

GENETIC ENGINEERING AND DESIGN PROBLEMS

John S. Gero, Vladimir A. Kazakov, and Thorsten Schnier
Key Centre of Design Computing
Department of Architectural and Design Science,
The University of Sydney NSW 2006 Australia.
E-mail:{john,kaz,thorsten}@arch.su.edu.au

1. GENETIC ENGINEERING EXTENSIONS OF GAs

Genetic engineering is a technology for direct intervention in the genetic structure of natural organisms. It is used for the rapid manufacture of improved organisms. Usually, its implementation includes two stages. The first stage is devoted to an analysis of the genetic material of highly successful (with respect to some criterion) organisms. The goal here is to identify genetic features (we shall call them 'evolved' genes) directly linked to their high level of success. The second stage includes attempts to manufacture improved organisms using these evolved genes. This includes some form of direct manipulation of genotypes of the trial organisms which results in the acquisition of these evolved genes by their genotypes. For example, severe combined immunodeficiency (SCID), an illness that occurs if a defective gene is inherited from both parents, can be treated by inserting normal copies of the gene into the person's white blood cells (Anderson 1995).

It seems natural to extend genetic algorithm paradigms on the basis of this simple model of genetic engineering practice. Let us illustrate how it can be done using a simple problem where solutions are coded as strings of 19 different integer numbers from the range $[0, 18]$. Some fitness function is defined over the phenotype. 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, Figure 1. 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. 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.

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 one of statistical tests to check if the former one

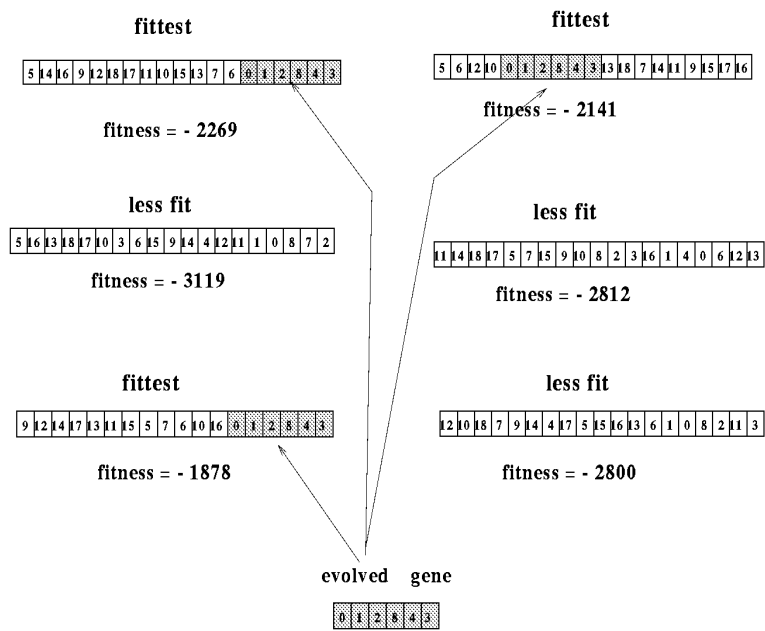


Fig. 1: The identification of the evolved gene $A = \{0, 1, 2, 8, 4, 3\}$ based on distinguishing between high and low fitness phenotypes.

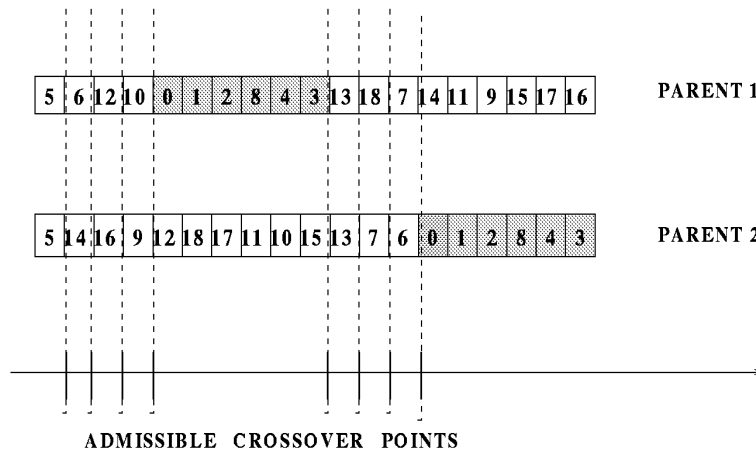


Fig. 2: Encapsulating evolved genes by restricting the choice of crossover points to regions not occupied by evolved genes (for a fixed length genotype).

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 cross-over 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. This is illustrated in Figure 4. Again, the problem is coded into strings of integer numbers in the range of $[0, 18]$. In the figure, the set of all possible genotypes of a certain length is represented by one of the concentric arcs. In Figure 4(a), only the basic alphabet of integer values is used. If analysis shows that individuals with the sequence 0, 1, 2 and the sequence 8, 4, 3 are found to be particularly successful (highlighted in the figure), then evolved genes can be created and added to the alphabet (represented by the symbols A and B). In Figure 4(b), the new situation is shown. The sequence 0, 1, 2, 8, 4, 3 can now be represented by a genotype

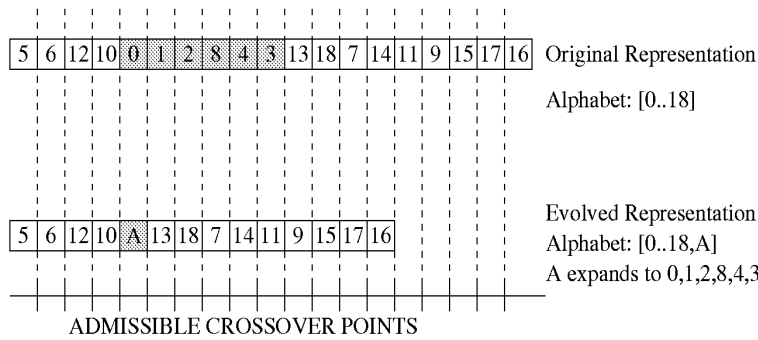


Fig. 3: Encapsulating evolved genes by replacing them with a newly introduced symbol.

of length two, instead of six as in Figure 4(a). If this sequence is found to be successful, a new evolved gene can be created for it in the next round of genetic engineering.

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 gene *A* in such a way that they acquire it with minimal changes. It can be done using a 'directed' macromutation - a sequence of mutations which leads to the creation of *A*, 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 GAs 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 (the sample points are further apart from each other) 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.

