

Chapter ?

EVOLVING DESIGNS BY GENERATING USEFUL COMPLEX GENE STRUCTURES

By M. A. Rosenman and J. S. Gero

Introduction

This chapter presents two examples of work for evolving designs by generating useful complex gene structures. where the first example uses a genetic engineering approach whereas the other uses a growth model of form. Both examples have as their motivation to overcome the combinatorial effect of large design spaces by focussing the search in useful areas. This focussing is achieved by starting with design spaces defined by low-level basic genes and creating design spaces defined by increasingly more complex gene structures. In both cases the low-level basic genes represent simple design actions which when executed produce parts of design solutions. Both works are exemplified in the domain of architectural floor plans.

Evolving Complex Design Genes Using a Genetic Engineering Approach

Evolutionary systems are often superior to other search algorithms for problem consisting of large unstructured search spaces, where no heuristic exists to guide the search. However, the search time for evolutionary systems depends greatly on efficient codings. Since the search time grows exponentially with the size of the genotype, one way to reduce the search time is to have the system learn more efficient problem specific coding containing the minimal genotype size while at the same time not excluding potential solutions and producing small numbers of illegal solutions. To achieve this, an evolutionary system which identifies successful combinations of low-level, (basic) genes and combines them into higher-level, (complex) genes is presented. Genes evolve in ever-increasing complexity, thus encoding a higher number of the original basic genes. This results in a continuous restructuring of the search space, allowing potential successful solutions to be found in much shorter search time. This restructuring changes the landscape by changing the probabilities of particular parts of the space being located. It increases the probability of finding those solutions which contain combinations of genes which contribute to good solutions and, as a consequence, reduces the probability of landing on those solutions which contain combinations which do not contribute to good solutions. The evolved coding can then be used to solve other related problems. These concepts are demonstrated by an example from the design of two-dimensional architectural floor plans.

Basic and Complex Genes

Let us take an example where solutions are coded as strings of 19 different integer numbers from the range [0; 18]. Some fitness function is defined over the phenotype where higher values are better. Mutation here is defined as a pair-wise swap of two randomly picked genes. Assume that a standard GA is running for a predefined number of generations. Let us examine the genotypes of the evolved population. First, we classify each genotype as being either high fit or low fit, if the fitness of its phenotype is above or below some threshold. Then we try to identify features which distinguish the genotypes of the high fit phenotypes from the genotypes of the low fit phenotypes, Figure 1. Here it is easy to see that the fittest genotypes all contain the string of genes $A = \{0; 1; 2; 8; 4; 3\}$ and all that are less fit lack it.

Figure 1. The identification of the evolved gene $A = \{0; 1; 2; 8; 4; 3\}$ based on distinguishing between high and low fitness phenotypes.

Hence, we can assume that A is an evolved gene and that its presence in a genotype leads to a high level of fitness. This hypothesis can be tested by generating two random sets of genotypes: one with and one without this evolved gene and using statistical tests to check if the former one is fitter than the latter one. If A withstands this testing we declare it our current evolved gene. Now we operate under the assumption that the presence of high levels of A in the genetic pool leads to a fitter population. Hence, we want to increase the presence of A in the population. The first step is to encapsulate A into an evolved gene, and protect it from disturbance by genetic operations like crossover and mutation. This can be done by marking the sections in the genotypes that contain the evolved gene, Figure 2, or by introducing a new symbol and replacing all or a high percentage of the occurrences of the evolved gene with the new symbol, Figure 3. In the first case, the length of the genotype does not change, and a fixed-length representation can be used. In the second case, a new symbol is added to the alphabet used in the genotypes and the genotypes may change length. The encapsulated evolved genes function as fixed subassembled genetic blocks in the population.

Figure 2. Encapsulating evolved genes by restricting the choice of crossover points to regions not occupied by evolved genes (for a fixed length genotype).

Figure 3. Encapsulating evolved genes by replacing them with a newly introduced symbol (Gero and Kazakov, 1996).

Evolving a Representation

The starting point for evolving complex genes is a population of randomly created individuals using a coding with basic genes. This coding has to allow for genotypes of variable lengths. The individuals are evolved through a number of generations using a given fitness function. At the same time an additional operation identifies particularly successful combinations of genes. For every such gene combination, a new evolved gene is created that represents this combination and is then introduced into the population. At first, the evolved genes are composed of basic genes but in later cycles most evolved genes are composed of combinations of other, lower-level, evolved genes or combinations of these with basic genes. This growing hierarchy of representations gives rise to a more complex and abstract coding which contains domain specific knowledge that is 'learnt' by the system.

An evolved complex gene subsequently becomes an 'atom' in a new coding and hence the lower-level genes that are represented by it are protected from disturbance from genetic operations such as crossover and mutation. An effect of replacing a number of lower-level genes with evolved genes is that variable-length genotypes have to be catered for, even if the 'unpacked' individuals were to be of the same length.

Since the length of any genotype may not be restricted, the search space is infinitely large. It can be illustrated by an (infinite) number of concentric circles, each defining the space of solutions that can be defined by genotypes of different lengths. Solutions produced by genotypes of length one (the basic genes) are in the centre. The further away a solution is from the centre, the more complex it is and the larger the space that has to be searched to find it. Each time a complex gene is evolved the structure of the search space changes. The solution that is created from it is moved into the centre and all solutions that can be derived from it move towards the centre accordingly, Figure 4.

Figure 4. Example of an evolving representation: (a) original representation and (b) representation with evolved genes. Some of the corresponding genotypes are given, capital letters denote evolved genes. The transformation from phenotype to genotype is not always unique, e.g. the genotypes 'ABc' and BAc' produce the same phenotype. Arc segments indicate that only part of the space is shown (Schnier and Gero, 1996).

Using the Evolved Representation - a Genetic Engineering Approach

The evolved representation can be now used to solve other, similar, problems. The initial population is generated using both the original basic genes and the evolved genes. The presence of the original basic genes ensures that the whole original search space can still be searched while the evolved genes restructure the search space in favour of structures that were established as useful when the complex genes were evolved.

The introduction of evolved genes obviously changes the probability that a part of a genotype maps onto a useful feature. While the number of different genes that can be used in the genotypes expands, the length of the genotype that is required to describe a feature shrinks. The net effect is that a smaller space has to be searched. If, for example, we want to find the window-like shape composed of four squares that was used in the example above,

and assume we already know how long the genotypes have to be to find it, then the original search space would have a size of about $4^{14} = 2^{28}$, while with the evolved representation, only a space of $6^5 = \sim 2^{13}$ possible genotypes has to be searched.

