Evolving representations of design cases and their use in creative design

John S. Gero and Thorsten Schnier
Key Centre of Design Computing and Cognition
Department of Architectural and Design Science
University of Sydney NSW 2006 Australia

Abstract: In case-based design, the adaptation of a design case to new design requirements plays an important role. If it is sufficient to adapt a predefined set of design parameters, the task is easily automated. If, however, more far-reaching, creative changes are required, current systems provide only limited success. This paper describes an approach to creative design adaptation based on the notion of creativity as 'goal oriented shift of focus of a search process'. An evolving representation is used to restructure the search space so that designs similar to the example case lie in the focus of the search. This focus is than used as a starting point to generate new designs.

1. Introduction

The first step in any attempt to generate computational design creativity is to identify what characterizes creativity in design. Creativity itself is difficult to define, and, like definitions of 'life' for example, it is often possible to find examples that satisfy any specific definition, but still contradict one's intuition about the meaning of 'creativity'. However, some models exist that allow the definition of properties that are necessary conditions, if only for certain 'kinds' of creativity.

This section present some aspects of modelling creativity. Particular attention is paid to the special requirements and restrictions imposed because the the creative process is done inside a computational process, especially 'closed world' considerations. Also important for design computation is the representations of designs.

Models of Creativity

Creative design has been characterized in computational terms as `` that design activity which occurs when a new variable is introduced into the design" (Gero 1994). This is opposed to 'routine design', where ``knowledge about variables, objectives expressed in terms of those variables, constraints expressed in terms of those variables and the processes needed to find values for those variables, are all known *a priori*". A third alternative, 'innovative design', occurs when no new variables are introduced, but one or more variables are used with values outside the usual scope.

Creative design according to this definition requires the addition of one or more

variables, but it does not require that the other variables remain unchanged. There are therefore two cases that can be distinguished:

- One or more variables are added, while the other variables remain part of the design (additive case)

- One or more variables are added, while at the same time one or more other variables are deleted from the design (substitutive case).

The two cases can best be visualized in terms of design space. If a design has *n* variables, then these variables can be seen as coordinates in an *n* dimensional space. Every possible design can thus be mapped onto a point in this space, the design space.

Figure 1 illustrates the concepts described above in a simplified, schematic example. The original search space with two dimensions is illustrated in Figure 1(a). Figure 1(b) shows an instance of 'innovative design' (as defined above), where no new variables were introduced, but the existing ones use values beyond the usual range. Figure 1(c) and (d) illustrate the two possibilities of 'creative design': moving the state space by substituting variables (c) and expanding the state space by adding variables (d).