Another advantage of the proposed approach is the possibility to use the evolved genes in a new environment. The evolved genes contain information

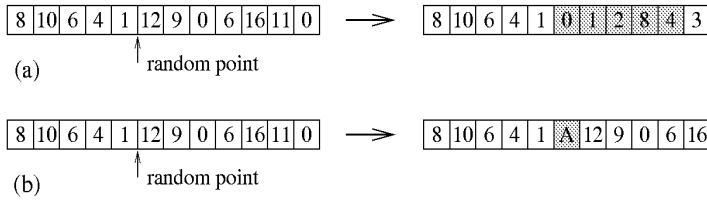


Fig. 6: Genetic therapy model: (a) evolved gene $A = 0, 1, 2, 8, 4, 3$ replaces randomly selected part of equal length in a fixed-length genotype; (b) evolved gene $A = 0, 1, 2, 8, 4, 3$ is inserted at a random position into a variable-length genotype.

about the problem that can be re-used to help solve similar problems. An analogy from the practice of genetic engineering here is the incorporation of genes of a bacterium (*Bacillus thuringiensis*) that encode a certain toxin into plants like tomato, potato, cotton and rice to give them resistance against some leaf-eating caterpillars (Plucknett & Winkelmann 1995).

After we have done this - 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 the next predetermined number of generations. Since we do not want reproduction to damage the evolved genes which are now present in the genetic material we choose the crossover point only to be outside the genotype's sections occupied by this evolved gene in any of the parents, Figure 2, and 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 by automatically, by using one out of a vast arsenal of string analysis techniques developed in genetic engineering and speech processing, (Collins & Coulson 1987, Karlin & Dembo 1990, Karlin, Dembo & Kawabata 1990, Needleman & Wunsch 1970, Sankoff & Kruskal 1983, Schuler, Altschul, S.F. & D.J 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.). If the type of evolved gene's characteristic for particular type of prob-

also be designed differently. It should produce a particular type of evolved genes in a particularly encoded (position-dependent, position-independent, order-based, fixed length, variable length, etc.) genotype and still lead to as little changes 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 a macromutation (directed sequence of standard mutations) similar to the example. Usually one of the techniques of genetic engineering of the 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 not the whole trial genotype but only its pieces which are similar enough (according to some measure) to the evolved genes. Note that in this case the locations of newly created evolved genes are determined by the structure of the 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 of this insertion or replacement is chosen.

In the remainder of this chapter we present two applications of genetic engineering in GAs. The first deals with spatial layout planning problems whilst the second deals with case-based layout design.

2 SPATIAL LAYOUT PLANNING PROBLEM.

2.1 Introduction.

Spatial layout problems have numerous applications in architectural design, VLSI design, etc. We use its formalization as a capacitated quadratic assignment problem (Liggett 1985). The number of activities to be placed, m , their areas $\{a_0, \dots, a_{m-1}\}$ (in terms of elementary square units) are given as well as the set of feasible locations, n , their areas, $\{l_0, \dots, l_{n-1}\}$ and the matrix of distances between them, $|d_{k,n}|, k, n = 0, \dots, n-1$. The "forces" of interactions between activities, $|q_{i,j}|, i, j = 0, \dots, m-1$ are also given. The objective is to find a feasible layout of activities such that its cost, I , is minimal

$$-I = \sum_{i,j} q_{i,j} d_{\rho(i),\rho(j)} \rightarrow \max$$

The mapping $\rho(i)$ of the activity i onto the feasible location is defined using an order-based genetic representation. In order to define it for this problem we choose a trajectory (path) along the possible locations. The path should be continuous within each of the closed locations, and adjacent non-overlapping elementary squares should cover all feasible locations.

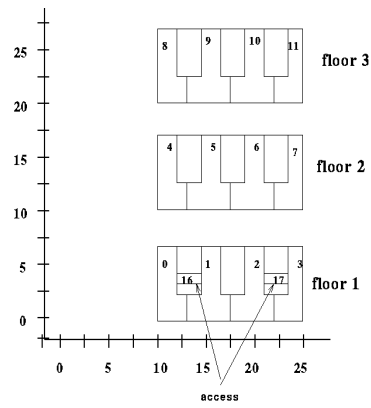


Fig. 7: Zone definition - graphical representation.

2.2 Example of space layout planning problem

As the test example we use the problem of the placement of a set of office departments into a four-storey building (Liggett 1985). The areas of the $m = 19$ activities (office departments) are defined in Table 1 in terms of elementary square modules. There is one further activity (number 19) whose location is fixed. The interaction matrix q_{ij} , $i, j = 0, 19$ is given in Table 2. There are $n = 18$ feasible zones or locations numbered from 0 to 17, Figure 7. The areas of these zones are defined in Table 3. The activity number 19 is an access area which has a fixed location - zones numbers 16 and 17, Figure 7. Since the layout cost does not depend on the areas of the zones numbered 16 and 17 or on the area of the activity number 19 they are not shown in Table 3. The travel matrix between these zones is defined in Table 4 The genetic representation is defined using the path shown in Figure 8.

Activity Number	Activity Name	Number of modules
0	Dept. 0210	2
1	Dept. 0211	2
2	Dept. 0220	8
3	Dept. 0230	8
4	Dept. 0240	15
5	Dept. 6815	13
6	Dept. 0300	15
7	Dept. 0400	7
8	Dept. 0500	6
9	Dept. 0600	12
10	Dept. 0700	53
11	Dept. 6300	10
12	Dept. 6881	16
13	Dept. 0800	18
14	Dept. 0900	31
15	Dept. 1000	61
16	Extra module	1
17	Extra module	1
18	Extra module	1

Table 1. The definition of the activities ((Liggett 1985)).

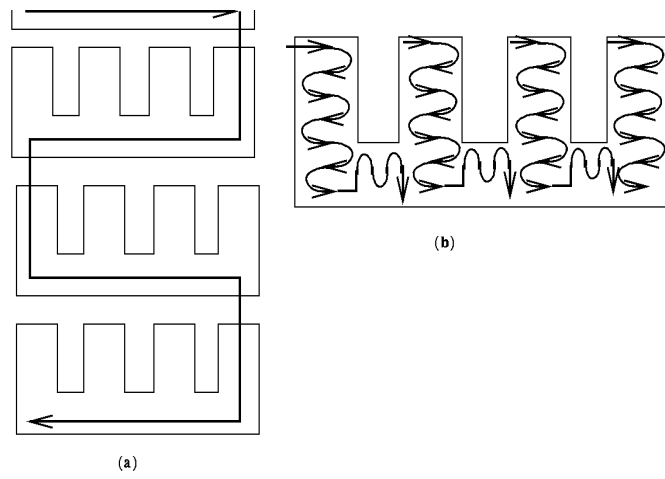


Fig. 8: The pattern used to map the genotype onto the phenotype: (a) The order of between floor mapping and (b) of the mapping within the floor.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0		3	3	2	2	2			3							2				3
1			3	2	2	2			3							2				
2				2	2			3							2					
3					3	2			3							2				
4						2			3							2				3
5									3							2				
6														3		2				
7														3		2				
8																2				3
9															2					
10															3	2				
11															3	2				
12															3	2				3
13																2				
14																2				
15																2				3
16																				
17																				
18																				
19																				

Table 2. Activity interactions matrix ((Liggett 1985)).

zone	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N Modules	20	22	20	20	18	20	18	18	16	18	16	16	14	16	14	14

Table 3. Zone definition (measured in number of modules) used by Liggett ((Liggett 1985)).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0																		
1		5	17	23	5	6	19	24	6	7	20	25	7	8	21	26	2	20
2			13	18	6	5	18	23	7	6	19	24	8	7	20	25	2	15
3				5	23	18	5	6	24	19	6	7	25	20	7	8	15	2
4					24	19	6	5	25	20	7	6	26	21	8	7	20	2
5						5	18	23	5	6	19	24	6	7	20	25	3	21
6							13	18	6	5	18	23	7	6	19	24	3	16
7								9	23	18	5	6	24	19	6	7	16	3
8									24	19	6	5	25	20	7	6	21	3
9										5	18	23	5	6	19	24	4	22
10											13	18	6	5	18	23	4	17
11												5	23	18	5	6	22	4
12													24	19	6	5	17	4
13														5	18	23	5	23
14															13	18	5	18
15																5	18	5
16																	23	5
17																		18

Table 4. Distance matrix ((Liggett 1985)).