To further increase the proportion of the fit evolved genes in the population, the evolved genes can be artificially introduced into genotypes lacking it, as it is done in genetic engineering. We try to make our trial population fitter before further reproduction by changing (the majority of) these genotypes which lack evolved genes in such a way that they acquire them with minimal changes. It can be done using a 'directed' macromutation, a sequence of mutations which leads to the creation of a previously evolved gene, for example $A = \{0,1,2,8,4,3\}$ in Figure 5. Minimal changes to genetic material here mean that a number of elementary mutations in a macromutation should be minimal. As a result of the introduction of evolved genes and the measures which ensure their survival at above the natural rate, the evolutionary path of a standard GA is shortcut. If the genetic encoding is fixed-length and position-dependent then the evolved genes are just building blocks of a standard GA theory, and the genetic engineering extension of GA can be viewed as a way of explicitly handling such blocks. If the coding is position-independent and variable-length then it is clear that the effect of evolved genes usage instead of original genes is twofold: first, it allows the sampling of larger areas of a search space subject to the same computational resources; and second, it allows the searching of a particular partition of the search space which contains the fittest points more thoroughly, again subject to the same computational resources.

Figure 5. The genetic 'operation' to create evolved gene $A = \{0, 1, 2, 8, 4, 3\}$ using directed macromutation (Gero et al., 1997).

After we have identified newly evolved genes and made the population genetically healthier by 'operating' on those genotypes which lack the evolved genes using directed macromutations, we run a standard GA for a predetermined number of generations. We set the mutation rate of the evolved gene's components at a much lower level than the standard mutation rate level. Then we again analyze the genetic material of the evolved population, identify newly evolved genes, test them together with previously evolved genes to produce the current set of evolved genes and 'operate' on those genotypes which lack these genes. The cycle is then repeated. In practice, the analysis of genetic material is not carried out 'manually' as we did in our illustrative example but automatically, by using one of several string analysis techniques developed in genetic engineering and speech processing, (Collins and Coulson, 1987; Karlin et al., 1990; Needleman and Wunsch, 1970; Sankoff and Kruskal, 1983; Schuler et al., 1991). In the general case, the type of evolved genes naturally developing is problem and encoding dependent. It could be any arbitrary feature of a genotype, not necessarily a substring (fixed group of genes with arbitrary ordering, periodic pattern, etc.) (Gero and Ding, 1997). If the type of evolved gene's characteristic for a particular type of problem is known then it is possible to design a much more efficient method for genetic analysis. Techniques employed during genetic 'operations' could also be designed differently. It should produce a particular type of evolved gene in a particular encoded (position-dependent, position-independent, order-based, fixed length, variable length, etc.) genotype and still lead to as little change of genetic material as possible. These conditions still leave a significant freedom of choice. The simplest way of designing such an 'operation' technique is to use macromutation (directed sequence of standard mutations)

similar to the example. Usually one of the techniques of genetic engineering of natural organisms (genetic surgery, genetic therapy, etc.) is used as a template to design a particular operation technique. If one chooses genetic surgery as a model then the genetic 'operation' changes only the genotype's pieces which are similar enough (according to some measure) to the evolved genes and not the whole trial genotype. If genetic therapy is chosen as a model of genetic 'operation' then the evolved genes are inserted into trial genotypes in variable-length genotypes or they replace pieces of equal length in fixed-length genotypes, Figure 6. Usually, a random position for this insertion or replacement is chosen. Related ideas have been developed in genetic programming (Angeline, 1994). The main differences to genetic engineering GAs include the way evolved genes are identified (using the fitness function of partial genotype or by just random selection of the parts of genotype), less with developed tools for manufacturing the population whose genotypes are reached using evolved genes, and the concept of the re-use of evolved genes in related problems is absent.

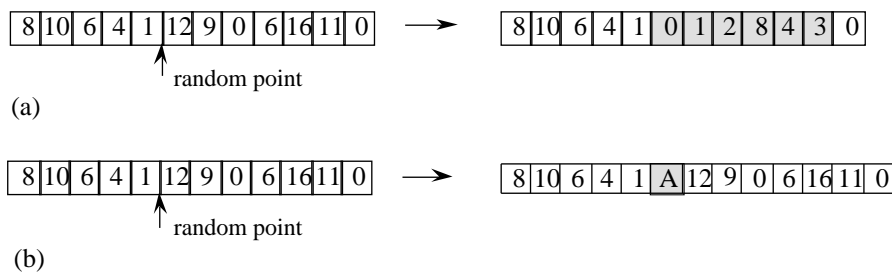


Figure 6. Genetic therapy model: (a) evolved gene $A = \{0,1,2,8,4,3\}$ replaces randomly selected part of equal length in a fixed-length genotype; (b) evolved gene $A = \{0,1,2,8,4,3\}$ is inserted at a random position into a variable-length genotype (Gero and Kazakov, 1996).

An Example: The Design of Architectural Floor Plans

In this example, we show how the idea can be used in the design of architectural floor plans. (This example is described in more detail in Schnier and Gero (1995)). As the basic coding, we use a turtle graphics-like coding with only four different basic genes, that either draw a line in the current direction, move the pen ahead, or change the current direction, Figure 7. This coding constitutes a simple grammar for constructing orthographic shapes.

Figure 7. Basic coding, arrows show pen position and current direction before and after the gene is drawn.

An Implementation for evolving the representation

The two floor plans shown in Figure 8 are used together as the designs to be represented. The fitness function compares individuals with this drawing, and rewards individuals depending on how much of the drawing they fit.

Figure 8. Room plans used as example cases (Schnier and Gero, 1995; Gero and Schnier, 1995).

To create evolved genes, successful combinations of genes in the population have to be identified. We will only consider pairs of successive genes in the creation of evolved genes, using the following algorithm:

1. Create a table of all different pairs of successive genes that occur in the population.
2. For all individuals in the population divide the fitness by the length of the individual. In the table, add this value to all pairs occurring in the genotype of that individual.
3. Find the pair with the highest sum of fitnesses in the table.
4. Create a new evolved gene with a unique designator, and replace all occurrences of the pair in the population with the new evolved gene. The number of evolved genes is kept to a certain percentage of the population (3% in the examples shown). Figure 9(a) shows a branch of the hierarchical composition of one of the evolved genes (no. 363) from lower-level evolved genes and basic genes (numbers in brackets). Figure 98(b) shows how an individual is composed from six evolved genes.