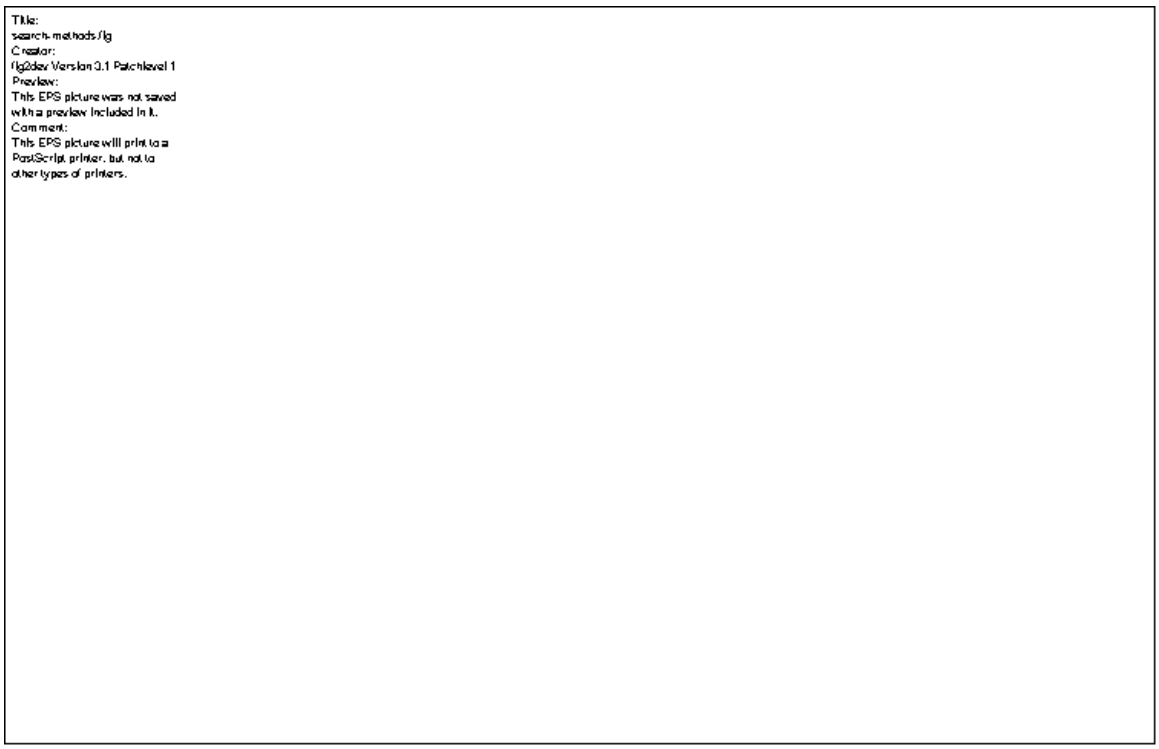

Figure 1 Changing state space: (a) original state space, (b) state space with expanded borders, (c) moved state space due to variable substitution, (d) expanded state space due to variable addition.

With respect to the addition of variables, again two cases are distinguished:

- Homogeneous addition: the variable added is of the same type as other variables already in the design (for example transform an irregular pentagon into a hexagon

by adding a side, see figure2(a)).

- Heterogeneous addition: the variable added is of a different type from those already in the design (for example changing one side of a triangle into a section of a circle, the variable added is the radius of the circle, see figure 2(b)).

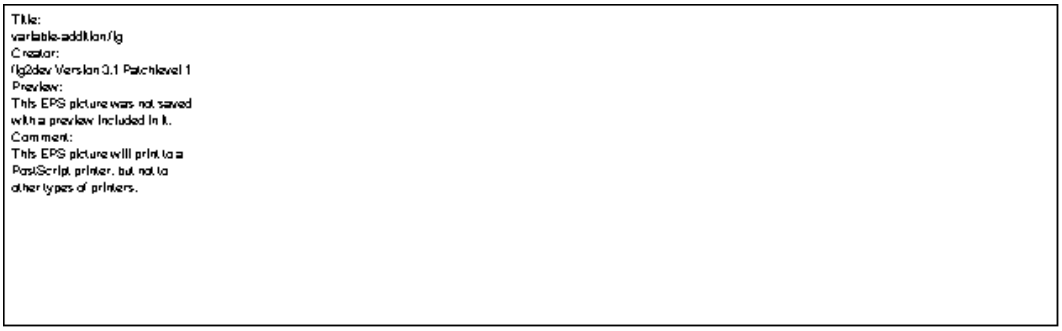

Figure 2 Variable addition: (a) homogeneous addition, (b) heterogeneous addition

.

In computational terms, routine design can also be seen as 'search', (sometimes also called 'exploitation') in so far as a certain, predefined search space (the *n*-dimensional design space) is searched for a design solution. For creative design, where the search proceeds outside the boundaries of a predefined search space, the term 'exploration' can be used (see Gero 1994).

Creativity and closed worlds

Since the subject of this paper is creativity inside a computational process, attention has to be paid to the limitations to which computer programs are generally subject. In this context, three basic *limitations* are of interest:

1. The size of memory available to the program is limited, this means that the total number of different states the program can assume is limited (finite state machine).

2. The computing power available to run the program is limited, this means that the number of different states a program can assume in an acceptable time is limited, and usually much less than the number of different possible states.

3. The time allocated to create the program is limited. This means that any program can take into account only a limited set of relations.