This yields an order-based genetic representation which is used to illustrate the principles of genetic engineering based GA in the previous section.

2.3 Evolved genes of the spatial layout problem.

One feature of the example example is the character of genetic regularities which naturally occur in spatial layout problems - there are clusters of genes, irrespective of the order of the genes within the cluster, Figure 11. In another words in many spatial layout problems the cost of a layout whose genotype includes some cluster of genes is significantly better than the one without such a cluster (at least on average). This cluster of genes becomes an evolved gene. Since we employ a continuous genetic representation these clusters of genes (evolved genes) are mapped onto a spatially compact group of activities (although the actual configuration of these activities varies when their positions within genotypes or actual order of genes within clusters change). This is the equivalent of treating a group of activities as a new “super-activity” which is never disaggregated. This new super-activity can be located anywhere in the layout in the same way as any other activity. The improvement of the cost that can be achieved by an optimal ordering genes within such ”super-group” is much smaller. Correspondingly, a typical run of a standard GA proceeds in two stage. During the first stage clusters are evolved and much of the cost improvement achieved while the second stage is devoted to optimal ordering of the genes within each cluster. Although the typical GA run actually follows this scenario it still tries to perform both of them simultaneously which leads to a waste of computational resources.

It seems natural to design an optimization procedure which operates according to this scenario, separating both searches. First, it identifies these gene groups and chooses their optimal positions with respect to each other (which is the major source of cost savings). This yields an optimization problem with a smaller number of parameters than the length of the initial genotype (instead of the positions of all genes in such group we have just the position of the whole groups). Second, it chooses positions of the gene-components (activities) within each of these groups (which improves the cost

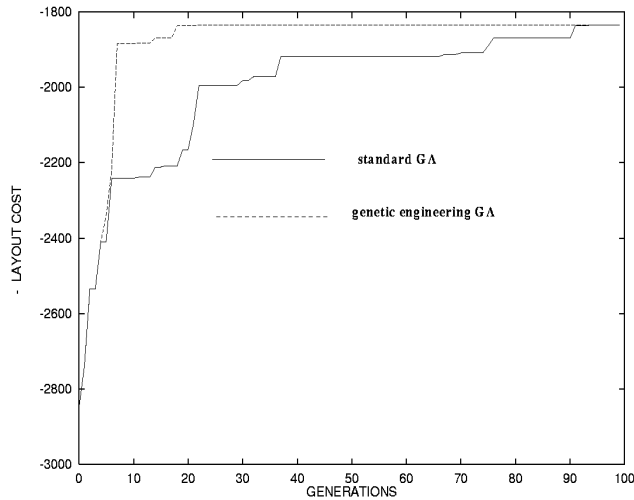


Fig. 9: The best fitness vs. generation number for the standard GA and genetic engineering based GA in Example. The results are averaged over 10 runs with different initial random seeds, which converge to the best solution found.

only marginally). Since the effective size of the search space in these two subsearches is smaller than the overall size of the search space of the original layout planning problem it can either be done faster or produce a better performing solution for the same computational expense. It is also clear that the local search in such large-scale space of “granulated” cells corresponds to the non-local search in the original state space.

Note that one does not have to establish beforehand the existence of the evolved genes in such a form - one can simply try an algorithm based on a corresponding notion of the “group”-like evolved genes and a two level optimization process. Its success would be a sufficient condition for the existence of such genes. Otherwise the genetic engineering based GA degenerates into a standard GA.

2.4 Results

The result of the run of the standard GA which converges to the best solution (averaged over 10 such runs from different initial random seeds) is presented in Figure 9 (bold line). The population size was 200, probability of crossover 0.6 and the probability of mutation 0.01. We use elitist GA with a generation gap of 3. 13 of the 100 GAs runs converged to the solution $\{7, 6, 13, 17, 14, 12, 15, 5, 18, 11, 10, 9, 0, 1, 2, 8, 3, 16, 4\}$ with the cost of the corresponding layout of 1834. The analysis of the evolutionary path of the GA shows that the search process consists of a first stage (5-10 generations)

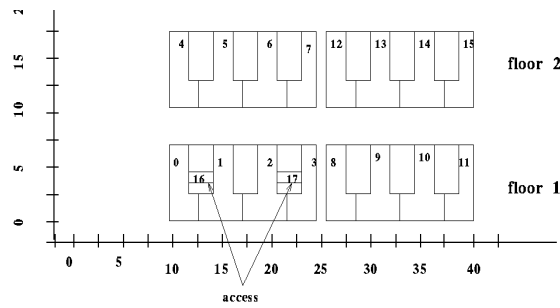


Fig. 10: Modified floor locations from Example - graphical representation.

when the crossover serves as the major constructive tool and when non-local search takes place and of a second stage when pair-wise swapping of the genes is the driving force of a search. During the first stage the algorithm finds the point which belongs to the basin of attraction of one of the minima. During the second stage (about 90 generations) crossover does not make any contribution to the search, the actual improvement is due to mutations only and the algorithm actually searches locally within the basin of attraction just found.

The result of the genetic engineering based GA are presented in Figure 9 with the dotted lines. Two evolved genes were identified after 5 generations: $\{0, 1, 2, 8, 3, 4\}$ and $\{12, 14\}$. After generation 10 two further evolved genes evolved, $\{6, 13\}$ and $\{15, 12, 14\}$, which includes the previously evolved gene $\{12, 14\}$. On average just two runs were required with different initial seeds to find the layout with the cost 1834.

The computational savings in terms of generations are of the order of 90% (although some extra computations have to be done in order to facilitate the extra processing caused by additional gene analysis and processing).

2.5 Using previously evolved genes to solve families of layout planning problems

In many cases one has to solve a number of similar layout planning problems (for example, to place essentially the same set of activities into different locations, etc). Usually, it is very difficult to use information about optimal layouts already designed to generate solutions for a similar problem.