Figure 9. Evolved representation: (a) part of the hierarchical composition of the evolved gene 363; and (b) composition of the individual with the genotype (289 267 287 363 246 246) (Schnier and Gero, 1995; Gero and Schnier, 1995).

Using the representation

After a representation based on the examples in Figure 8 has been developed, it is used for new different fitness requirements. A standard evolutionary algorithm is used, where the

fitness requirements are coded into the fitness function. The representation is not evolved further, instead the set of evolved genes learned from the examples is used, together with the original basic genes. As an example, the new requirement was to create a floor plan with minimal overall wall length, while at the same time fulfilling the following additional requirements:

1. no walls with 'open ends', that is, no walls that do not build a closed room;
2. 6 rooms; and
3. room sizes 300, 300, 200, 200, 100 and 100 units.

The additional requirements were given higher priority than the minimization of the wall length.

Results

Figure 10(a) shows the result of one run, after 150,000 crossovers were performed. The population size was 1,000 individuals. 320 evolved genes created from the examples in Figure 8 were introduced into the population by using them with equal probability in the generation of the initial random population, and by the mutation operator. Shown are the best individuals (rotated copies have been omitted). To compare the performance with a standard genetic algorithm without evolving coding, Figure 10(b) shows the results of a run with only the basic coding, but identical fitness conditions. No rooms of more than unit size were produced. More than two thirds of the 15,400 genes used in the genotypes of the final population are evolved genes, encoding between 2 and 45 low-level genes. This shows that the new results indeed make use of the evolved representation to a very large degree, but at the same time use basic genes to 'fill in the holes' between evolved genes.

Figure 10. New floor plans, using coding knowledge from the example cases (see text for fitness requirements): (a) using 320 evolved genes, (b) using only the basic genes (Schnier and Gero, 1995).

Evolving Complex Design Genes Using a Hierarchical Growth Approach

While the previous example used a genetic engineering approach in which useful combinations of genes in existing solutions are found and subsequently used, the following example investigates how complex genes can be evolved through a bottom-up hierarchical growth approach. The evolutionary approach for synthesizing design solutions is based on a genotype which represents design grammar rules for instructions on locating appropriate building blocks. A decomposition/aggregation hierarchical organization of the design object is used to overcome combinatorial problems and to maximize parallelism in implementation.

An Evolutionary Model Of Design

Whereas the process of generation of form in living systems involves the placement of different kinds of protein in particular locations, we can describe the process of generation of form as involving the placement of units of different kinds of material in particular

locations. An object can be 'grown' by locating a required number of such units, i.e. cells or building blocks, one at a time in sequence. The form produced will depend on the form of the building block and the location procedures, i.e. rules of growth, used. The genotype for a homogeneous object is thus the sequence of coded instructions for selecting and locating material units. When this code is interpreted and executed, the phenotype, i.e. the object (or rather its representation), will be generated. A general model of form growth can be proposed as:

For given total units of material required
SELECT a unit of material, Mm
LOCATE unit of material, Mm relative to other units

A gene in such a model becomes (Ot, Mm, L(Mm)), where L(Mm) is the instruction for locating the unit of material Mm relative to the generated object at each step, Ot. Initially Ot is a single unit. The genotype is a sequence of such genes. Where a homogenous object is considered, the material identification is constant and the gene is basically a sequence of location operators.

Design Through Hierarchical Decomposition / Aggregation

Simon (1969) points out that it is only possible for complex organisms to evolve if their structure is organised hierarchically. The generation of an object can be achieved through the recursive generation of its components until a level is reached where the generation becomes one of generating an element which is composed of basic units. Such an approach assumes a formulation of the decomposition structure of an object.

A top-down hierarchical approach, is used in Genetic Programming, (Koza, 1992; Rosca and Ballard, 1994). The approach used here is a bottom-up multi-level approach where, at each level, a component is generated from a combination of components from the level immediately below. Figure 11.

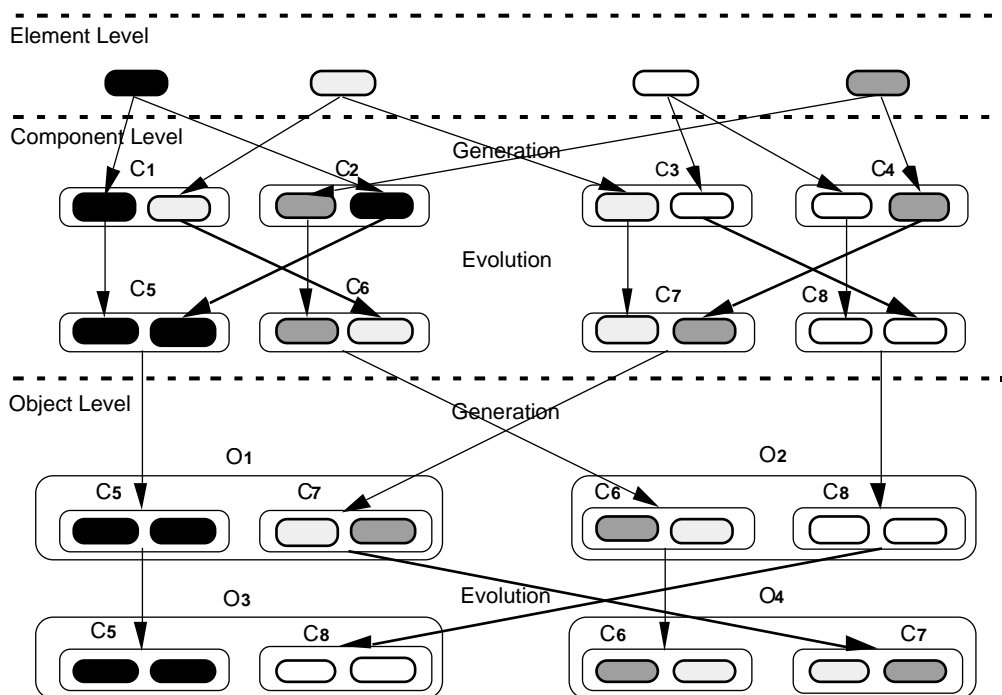


Figure 11. Multi-level combination and propagation.

At each level, an initial population is generated and then evolved over a number of generations until a satisfactory population of objects at that level is obtained. Members of that population are then selected as suitable components for generating the initial population at the next level. The process is repeated for all levels.