How do these general limitations apply to computational design creativity? There are three main *implications*:

1. Due to Limitation 1, the total set of different designs that a program can generate

is limited and defined a priori. However, this does not mean that the set of possible designs is also known a priori. As Langton (1988) observes, Turing's halting theorem can be expanded to show that

> *``it is impossible in the general case to determine any nontrivial property of the future behaviour of a sufficiently powerful computer from a mere inspection of its program and initial state alone."*

Depending on the complexity of the representation, it might therefore be impossible to deduce the set of possible designs from the set of states the representation can assume, and it might also be impossible to decide if a given design is part of the total space of possible designs or not (without enumerating all possible states).

2. Due to Limitation 2, the set of different designs that a program can produce and evaluate in an acceptable time-frame is limited, and is usually much less than the number of possible designs. This means that the space searched usually has to be much smaller than the space of possible designs.

3. Due to Limitations 2 and 3, the evaluation of the design can take into account only a certain, limited set of interactions between a design and its environment. For example, individuals in an artificial life application can develop 'vision' only if a) the individuals have access to some kind of optical sensory organs, and b) in every time-step of the evolution, the appropriate input signal for every instance of those organs is calculated (see for example Yaeger 1994). However, even this would not allow for individuals developing, for example, flight.

Together, these implications mean that computational design processes can work only inside a so-called 'closed world'. This can be seen as a contradiction of the definition of 'creativity' above, since the total set of possible designs can always be seen as a large, unchanging search space. However, it seems that similar limitations hold for human designers. For example, as stated in Maher et al. (1989), a very good knowledge both in breadth and in depth about a field is a necessary condition for creativity in humans. This suggests that human creativity also may work in a (very large) closed world, restricted by the knowledge of the person. And humans are also restricted by the number of design alternatives they can consider in a limited time. The way human designers cope with their 'limited computational power' is that they *focus* onto a certain subset of design variables. If this focus remains unchanged, the process still would be called 'routine design'. Creativity, then, seems to require the ability to move this focus into different areas of the design space, and to do this not randomly, but directed towards improved designs.

In the light of this, a necessary condition for a creative design process can be defined as

> *the ability to perform goal-oriented shifts of the focus of the search activity*

whereas non-creative processes have to search a fixed sub space or the whole search space at once.

This also means that the same design can be considered creative or not, depending on the way it was created. Taken together, there are a number of *methods* which design processes can use:

1. A process can search only a small search space, accepting the outcome of this search, even if it represents only a local solution, and better designs lie outside the search space. For example, the search space can be restricted to designs where methods to optimize them analytically are known. This method would be categorized as 'routine design'.

2. Using domain knowledge to create a search space that is known to contain the desired design. This could be the case in a situation where, say, theoretical analysis shows that all designs outside a certain space give results that violate one or more of the design restrictions. This is a second alternative for 'routine design'.

3. Searching the desired design in a large space without heuristics. This could either use a random search or try to enumerate all possible designs.

4. Using domain knowledge to guide the search for the desired design in a very large search space. Search strategies such as 'hill-climbing' and 'evolutionary search' fall under this criterion.

5. Focussing onto a subspace, but with the ability to use domain knowledge to change the focus. This is the 'creative design' case.

Figure 3 illustrates these five possibilities. In practice, a combination of the different methods will usually be applied. The restriction of the total search-space in Methods 3 to 5 will be imposed not by the total set of computationally possible designs (Implication 1), but by restrictions due to the limited number of interactions between design and environment that can be considered (Implication 3), and by additional search-space restriction as in Methods 1 and 2 above.

Figure 3: Methods to search a large search space (black dot: desired final design if its existence were known; broken line: total search space, bounded by restriction 1; continuous line: search space searched): (a) focus on a subspace, possibly excluding desired design; (b) use domain knowledge to set focus to include desired design; (c) search whole search space; (d) search in whole search space using domain knowledge; (e) focus on subspace, but move focus

One way to describe the change in focus is in terms of adding or substituting design variables, as described above. This notion is appropriate when representations are used that allow the identification of the individual design variables, for example design prototypes. For other applications, like those based on evolutionary systems, only the more general description of focus change can be used.