The major advantage of the genetic engineering based GA is the possibility to re-use the evolved genes (information about optimal sublayouts) in a family of similar problems. This possibility is based on an intuitively obvious

assumption that in many layout planning problems some activities “gravitate” to each other much more strongly than they are attracted to the rest of the activities and therefore should be placed as a compact spatial group in any layout. If the possible locations (distance matrix) is changed then one still has to place such activities in a compact spatial group (although the actual physical placement could be quite different). Even the changes to the activity’ interaction matrix could preserve the attraction of some activities to each other compared to their attraction to the rest of activities, if the actual scale of attraction of the activities within the group with respect to the ones outside it do not change.

In order to check this intuition we first modified the problem data in Example, by changing the geometry of the system to be that shown in Figure 10. This yields the distance matrix shown in Table 5.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
0																		
1		5	17	23	5	6	19	24	35	40	53	58	36	41	54	59	2	20
2			13	18	6	5	18	23	30	35	48	53	31	36	49	54	2	15
3				5	23	18	5	6	18	23	36	41	19	24	37	42	15	2
4					24	19	6	5	13	18	31	36	14	19	32	37	20	2
5						5	18	23	40	45	58	63	30	36	49	55	3	21
6							13	18	35	40	53	58	25	31	44	50	3	16
7								9	22	27	40	45	12	23	31	37	16	3
8									13	18	31	36	5	10	23	28	21	3
9										5	18	23	5	10	23	28	4	22
10											13	18	10	5	18	23	4	17
11												5	23	18	5	10	22	4
12													28	23	10	5	17	4
13														5	18	23	5	23
14															13	18	5	18
15																5	18	5
16																	23	5
17																		18

Table 10. The modified distance matrix for Example 3, which corresponds to the floor locations shown in Figure 10.

The activity interaction matrix remains the same. The results of the optimization are shown in Figure 12. We run the standard GA and genetic engineering based GA beginning with the empty set of evolved genes (dotted line) and from the previously evolved one(dotted line). We can see that the re-use of the evolved genes yields about a 20% savings in terms of the number of generations over the genetic engineering based GA.

3. CASE-BASED LAYOUT DESIGN

3.1 Introduction

Case-based reasoning has been introduced into design to allow the reuse of knowledge from design cases. It is based on the observation that designers very often reuse features of previous designs (their own or other designers) rather than having to start from first principles or compiled knowledge with every new design (Maher, Balachandran & Zhang 1995). The word ‘feature’ is used here in a very general way, for a building, it could mean layouts of rooms, the general topology, materials and material combinations, structural details, form elements, etc. Case-based reasoning is different from knowledge-based design systems in that the expert knowledge is not compiled and stored, but is available only implicitly in a database of previous design cases. Since requirements and environmental conditions will usually vary between the

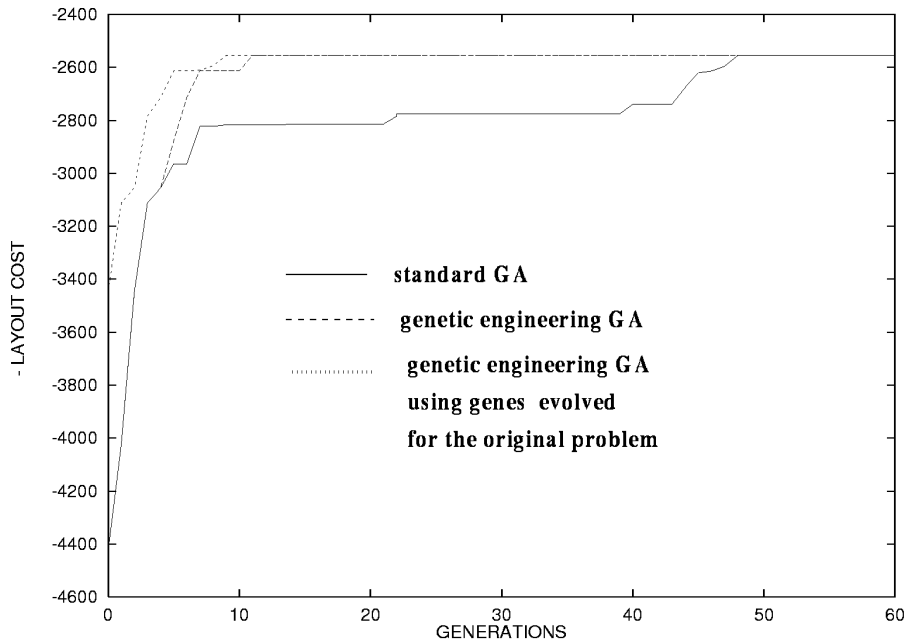


Fig. 12: The best fitness vs. generation number for the standard GA and genetic engineering based GA in modified example. The results are averaged over 10 runs with different initial random seeds, which converge to the best solution found.

retrieved case and the current design, aspects of the retrieved case have to be changed to fit new conditions. Design adaptation is therefore an important step in the application of case-based design.

”Case adaptation can be simply stated as making changes to a recalled case so that it can be used in the current situation. Recognizing what needs to change and how these changes are made are the major considerations. Adapting design cases is more than the surface considerations of making changes to the previous design, it is a design process itself (Maher et al. 1995).”

Automatic adaptation of design cases has been studied by Dave, Schmitt, Faltings & Smith (1994) and Schmitt (1993) in building design. These works shows that the potential for adaptation of a case very much depends on its representation. Every adaptation operation first requires the transformation from a general case representation (e.g. a three-dimensional model, or a file from an architectural drawing program) into a special representation. The adaptation is applied only to this special representation, and the result is then transformed back into the general case representation. Usually, the transformation into a special representation also includes a strong parameter reduction. This reduces the size of the search space and is necessary to keep

the computational complexity within bounds. However, it also often means that some, possibly more desirable, design solutions are excluded.

The adaptation process in case-based design can be seen as dealing with two problems:

1. deciding which features are characteristic for a design and are to be kept, and which features have to be changed; and
2. producing new designs, fulfilling the new requirements, reusing features that have been identified as characteristic, and adapting these features.

By using a specified representation for the case adaptation, the features of the design that can be changed and those that are left unmodified are defined. For example, if the adaptation is based on a representation that uses overall dimensions, then room areas can be adapted by stretching or shrinking parts of the design, but the topologies cannot be modified.

3.2 Basic and evolved representation

In order to use evolutionary systems in case-based design, the designs have to be coded into a genotype suitable for genetic operations. As described in section 3.1, the representation of the design case defines what adaptations are possible. Since our goal is to leave this decision to the evolutionary system, we have to start with a very simple representation that restricts as little as possible the designs that can be described by it. The representations used in the examples in this section are all based on a square grid, and consist of sequences of basic shapes (squares, vectors). To allow for shapes of different complexity, variable-length genotypes are used.

For the evolved genes, a number of basic shapes are composed into a more complex, evolved shape. A single evolved gene therefore can describe a complex shape, and different evolved genes can describe shapes of different complexity.

Figure 4 is redrawn using vectors and shapes to describe this application.