The advantages of such a hierarchical approach are that only those factors relevant to the design of that component are considered and factors relevant to the relationships between components are treated at their assembly level. Instead of a single long genotype consisting of a large number of basic genes, the genotype is composed of a set of chromosomes relevant to their particular level. In addition to reducing the combinatorial problem substantially, parallelism is supported since all the different chromosomes (components) at a particular level can be generated in parallel. If the set of possible alternatives of component types is sufficiently large and varied, then many different combinations of members of different such sets are possible, at the next level, with a good chance of satisfying the criteria and constraints at that level. Only when no such possible combination satisfies such criteria is there a need for some generation of new alternatives at the lower level.

In a flat model of form generation, a genotype will consist of a string of a large number of basic genes. In a hierarchical model, there are a number of component chromosomes, at different levels, consisting of much shorter strings of genes which are the chromosomes at the next lower level. All in all, the total number of basic genes will be the same in the flat and hierarchical models.

Design Grammars for Genotype Representation

Design grammars deal with a vocabulary of design elements and transformation on these elements and hence define a design space, (Stiny and Mitchell, 1978; Stiny, 1980) While design grammars provide a syntactic generative capability, they lack the evaluative mechanisms for directing the generation towards meaningful solutions. Using a design grammar as the genotype representation in an evolutionary approach allows for the generation of meaningful solutions with regards to some requirements formulated in a fitness function, (Woodbury, 1993).

The aim of the design process, in this evolutionary approach, is the attainment of a set of instructions, a genotype, that when executed, yields a design description of a product, a phenotype, whose interpreted behaviours satisfy a set of required behaviours, the fitness function. In this approach, a grammar rule is a gene, the plan (sequence of rules) is the genotype and the design solution is the phenotype. The approach taken here, is based on the premise that the grammar rules are fundamental operators, which cannot be decomposed or recomposed, that the particular grammar contains all required rules and that the aim of the design process is to find satisfactory sequences of such rules.

A General Model for A Hierarchical Evolutionary Approach to Design

The general model of design using a hierarchical evolutionary approach may be stated as follows:

for all levels in the object hierarchy

for all components at that level

GENERATE initial population of members by synthesizing lower level units through random application of rules

EVOLVE population until satisfactory

A House Design Example - Space Generation

The above concepts can be exemplified through the generation of 2-D plans for single-storey houses. Previous work demonstrated that a single-level approach was not able to converge towards satisfactory solutions mainly due to the interactions of the various factors of the fitness function required for the various elements, (Jo, 1993; Jo and Gero, 1995). A more detailed presentation of this work can be found in Rosenman (1996).

A house can be considered to be composed of a number of zones, such as living zone, entertainment zone, bed zone, utility zone, etc. Each zone is composed of a number of rooms (or spaces), such as living room, dining room, bedroom, hall, bathroom, etc. Different houses are composed of different zones where each zone may be composed of different rooms. Each room is composed of a number of space units. Generally, in a design such as a house, the space unit will be constant. The scale (level of abstraction) of the space unit depends on the precision required in differences between various possible room sizes. The

smaller the unit, the longer the genotype for a given size of room but the greater the shape alternatives.

The Design Grammar

In the above formulation, the generation of spaces, basically comes down to locating spatial component units for that level. At the room level, the component unit is a fundamental unit of space. At the zone level, the component unit is a room and at the house level the component unit is a zone.

The design grammar used here is based on the method for constructing polygonal shapes represented as closed loops of edge vectors, (Rosenman, 1995). The grammar is based on a single fundamental rule which states that any two polygons, P_i and P_j , may be joined through the conjunction of negative edge vectors, V_1 and V_2 , (equal in magnitude and opposite in direction). The conjoining of these vectors results in an internal edge and a new polygon, P_k . This rule ensure that new cells are always added at the perimeter of the new resultant shape.

The fundamental conjoining rule can be specialized for different types of geometries. Orthogonal geometries are based on the following four vectors of unit length: $W = (1, 90)$, $N = (1, 0)$, $E = (1, 270)$, $S = (1, 180)$ so that the two pairs of negative vectors are $N - S$ and $E - W$. These two pairs of negative vectors allow for the generation of all polyminoes. Orthogonal geometries will be used in this example without loss of generality. Other (sub)rules may be formed for other geometries, (Rosenman, 1995).

Genotype and Phenotype

A polygon is described by its sequence of edge vectors. A suffix is used to identify individual edges of the same vector type. Thus the square cell of Figure 12 is described as (W_1, N_1, E_1, S_1) . The sequence of edge vectors for a shape is the phenotype providing the description of that shape's structure. The genotype for any generated polymino is the sequence of the two subshapes (polyminoes) used and the two edges joined. An example of the generation of a trimino is shown in Figure 12. Figure 12 shows a basic unit or cell, P_1 , which provides a starting point for the generation of polyminoes. Each generated shape is accompanied by its genotype and phenotype. The generation of these polyminoes occurs from a random selection of edges in the first shape conjoined with a random selection from equal and opposite edges in the second shape. At each step in the generation, the phenotype is reinterpreted to generate a new edge vector description and the conjoining (sub)rules applied. The genotype for the generated trimino is given as $(P_2, P_1, N_2|S_1)$. This can be expanded as $((P_1, P_1, E_1|W_1), P_1, N_2|S_1)$. When the same units are used for generation, the unit can be omitted and the genotype represented as the sequence of edge vector conjoinings. That is $P_3(g) = (E_1|W_1, N_2|S_1)$. The length of the genotype depends on the size of the polymino to be generated, that is on the area of the polymino. This corresponds to required room sizes.

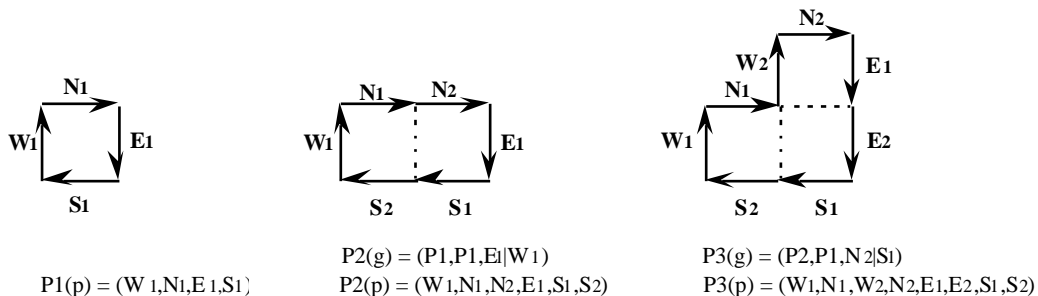


Figure 12. Generation of a Trimino

Once a population of different rooms is generated for each room type in a given zone, the zone can be generated through the conjoining of rooms in a progressive fashion. Because of the cell-type structure of the polygons, the conjoining may occur at any appropriate pair of cell edges. Therefore, a large number of possible zone forms can be generated from two

rooms. An example of some possibilities arising from the conjoining of two polyminoes is given in Figure 13.