Representation of designs

In design computing, every design is represented in the computer in a special, usually problem-dependent representation. One example of a universal representation for designs is the design prototype, as specified in gero90. An important aspect of that definition is the distinction between state spaces for function, behaviour, and structure. Every design has a representation in all of these state spaces. Direct transformations are only possible between structure and behaviour, and between behaviour and function. This is illustrated in Figure 4.

Figure 4: The three subspaces of the design space: function (F), behaviour (B), structure (S) and the possible direct transformations between them.

Through the relations between these spaces, changes in one of them usually influences the other spaces as well. In the usual case, the modifications during a design are done in the structure space, while the quality of a design (the 'fitness' of the design in terms of evolutionary systems, see below) is assessed in the behaviour space.

If a process wants to add variables to a design, it has to modify the representation. A basic requirement for creative design computing is therefore that representations can easily be modified. Representations that provide the necessary flexibility without the need of special high-level knowledge can be found for example in the area of evolutionary systems.

Genetic engineering

One foundation for evolving representations is genetic engineering. It is derived from genetic engineering notions related to the human intervention in the genetics of natural organisms. When there is a group of phenotypes which exhibit particularly apposite performance in a fitness we can hypothesize that there is a unique cause for it and that that unique cause can be directly related to the organism's genes which appear in a structured form in its genotype. Genetic engineering is concerned with locating those genes which produce the fitness under consideration and in modifying those genes in some appropriate manner. This is normally done in a stochastic process where we concentrate on populations rather than on individuals.

Organisms which exhibit high performance in the fitness of interest are segregated from those organisms which exhibit low performance in that fitness. This bifurcates the population into two groups. The genotypes of the former organisms are analysed to determine whether they exhibit common characteristics which are not exhibited by the organisms in the latter group, Figure 5. If the genotypes of these two groups are disjunctive, the gene groups which are unique to the high performing group are isolated on the basis that they are responsible for the high performance in the fitness of interest. These genes are grouped together and become a single evolved gene. In natural genetic engineering these isolated or evolved genes are the putative cause of positive fitnesses. If they are associated with positive fitnesses they are reused. It is this latter purpose which maps on to our area of interest. (In natural genetic engineering it is also interesting to look for gene groups responsible for negative fitnesses. If these gene groups can be isolated they can be substituted for by 'high fitness' gene groups which do not generate the negative fitness.)