3.3 Genetic engineering application

As mentioned above, the problem of adaptation in case-based design can be split into two parts. The genetic engineering approach described here is a two-stage process.

Genotypes that represent outlines are constructed as sequences of vectors. The concentric circles of Figure 13 represent the search space for genotypes of different length, open arcs indicate that only part of the search space is shown. In Figure 13(a) the basic coding is used. Designs produced by genotypes of length one (the basic genes) are in the centre. The further

away from the centre a design is, the larger is the search space that has to be searched to find it, and the more complex it is. Every time an evolved gene is created the structure of the search space changes. The shape that is created by the evolved gene moves into the centre and all shapes that can be derived from the evolved gene move towards the centre with it. The more instances of the evolved genotype contains, the more it is moved. Figure 13 illustrates this: If the two closed shapes from the fourth circle in Figure 13(a) are identified as particularly successful and an evolved gene is created for them, the search space changes as shown in Figure 13(b). The two closed shapes are now basic building blocks and some shapes from the original fifth circle can now be found in the second circle, and the shape with four squares that previously was on the 14th circle (it requires a genotype of length 14, since the shape cannot be drawn without drawing two lines twice) is now on the fifth circle. Since the introduction of evolved genes increases the size of the alphabet, the size of the circles also grows.

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 search space has to be searched. If, for example, we want to find the window-shaped 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 $6^5 \approx 2^{13}$ possible genotypes has to be searched.

In the adaptation step, it is important to note that the evolutionary system can make use of any or all of the features encoded in the new representation, or none of them. Typically, the features that are usable to solve the new problem are used, while the other will be eliminated from the population sooner or later. But it is also possible, but harder, for the system to create new designs where none of the evolved genes can be used. Solutions that are close to the example designs (in any respect) are easier to find than solutions that are less closely related, while any single feature can be adapted. In other words, the evolved transformation introduces a bias into the second-step search for results that share features with the examples, without restricting the space for possible results.

3.4 Example one: simple shapes

In this example, we show how the idea can be used in the design floor plans (This example is described in more detail in Schnier & Gero 1995).

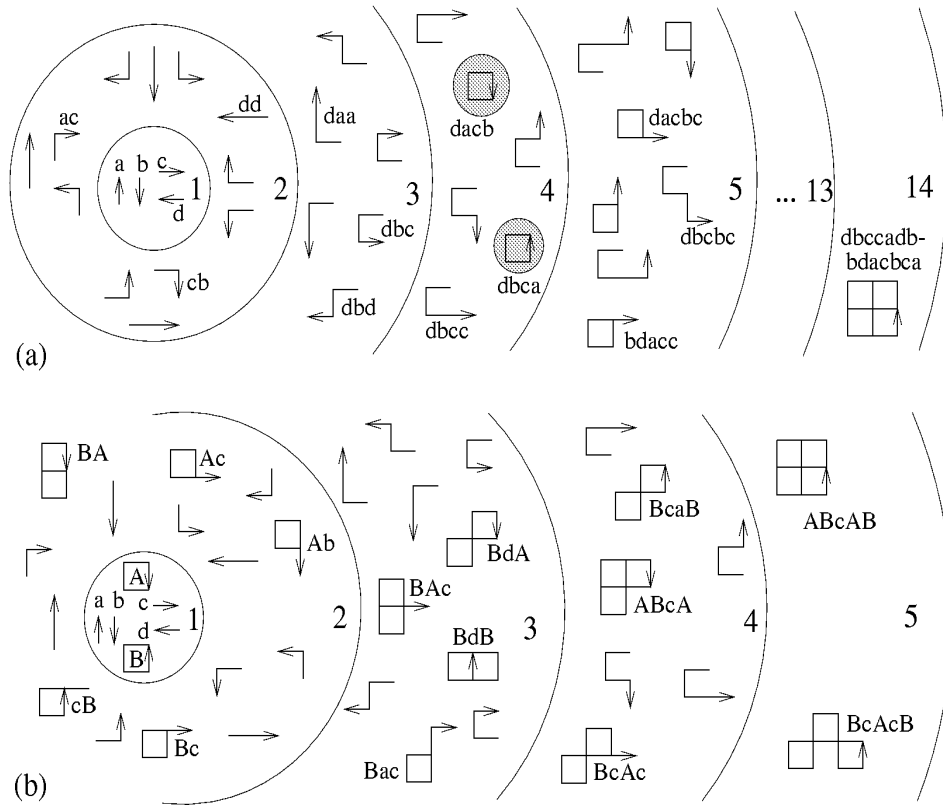


Fig. 13: Example of an evolving representation: (a) original representation and (b) representation with evolved genes. Some of the corresponding genotypes are given, capital letters denote evolved genes. The transformation from phenotype to genotype is not always unique, e.g. the genotypes 'ABc' and 'BAC' produce the same phenotype. Arc segments indicate that only part of the space is shown.

As 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 14.

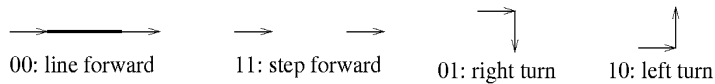


Fig. 14: Basic coding, arrows show pen position and current direction before and after the gene is drawn.

This coding is able to represent any kind of two dimensional shape composed of horizontal and vertical straight lines, and has been used for building elevations and floor plans. An extension to three dimensions and different shapes is straight forward.

Evolving the representation i

The example discussed here uses the coding above to create floor plans, the two floor plans shown in Figure 15 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.

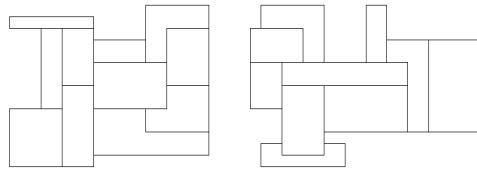


Fig. 15: Room plans used as example cases.

To create evolved genes, successful combinations of genes in the population have to be identified. Ideally, this has to be any number and combination of genes in the genotypes. However, any composition of more than two genes can be constructed from a number of compositions of two genes. For example to combine three genes, in a first step an evolved gene is created that combines two of the original genes, and in the next step this new evolved gene is combined with the third original gene. An additional feature of the basic representation used here allows further simplification: the further apart two genes are in the genotype, the less likely they are to depend on each other in the fitness. Therefore, we have only to 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 16(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 16(b) shows how an individual is composed from six evolved genes.