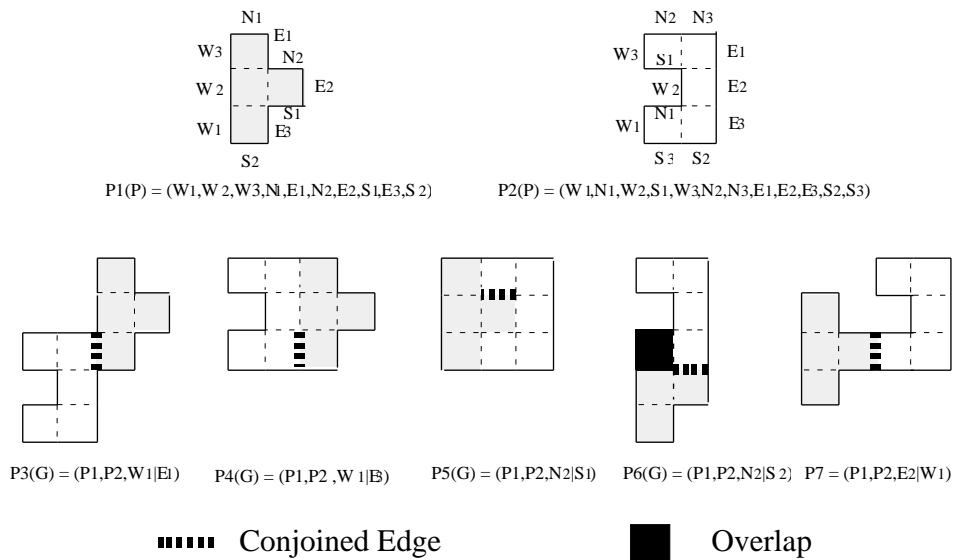


Figure 13. Some Examples of Conjoining Two Polyminoes

The two polyminoes, P1 and P2, represent instances of two different room types and the polyminoes resulting from the joining of the two rooms represent instances of a particular zone type. When one pair of edges are conjoined other edges may also be conjoined, e.g. P4, P5 and P6. In the case of overlap, as in P6, the resultant shape is discarded.

The same process used for generating zones is used to generate houses. The joining of different instances of different zone types generates different instances of houses.

The Evolution Of House Designs

The above grammar can be used to generate initial populations for each level in the spatial hierarchy. Each such initial population is then evolved, as necessary, so that solutions are 'adapted' to design requirements.

The Evaluation Criteria - Fitness Functions

At each level, different fitness functions apply according to the requirements for that level. While the requirements for designs of houses involve many factors, many of which cannot be quantified or adequately formulated in a fitness function, some simple factors have been used initially to test the feasibility of the approach. For this example, the fitness function for rooms consists of minimizing the perimeter to area ratio and the number of angles. This requirement tends to produce compact forms, useful as rooms. For zones, the fitness function consists of minimizing a sum of adjacency requirements between rooms reflecting functional requirements. At the house level, the fitness function consists of minimizing a sum of adjacency requirements between rooms in one zone and rooms in other zones. This has the tendency to select those arrangement of zones where adjacency interrelations are required between rooms of different zones. In addition to these quantitative assessments, qualitative assessments will be made subjectively and interactively by a user/designer.

Although the above criteria have been described in terms of optimizing functions, the aim is not to produce global optimum solutions but rather to direct the evolutionary process to produce populations of good solutions either as components for higher levels or at the final level itself. So that, even though the global optimum solution for the shape of a room using the above criteria, may be known, this may not be the optimum solution at the zone and house levels. By selecting other non-optimal but good solutions, according to the given criteria, good unexpected results may be achieved for the overall design.

Propagation - Crossover

Simple crossover is used for the production of 'child' members during the evolution process. Looking first at the room level to see the effect of such a crossover process, crossover can occur at any of the four sites as shown in Figure 14(a) with two results as shown in Figure 14(b). Since we are always dealing with cells of the same space unit, the cell identification in the genotype representation has been omitted for simplicity.

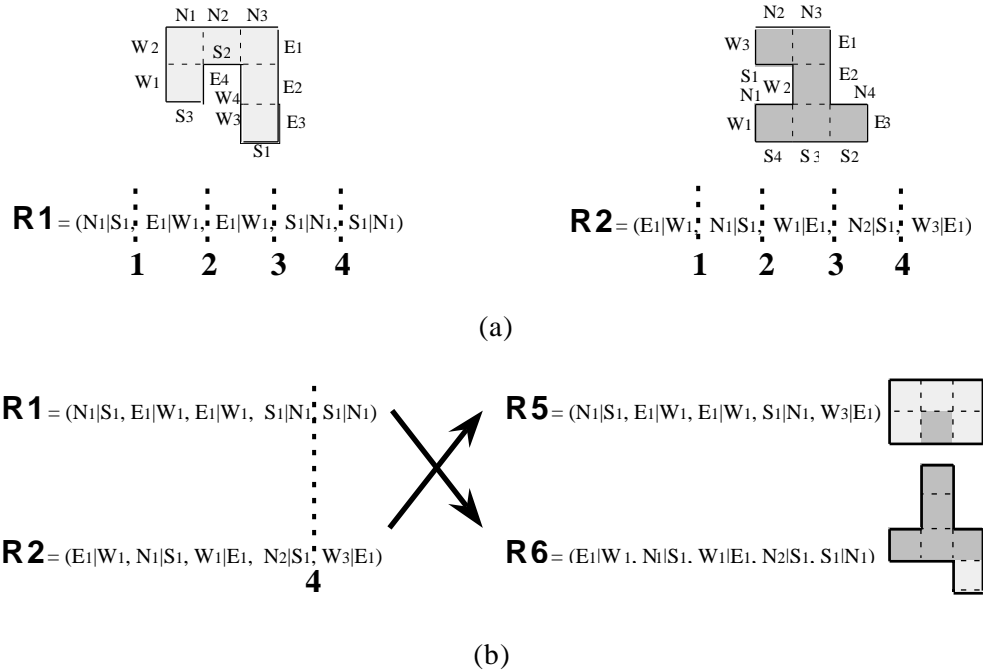
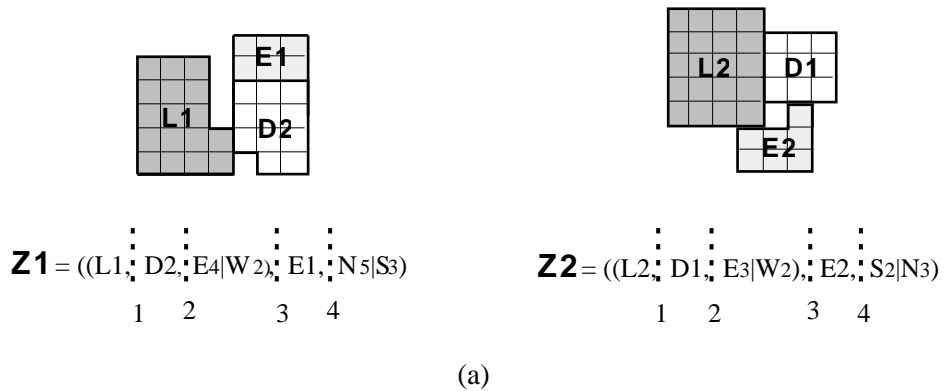