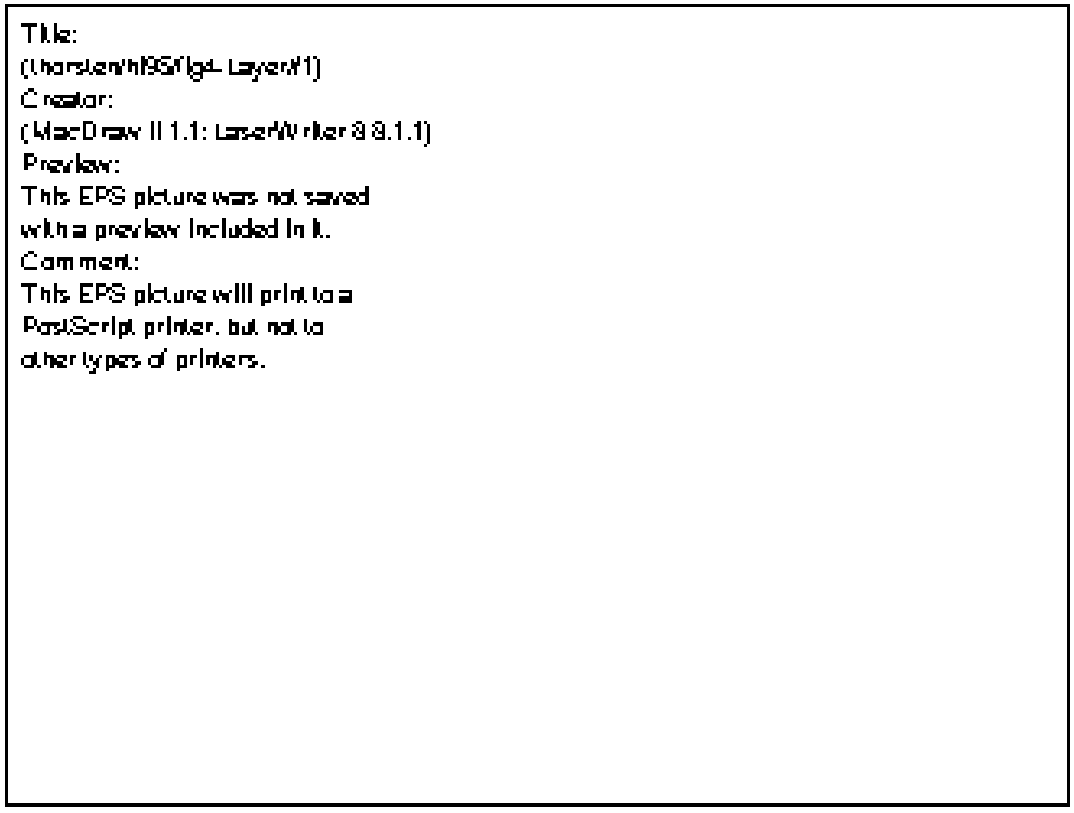

Figure 5: The genotypes of the 'high fitness' members of the population all exhibit gene combinations, X*, which are not exhibited by the genotypes of the 'low fitness' members. These 'high fitness' gene combinations are the ones of interest in genetic engineering.*

2. Evolving Representations in Case Based Design

In the previous section, it was shown that creative design can be modelled as a search process that is able to create and shift the focus of the search inside a larger search space. This section shows how this process can be used to support one important step in case-based design: the adaptation of a case to new design requirements.

Case-based design

Case-based reasoning has been introduced in design to allow the reuse of knowledge from design cases, rather than having to start from first principles or compiled knowledge with every new design (Maher et al. 1995). Contrary to knowledge-based design systems (Coyne et al. 1990), the expert knowledge is not stored as an explicit rule set, but is implicit in a database of previous design cases. Since requirements and environmental conditions will usually vary between the retrieved and the actual case, the design has to be changed to fit the 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 e.g. by Dave et al. (1994) in building design. They use two different adaptation operations: dimensional modification and topological modification. Dimensional modification changes the dimensions of the design elements without changing their number. It uses a subset of parameters that is created from the original three dimensional model by a process called data reduction. If dimensional modification is not sufficient to adapt the case to the new requirements, the topology of the model has to be modified as well. In Schmitt (1993), the author investigates the use of string grammars to automate topological modification. Here, the most interesting aspects are extracted from the design case and then are subject to modifications. Other representations used are case-specific rules and a wall representation developed by Flemming et al. (1988).

Dave et al.'s (1994) work 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) 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 desirable design solutions are excluded.

Obviously, there are two contradictory goals: the representation has to create a search space that on one hand is as small as possible, but that on the other hand does not exclude any possibly desirable design solutions. This is where the notion of 'focussing', as introduced in section 1.2, is useful. If a process can create a 'focus' inside the search space, then it is able to find good solutions based on the design case with reasonable computational effort, while no possible design solution is excluded.

The system described here is based on evolutionary systems, the focussing is done by allowing the coding in the system to evolve.

Evolutionary Systems

The phrase 'evolutionary systems' describes a collection of algorithms that are more or less based on Charles Darwin's theory of evolution. The algorithms can be described as a search in a multidimensional space. Associated with every point in the search space is a fitness value (or a vector of values). The closer a point is to the search goal, the higher is the fitness. A transformation is defined that maps every point of the

search space into a description of that point (e.g. a fixed-length bit string). This transformation corresponds to the mapping from the physical instance ('phenotype') to its genetic code ('genotype'). Starting with an initial random population of individuals (genotypes, associated fitness and process information), the following cycle is repeatedly executed:

- some good individuals are selected;
- new individuals are created, somehow based on the genotypes of the individuals selected in the previous step. This corresponds to the notion of 'replication with errors', some typical operators used are mutation and cross-over; and
- some poorly fitted individuals are deleted, corresponding to the notion of 'survival of the fittest'.

Figure 6 shows a flowchart of a general evolutionary system. For overviews of evolutionary systems, see for example (Spears et al. 1993, Fogel 1994, Bäck et al. 1991).