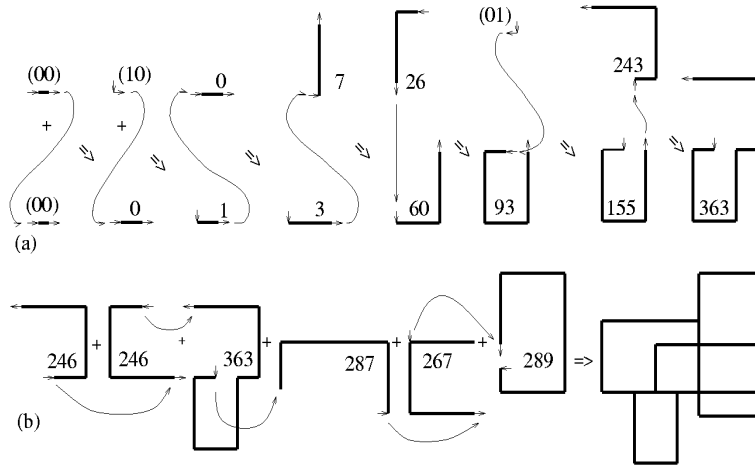


Fig. 16: Evolved representation: (a) part of the hierarchical composition of the evolved gene 363 and (b) composition of the individual with the genotype (289 267 287 363 246 246).

Using the representation

After a representation based on the examples in Figure 15 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;
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 17(a) shows the result of one run, after 150000 cross-overs were performed. The population size was 1000 individuals, and 320 evolved genes created from the examples in Figure 15 have been used. They 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, with the exception that individuals that are rotated copies of already drawn individuals have been omitted. Figure 17(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.

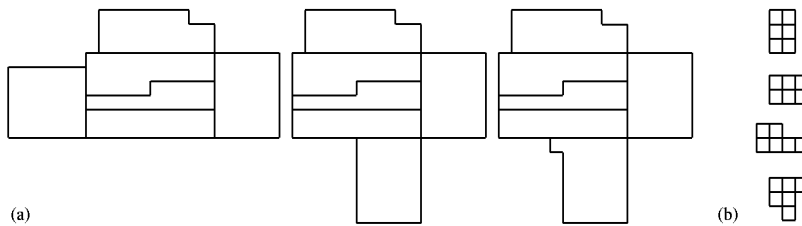


Fig. 17: New floor plans, using coding knowledge from the example cases (see text for fitness requirements). (a) using 320 evolved genes, (b) using only the basic genes.

As described above, every evolved gene represents a sequence of basic genes, and the length of this sequence directly corresponds to the amount of information the evolved gene contains about the example case. An analysis of the lengths of genes used in the genotypes of the final population can therefore show how much information from the example case is used in the new results. Figure 18 shows a histogram for the example presented here. The peak at the left contains the four basic genes, which together are used about 4600 times in the final population of 1000 individuals. Some of these occurrences are left and right turns at the beginning of the genotypes, due to the fact that while identical individuals are not permitted in the population, adding a turn at the front of the code creates a new individual with the same fitness as the original code. More than two thirds of the 15400 genes used in the genotypes of the final population are evolved genes, encoding between 2 and 25 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.

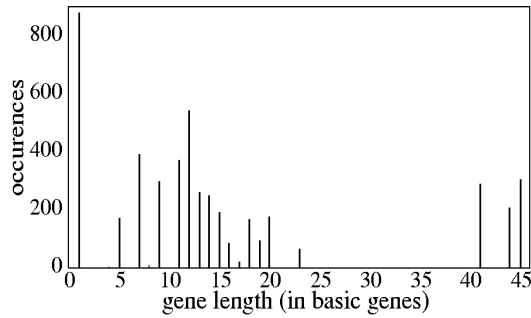


Fig. 18: Histogram of the lengths of evolved genes (in basic genes) used by the new results

3.5 Example two: learning style features

In this second example the genetic engineering approach is used to learn style features from a set of example floor plans, in this case from floor plans of Frank Lloyd Wright houses (for a more detailed description of this application, see Schnier & Gero 1996). An analysis of the style used shows that some of the style features are not simply products of outlines, but are linked to the functions of the rooms. To allow the system to pick up those style features as well, the basic coding is extended to allow it to represent lines of different colours, and the functions of the rooms are encoded in colours in the examples given to the system.

In the examples of the Frank Lloyd Wright prairie houses, lines that enclose living spaces use a different line type from lines that enclose service spaces or porches, and the fire place has a type of its own. Since only the main floor is considered here, no bedroom zones occur in the designs. Since the basic coding only allows horizontal or vertical lines, the diagonal lines at the wings of the Henderson house have been changed into a stepped shape. Figure 19 shows the floor plans used.

Figure 20 shows examples of evolved genes created from the four example designs. Shown are some of the last evolved genes created from the examples. Clearly visible are the shapes of rooms, and the different line types, associated with the different functions. Two of the evolved genes shown (310 and 318) have the fireplace as part of the line-drawing they code for (in this case from the Henderson house).

Using the evolved representation

In the second phase, the representations evolved from the example cases are used to create new designs that show similarities in style to the example cases. For this, a standard evolutionary system is used, with the set of basic and evolved genes used in the coding. The evolved coding, as shown in Figure 20, captures information about shape and function of parts of the example

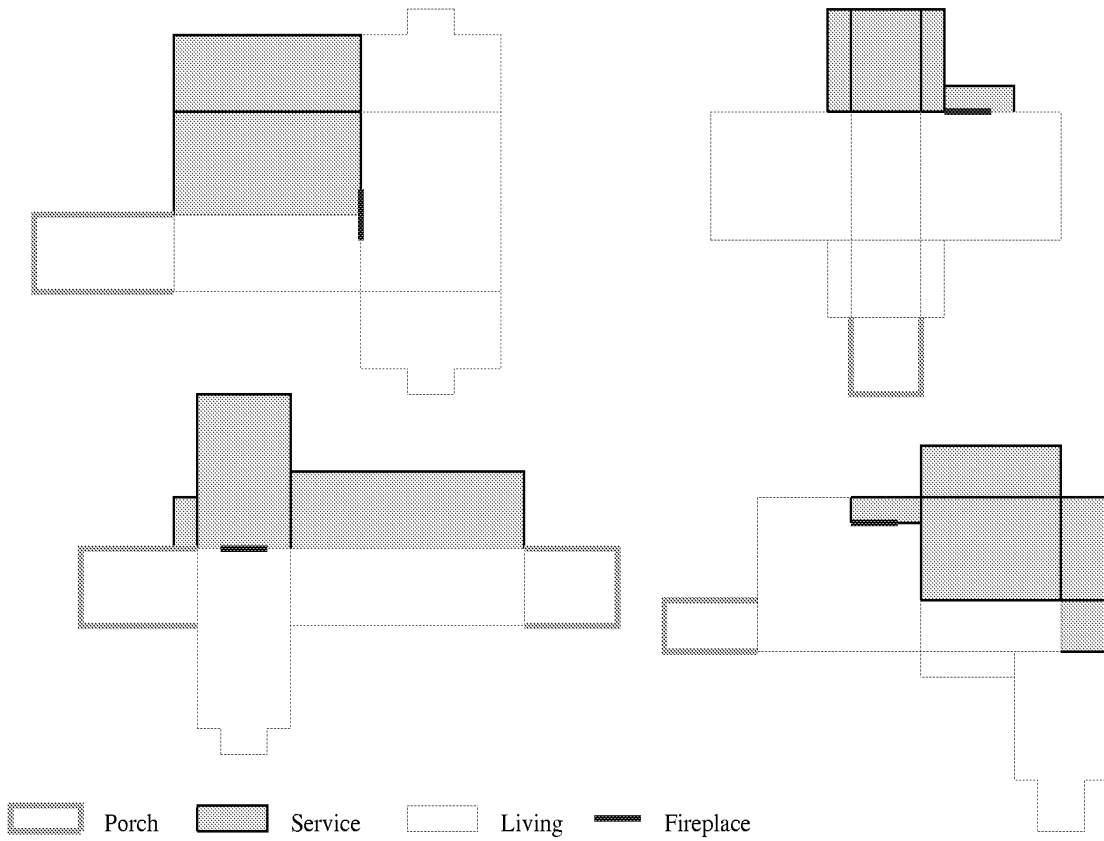


Fig. 19: Frank Lloyd Wright houses used to create the evolved coding: Henderson house (top left), Martin house (top right), Baker house (bottom left), Thomas house (bottom right).