Figure 14. Crossover at Room Level; (a) initial rooms R1 and R2 generated from unit square cell U1, (b) crossover at site 4

At the zone level, crossover occurs as shown in Figure 15. Two initial instances of living zones, Z1 and Z2 are shown in Figure 15(a). Each zone has one instance of each of living room, dining room and entrance. Figure 15(b) shows crossover for one of the four possible sites. A similar process is followed at the house level.



(a)

SITE 2 - CROSSOVER



$$\mathbf{Z5} = ((L1, D2, E3|W2), E2, S2|N3)$$

$$\mathbf{Z6} = ((L2, D1, E4|W2), E1, N5|S3)$$

(b)

Figure 15. Examples of Zone Crossover; (a) rooms and initial zones, Z1 and Z2, (b) crossover at Site 2

Implementation And Results

A computer program written in C++ and Tcl-Tk under the Sun Solaris environment has been implemented using the simple criteria described previously. Each evolution run, for all levels, tends to converge fairly quickly to some dominant solution. Rather than use a mutation operator to break out of such convergence, it was found that a more efficient strategy was to generate multiple runs with different initial randomly generated populations. This produces a variety of gene pools thus covering a more diverse area of the possible design space. Moreover, such runs can be generated in parallel. Users can nominate the population size, number of generations for each run and select rooms, zones and houses from any generation in any run as suitable for final room, zone or house populations. These selections are made interactively by users as solutions appear which are judged favourable, based perhaps on factors not included in the fitness function. Such selections may therefore not be optimal according to the given fitness function.

Results are shown in the following figures, Figures 16 to 19 for room, zone and house solutions.

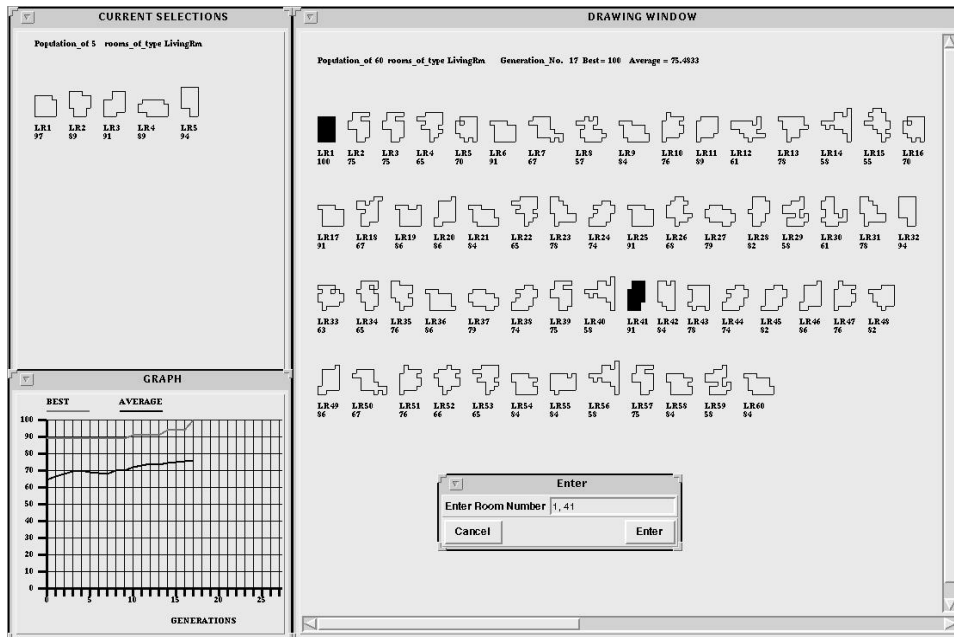


Figure 16. Results of Living Room Generation after the 17th generation.

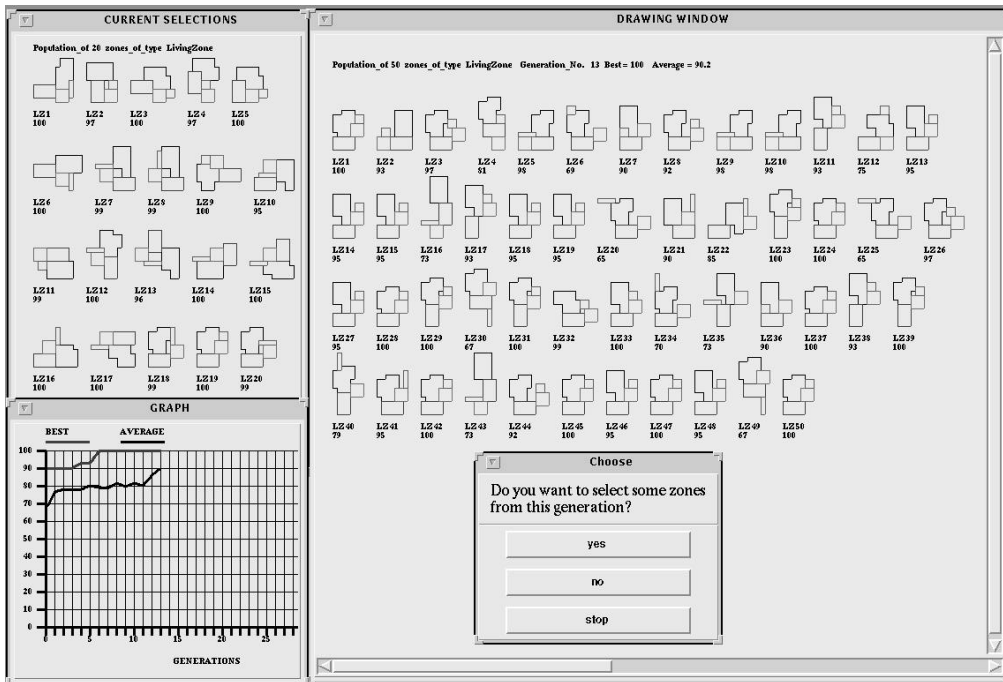


Figure 17. Results of Living Zone Generation.