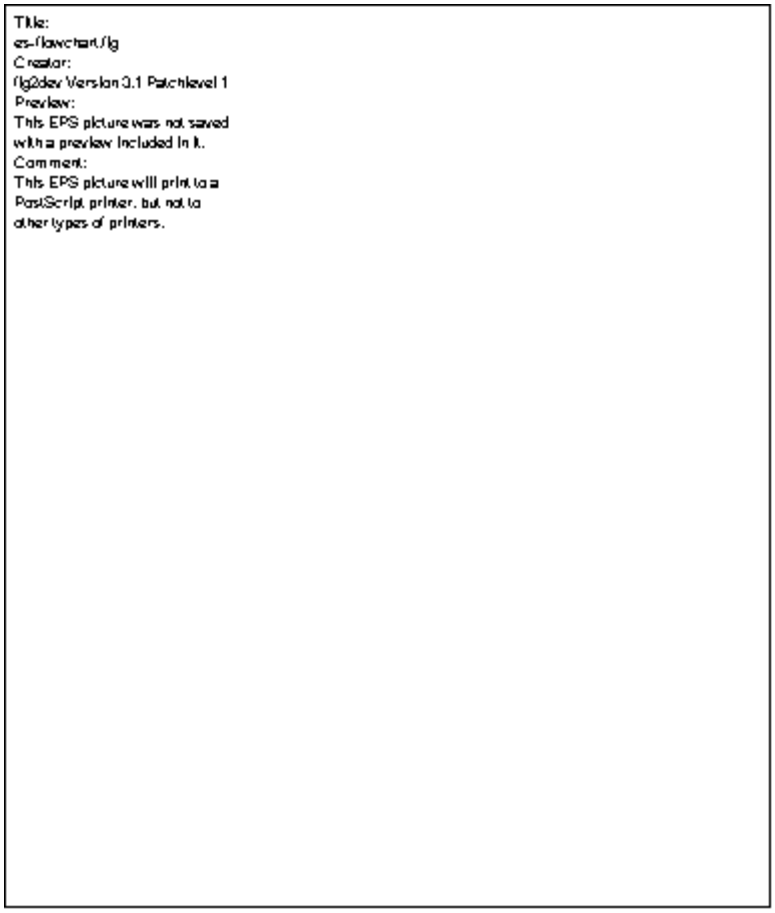

Figure 6: Flowchart of a general evolutionary system

Evolving representation

In evolutionary systems, the search space is defined by the coding of the genotype. This implies that creating and shifting a focus requires a change of this coding, and consequently a change of the transformation from genotype to phenotype. To be able to do this without having to introduce high-level knowledge, an analogy from nature is used.

Optimization in an unknown and/or changing environment is not the only interesting feature in evolution in nature. Evolution also shows strong self-organization into more and more complex and abstract representations. There is no information in the genotype, for example, for the positions of every single cell, or for every single connection of neurons in the brain. Instead, the genetic code only gives instructions for the pattern of growth, or the overall shape of organs, etc. This change in representation allows for a drastic reduction in the size of genetic code, and for less sensibility against disruptions of this code by mutations. Without this developing representation, higher level organisms would never have evolved.

The model for an evolving representation developed in this work follows this observation. Instead of using heuristics or other forms of domain knowledge to change the coding, it uses a bottom-up approach where the coding of the genotype gradually evolves, from a simple, basic coding towards more and more complex codings.

The starting point in the development of a system that creates and makes use of an evolving coding is a standard evolutionary system. The first step is to create a population of randomly created individuals. The coding of these individuals, the 'basic' genes, is chosen to be very low-level, putting as little domain knowledge into the coding as possible, and making sure not to exclude any interesting part of the search space (that is the coding does not introduce any restrictions in the space of designs that can be searched other than Limitation 1). The individuals are then subjected to the cycle of replication with errors and survival of the fittest. But at the same time, an additional operation screens the population, identifying particularly successful combinations of genes. For every such gene combination, a new, 'evolved' gene is created that represents this combination, and is introduced into the population. Figure 7 shows in a flowchart how the general evolutionary system is expanded into a system with evolving coding.