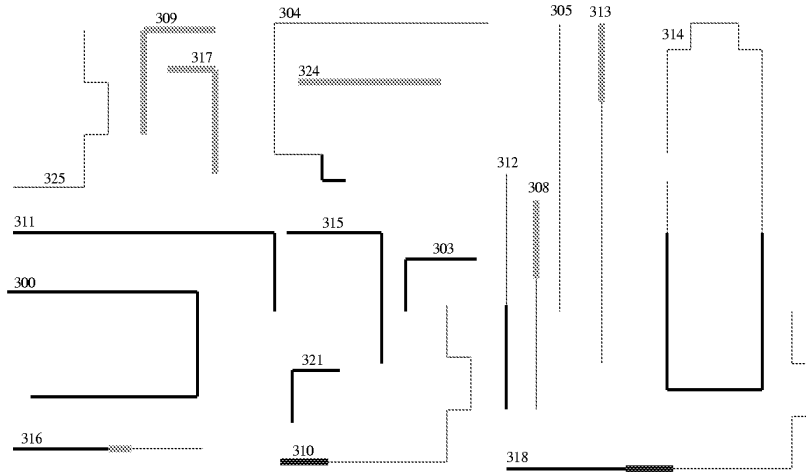


Fig. 20: Examples of evolved genes created from the example designs shown in Figure 19.

designs. However, the way these parts are assembled to create new designs is only influenced by the fitness function that evaluates new designs. This means that any gene that codes for a room of a certain type, for example, has no influence on what other room is next to it; and there is nothing preventing a design from using two evolved genes that each include a fireplace. As a result, topological constraints are not automatically satisfied by using the evolved coding. If designs created by the evolutionary system are to fulfill topological constraints, they have to be included in the fitness function (see Section 3.6 for how it is possible to integrate more topology information into the evolved coding). Frank Lloyd Wright's prairie houses follow a number of topological constraints, and they all have to be made part of the fitness. For the results presented here, fourteen different aspects influence the fitness. In the following list shows some fitnesses:

- one porch, size between 9 and 12 units
- porch connected to living area, and not connected to service area
- two to four rooms in the service area, total size between 45 and 60 units
- two to four rooms in the living area, total size between 55 and 70 units
- only one service and one living area, i.e. all rooms of that type are connected
- one fireplace, two units length, between living and service area
- no 'dead ends', i.e. lines that do not enclose any room.

For a human designer, it is relatively easy to find designs that fulfill all the fitness conditions. For a standard evolutionary system, the fitnesses can be integrated into a single fitness. This could for example be done by calculating a value between 0 and 1 for every individual fitness, and adding or multiplying them into a single fitness value. Unfortunately, by integrating all fitnesses into one value, the information about what fitness conditions are fulfilled and what conditions are not is lost to the system. As a result, the system ends up converging towards a population that is good in some respects while individuals good in different aspects are lost. Even after a very high number of individuals have been produced, the system is not able to find satisfying solutions.

A better way to handle a high number of individuals is therefore to utilize 'Pareto optimization' (see for example Radford & Gero 1988), where only partial rankings between individuals are established. As an additional measure to prevent convergence, a 'niching' Pareto algorithm is used (Horn & Nafpliotis 1993). One of the effects of Pareto selection is that the population grows over time, for example, in one of the runs presented below, the population grew from 500 to 1581 individuals.

Results

Two runs were done using a set of 326 individuals created from the floor plans in Figure 19.

Run 1 ran for 60677 loops, each loop creating two offspring individuals. The initial population was 1000 individuals, the final population consisted of 1652 individuals. From some 120000 produced and tested individuals, 14359 individuals were good enough to be introduced into the population.

Run 2 ran for 99207 loops, the population grew from 500 to 1581 individuals. Of the nearly 200000 individuals produced, 12502 made it into the population. Again, all 326 evolved genes were used.

The first result from Run 1, Figure 21(a), has a perfect fitness. The fitness function does not check if the fireplaces are straight, therefore a corner fireplace could result. Since none of the floor plans in the example drawing have a corner fireplace, this feature cannot have been part of the evolved coding. It therefore must be coded in basic genes. The second-best result from Run 1, Figure 21(b), has a penalty due to one segment of 'dead end' close to the fireplace, but fulfills all other fitness criteria.

Both results of Run 2, Figures 21(c) and (d), have perfect fitnesses. Again, the system has taken advantage of a weakness in the fitness function, that allowed it to put the porch inside the living space.

Figure 22 shows how one of the results (the second of Run 1, see Figure 21(b)) can be extended into three dimensions by a graphic artist. The resulting house is similar to the Thomas house.

3.6 Discussion

The two examples described in this section show how the genetic engineering approach can be used to extract information from one problem solution and reuse it in a different, but related environment. It does this without the need for a specification of what features are to be extracted and without restricting the set of possible solutions when the new problem is solved. In the first example, the knowledge gained can be described as 'long straight lines, rectangular shapes and certain more prominent, complex shapes. How much this knowledge helps in creating new layouts can be seen in Figure 17. If only the basic coding is used without evolving coding to solve the problem used in the first example, the system still finds solutions with six rooms, but all of them have only unit size (see Figure 17(b)). Since any lines that do not totally enclose a room are penalized by the fitness function, the evolutionary system has to create the next larger room size in "one step" from the small rooms. In other words, the first local maximum found in the fitness function is a layout with six unit-sized rooms, and the distance to the next local maximum is too far to be found easily by either mutation or cross-