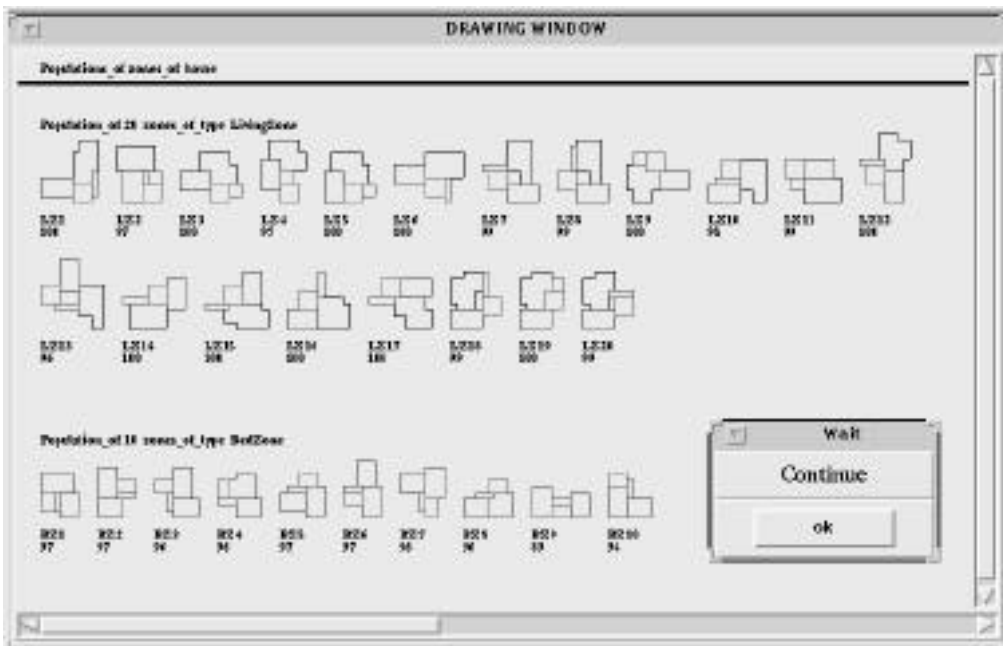


Figure 18. Results of Bed and Living Zones Generation

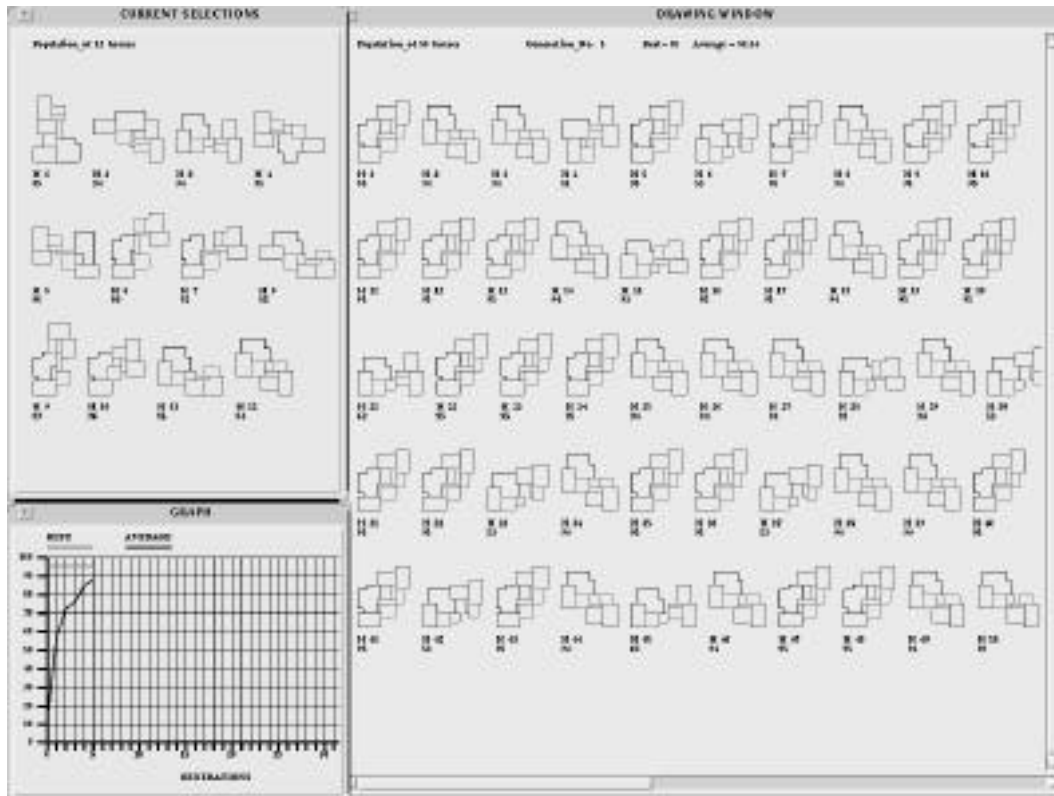


Figure 19. Results of House Generation.

Figure 16 shows the results of the 8th run of the generation of Living Rooms where the first 4 room shapes were selected from 7 previous runs. Figure 16 shows the 17th generation of the evolution of this population of 60 members. A fifth room shape was selected at the 14th generation and two more room shapes (Room Numbers 1 and 41) are being selected. The upper line in the graph shows the evolution of the best solution while the lower line shows the evolution of the population average. All in all for this example, a total of 10 shapes were selected from 13 runs. The maximum number of generations before convergence for a run was 50 and the minimum 19 with an average of 32.

Other rooms were generated in a similar way. The room areas generated were: (a) Living Zone: Living Room 24; Dining Room 15; Kitchen 9; Entrance 4; (b) Bedroom Zone: Master Bedroom 15; Bedroom 12; Bathroom 6; Hall 3. Figure 16 shows the results of the Living Zone generation. The initial population of 50 Living Zones at run 1 was randomly generated by selecting rooms from the final selections for the Living Room, Dining Room, Kitchen and Entrance. Figure 17 shows the 13th generation of the final run, run 7. Twenty Living Zones have been selected by the user. Figure 18 shows the set of Bedroom and Living Zones selected. Figure 19 shows the final set of houses generated in this example. All in all 7 runs were carried out for a total of 12 suitable house plans.