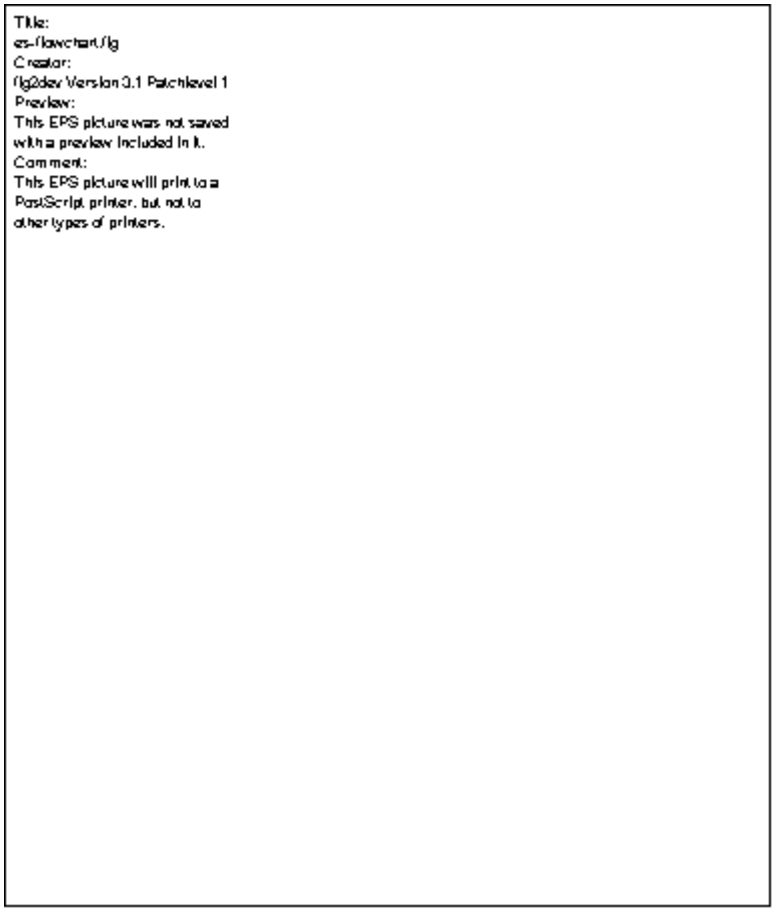

Figure 7: Extensions of the general evolutionary system made for evolving coding

In the first few cycles, the evolved genes will be composed from basic genes, but in later cycles most evolved genes will represent combinations of other, lower level evolved genes, or combinations of those with basic genes. This growing hierarchy of representations gives rise to a more and more complex and abstract coding that is increasingly adapted to the application. In other words, the process gradually collects application specific knowledge and codes it into the representation, rather than being coded into it by the user in the first place.

What does this mean in terms of search space? The length of the genotype is only restricted by the size of the computer memory (Limitation 1 in Section 1.2). The search space is extremely large with respect to the number of states that can be evaluated (Limitation 2), and can therefore be seen as having infinite size. However, since the alphabet used for genotypes is finite at any state during the evolution, the set of possible designs that can be defined by a genotype of a certain length is limited. The search space can therefore be illustrated by an (infinite) number of concentric circles, each defining the space of designs that can be defined by a genotype of a certain length. The inner circle contains the genotypes of length one, i.e the basic building blocks. The further away a design (or part of design) is from the centre, the more difficult it is to find by means of search. Every time an evolved gene is created, the structure of the search space is changed. The state of the new gene in the search space is moved into the centre, all design states in the next circle that can be derived

from that state are moved into the second circle, and so on. Figure 8 illustrates this: the original search space is illustrated in Figure 8(a), with the four basic building blocks in the centre. The building blocks code for vectors of one unit length with the directions up, down, left and right. The arrow points to the starting point of the next element drawn (if any). The second circle shows all designs that can be derived from genes of length two (i.e., using two building blocks). The other circles give some examples of designs using genotypes of length three, four and five. If now the two closed shapes in the fourth circle are identified as particularly successful and an evolved gene is introduced for both of them, the search space changes as shown in Figure 8(b): the squares are now basic building blocks, and the shapes on the fifth circle that are derived from the squares can now be found in the second circle. The more evolved genes a design state involves, the more it is moved towards the centre. For example, the shape with the four squares that is now on the fifth circle (i.e. can be constructed from genotypes of length five) would have been on the fourteenth circle before, since it requires fourteen lines, because the shape cannot be drawn without drawing two lines twice. Since the introduction of a new gene increases the size of the alphabet of the coding, the sizes of the circles also grows.