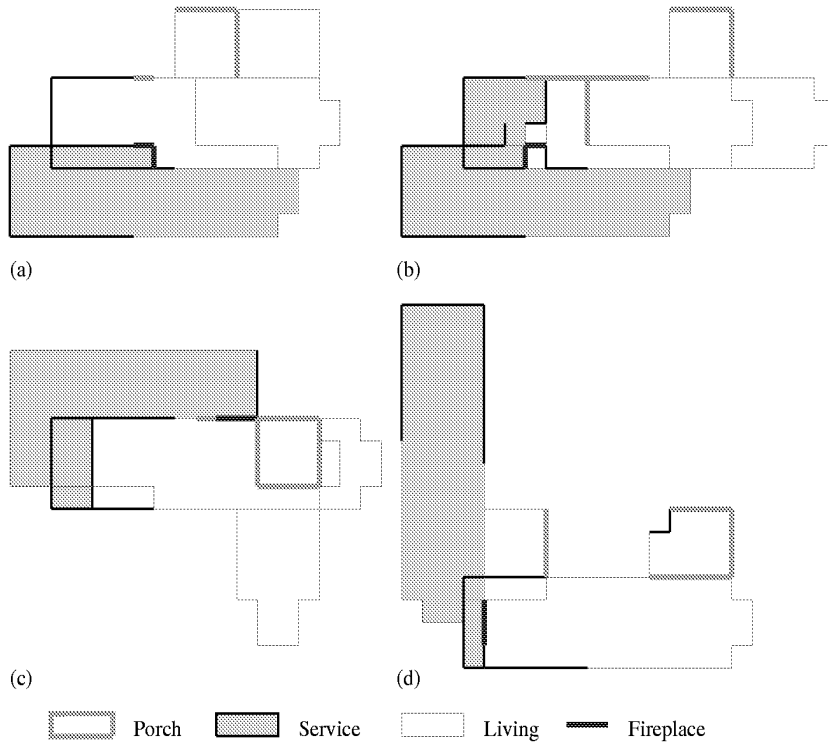


Fig. 21: Floorplans created using the evolved genes from the example designs shown in Figure 19, (a) and (b) initial population 1000 individuals, (c) and (d) initial population 500 individuals.

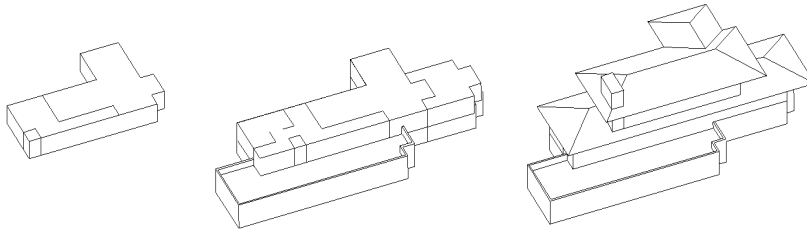


Fig. 22: Floorplan from Figure 11 (b) manually extended into three dimensions; shown are bedroom level, main floor level, and roof view.

over. The “granularity of possible solutions” seems to be too fine compared with the “granularity of solutions rewarded by the fitness function”. It is important to see the difference between this fitness function and the one used to create the evolved coding. The latter is smooth, and any improvement in the code is rewarded appropriately. If the fitness that was used to create new layouts had been used to evolve a coding, the system would have been no more able to find larger rooms than without evolving coding. This highlights the point that the gain is not the evolving coding, but the evolved coding when used in similar applications. In the second example, an expanded basic coding was required to represent additional, semantic information in the examples. This enabled the system to integrate more knowledge into the evolved representation. However, no aspects of the topology are coded in the evolved coding. This results in the fact that some features of the example design that the system could have learned have to be added as fitness. It also shows in the results: shapes that have been outer walls in the original drawings are used in the inside (e.g. the stepped line separating the right two parts of the living room in the second example of Figure 21(a)). One way to improve the ‘knowledge-content’ of the evolved coding would be to use a larger number of line types. Different line types could be used depending on the functions of both of the rooms a wall separates, and again different line types for outer walls. This way, knowledge that for example three out of four sides of the porch are outer walls, could be integrated into the evolved coding. To realize this, no changes other than changing the parameter for the number of line types used and modifying the example drawings are necessary in the first step. In the second step, a fitness function would have to be added that checks if lines are used in a correct context, while other fitness criteria would become unnecessary.

1. Acknowledgments

This work is funded by the Australian Research Council.

References

- Anderson, W. F. (1995). Gene therapy, *Scientific American* **273**(3): 96–98B.
Collins, J. & Coulson, A. (1987). Molecular sequence comparison and alignment, *Nucleic Acid and Protein Sequence Analysis: A Practical Approach*, IRL Press, Washington DC, pp. 323–358.

- Dave, B., Schmitt, G., Faltings, B. & Smith, I. (1994). Case based design in architecture, in J. S. Gero & F. Sudweeks (eds), *Artificial Intelligence in Design '94*, Kluwer Academic Publishers, Dordrecht, pp. 145–162.
- Gero, J. & Kazakov, V. (1995). Evolving building blocks for genetic algorithms using genetic engineering, *Proceedings of the IEEE Conference on Evolutionary Computing*, pp. 340–345.
- Gero, J. & Kazakov, V. (1996). Evolving design genes in space layout planning problems, Submitted to *Artificial Intelligence in Engineering*.
- Horn, J. & Nafpliotis, N. (1993). Multiobjective optimization using the niched pareto genetic algorithm, *Technical Report 93005*, Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois at Urbana-Champaign.
- Karlin, S., Dembo, A. & Kawabata, T. (1990). Statistical composition of the high-scoring segments from molecular sequences, *Annals of Statistics* **18**: 571–581.
- 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 U.S.A.*, Vol. 87, pp. 5509–5513.
- Liggett, R. (1985). Optimal spatial arrangement as a quadratic assignment problem, in J. Gero (ed.), *Design Optimization*, Academic Press, New York, pp. 1–40.
- Maher, M. L., Balachandran, M. B. & Zhang, D. (1995). *Case-Based Reasoning in Design*, Lawrence Erlbaum, Hillsdale, NJ.
- Needleman, S. & 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.
- Plucknett, D. L. & Winkelmann, D. L. (1995). Technology for sustainable agriculture, *Scientific American* **273**(3): 148–152.
- Radford, A. D. & Gero, J. S. (1988). *Design by Optimization in Architecture and Building*, Van Nostrand Reinhold, New York.
- Sankoff, D. & Kruskal, J. (eds) (1983). *Time Warps, String and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, MA.
- Schmitt, G. N. (1993). Case-based reasoning in an integrated design and construction system, *Management of Information Technology for Construction*, World Scientific Publishing, Singapore, pp. 453–465.
- Schnier, T. & Gero, J. S. (1995). Learning representations for evolutionary computation, *Australian Joint Conference on Artificial Intelligence AI '95*, pp. 387–394.

- Schnier, T. & Gero, J. S. (1996). Learning genetic representations as alternative to hand-coded shape grammars, *in* J. S. Gero & F. Sudweeks (eds), *Artificial Intelligence in Design '96*, Kluwer, Dordrecht, pp. 39–57.
- Schuler, G., Altschul, S.F. & D.J, L. (1991). A workbench for multiple alignment construction and analysis, *PROTEINS:Structure, Function, and Genetics* **9**: 180–190.

⁰this is a copy of the following: Gero, J. S., Kazakov, V. 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.