The total area of this house type is 88 sq units, corresponding to a genotype of length 87 in a single genotype. The example of Jo (1993), showed that no satisfactory convergence was obtained with a single genotype of this length. The size of the population and the number of generations and runs required to generate a satisfactory number of members depends on the genotype length. The longest genotype was of length 23, for the Living Room. The addition of more zones and rooms presents no problem for the hierarchical approach used here although it would present extra combinatorial problems for the single genotype approach.

Summary

Two approaches were presented, one using a learning approach to evolving complex gene structures from a given population of design solutions while the other uses a hierarchical growth approach. Each approach identifies or creates complex building blocks which are then treated as individual components of the overall design solution or of a higher-level component assembly. The growth approach uses a more strictly ordered hierarchical approach in which components are generated of the next level in the component/assembly hierarchy whereas the learning approach is able to use any level of component. The growth approach explicitly generates recognizably meaningful component building blocks whereas the learning approach generates component building blocks which may not be recognizable as entities. In general, the hierarchical growth approach is not necessarily restricted to component/assembly structures using the same gene codings and, in addition, the fitness functions used for the various components and assemblies are different reflecting relevant requirements. In the learning approach presented, the one fitness function is used to determine the evolved genes at all levels, although multiple fitnesses can be used (Gero and Ding, 1997)

What both approaches have in common is that they demonstrate the advantages of shortening the genotype representation by creating hierarchies of increasingly complex gene structures and restructuring the search space thus focussing the search for useful solutions much more efficiently. The advantage gained is that a linear increase in genotype representation is obtained as against an exponential increase, similar to that gained by stage-state decomposition optimization methods, such as dynamic programming.

These approaches lay the foundations for the development of tools to aid designers arrive at unexpected solutions for those classes of design problems for which evolved complex genes are beneficial, namely those design problems for which a hierarchical structure of components can be formulated.

Acknowledgments

This work is supported by the Australian Research Council Large Grant Scheme. Part of the work reported here was done in collaboration with Vladimir Kazakov and Thorsten Schnier.

References

- Angeline, P. (1994). Genetic programming and emergent intelligence, in K. Kinnear (ed.), *Advances in Genetic programming*, MIT Press, Cambridge, MA, pp.75-98.
- Collins, J. and Coulson, A. (1987). Molecular sequence comparison and alignment, *Nucleic Acid and Protein Sequence Analysis: A Practical Approach*, IRL Press, Washington DC, pp. 323-358.
- Gero, J. S. and Ding, L. (1997). Learning emergent style using an evolutionary approach, in B. Varma and X. Yao (eds), *ICCIMA'97*, Griffith University, Gold Coast, Queensland, Australia, pp. 171-175.
- Gero, J. S. and Kazakov, V. (1996) A genetic engineering extension to genetic algorithms, *Working Paper*, Key Centre of Design Computing, University of Sydney.
- Gero, J. S. and Schnier, T. (1995). Evolving representations of design cases and their use in creative design, in J. S. Gero, M. L. Maher and F. Sudweeks (eds), *Preprints Computational Models of Creative Design*, Key Centre of Design Computing, University of Sydney, pp. 343-368.
- Gero, J. S., Kazakov, V. A. and Schnier, T. (1997). Genetic engineering and design problems, in D. Dasgupta and Z. Michalewicz (eds), *Evolutionary Algorithms in Engineering Applications*, Springer-Verlag, Berlin, pp.47-68.
- Jo, J. H. (1993). *A Computational Design Process Model Using a Genetic Evolution Approach*, PhD Thesis, Department of Architectural and Design Science, University of Sydney, (unpublished).
- Jo, J. H. and Gero, J. S. (1995). A genetic search approach to space layout planning, *Architectural Science Review*, **38**(1):37-46.
- Karlin, S., Dembo, A. and Kawabata, T. (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring scheme, *Proceedings of the National Academy of Science USA*, **87**: 5509-5513.

- Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, Mass.
- Needleman, S. and Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins, *Journal of Molecular Biology*, **48**: 443-453.
- Rosca, J. P. and Ballard, D. H. (1994). Hierarchical self-organization in genetic programming, *Proc. of the Eleventh Int. Conf. on Machine Learning*, Morgan-Kaufmann, San Mateo, CA, pp.252-258.
- Rosenman, M. A. (1995). An edge vector representation for the construction of 2-dimensional shapes, *Environment and Planning B: Planning and Design*, **22**: 191-212.
- Rosenman, M. A. (1996). The generation of form using an evolutionary approach, in J. S. Gero and F. Sudweeks (eds), *Artificial Intelligence '96*, Kluwer Academic, Dordrecht, The Netherlands, pp.643-662.
- Sankoff, D. and Kruskal, J. (eds) (1983). *Time Warps, String and Macro-molecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, MA.
- Schnier, T. and Gero, J. S. (1995). Learning representations for evolutionary computation, in X. Yao (ed.), *AI'95 Eighth Australian Joint Conference on Artificial Intelligence*, World Scientific, Singapore, pp. 387-394.
- Schnier, T. and Gero, J. S. (1996). Learning genetic representations as alternative to hand-coded shape grammars, in J. S. Gero and F. Sudweeks (eds), *Artificial Intelligence in Design'96*, Kluwer, Dordrecht, pp.39-57.
- Schuler, G., Altschul, S. F. and Lipman, D. J. (1991). A workbench for multiple alignment construction and analysis, *PROTEINS: Structure, Function, and Genetics*, **9**: 180-190.
- Simon, H. A. (1969). *The Sciences of the Artificial*, MIT Press, Cambridge, Mass.
- Stiny, G. (1980). Introduction to shape and shape grammars, *Environment and Planning B*, **7**:343-351.
- Stiny, G. and Mitchell, W. (1978). The Palladian grammar, *Environment and Planning B*, **5**:5-18.
- Woodbury, R. F. (1993). A genetic approach to creative design, in J. S. Gero and M. L. Maher (eds), *Modeling Creativity and Knowledge-Based Creative Design*, Lawrence Erlbaum, Hillsdale, NJ, pp.211-232.

This paper is a copy of: Rosenman, M. A. and Gero, J. S. (1999). Evolving designs by generating useful complex gene structures, in P. Bentley (ed.), *Evolutionary Design by Computers*, Morgan Kaufmann, London, pp. 345-364.