computational model of curiosity to investigate its role in individual and social creative processes. He currently works as a freelance artificial intelligence consultant, specializing in assisting artists and designers in the development of state of the art computational systems. Before becoming a consultant, Rob studied artificial intelligence at Edinburgh University, where he developed an award-winning project that examined the use of evolutionary computing systems in the exploration of novel design spaces. Dr. Saunders is continuing his long-term interest in the research and development of advanced computations systems to model and support creative processes.

https://doi.org/10.1017/S089006040317401X Published online by Cambridge University Press