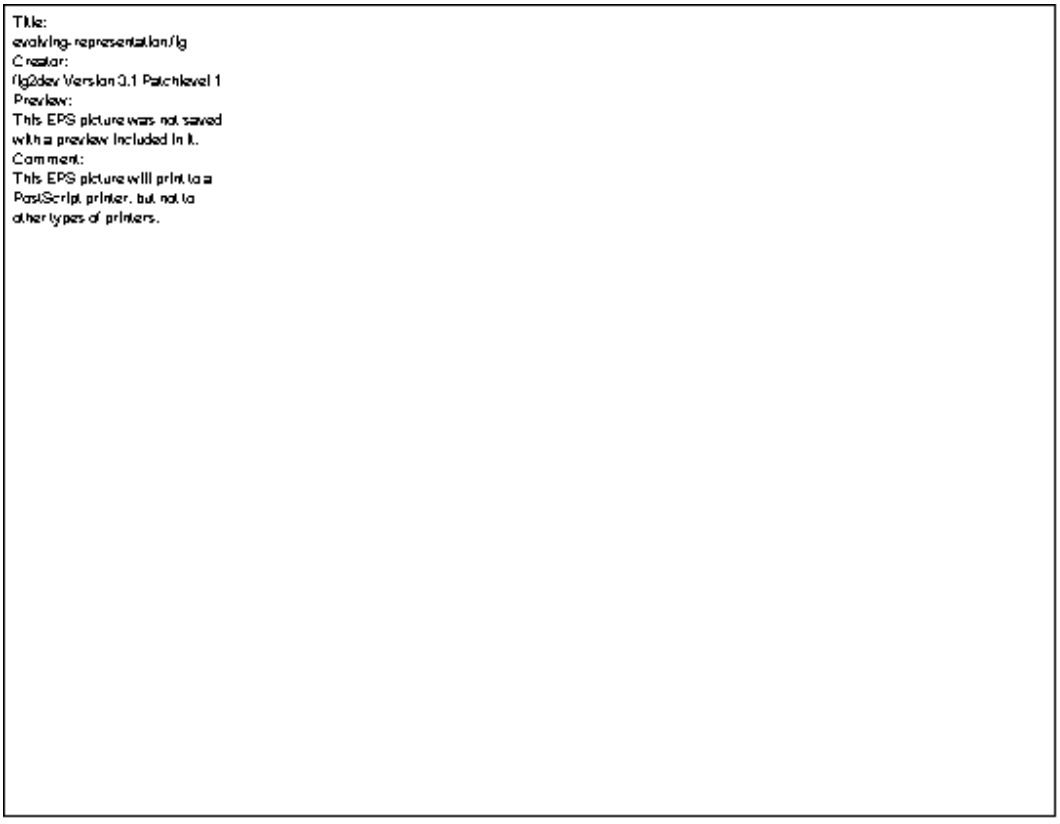

Figure 8: Example of an evolving representation: (a) original representation and (b) representation with evolved genes. Some of the corresponding genotypes are give, capital letters denote evolved genes. The transformation from phenotype to genotype is not always unique, e.g. the genotypes 'ABc' and 'BAc' produce the same phenotype

The introduction of evolved genes obviously changes the probability that a gene sequence maps onto a useful feature. While the number of different genes that can be used in a genotype expands, the length in the genotype that is necessary to describe a feature shrinks drastically. As an example, in Figure 9(a) a search space with the basic

coding is drawn, useful designs (black dots) may be far from the centre and from each other. The evolving representation modifies the search space, so that the useful designs lie much closer together. The part of the search space that has to be searched to find these designs is much smaller, Figure 9(b).

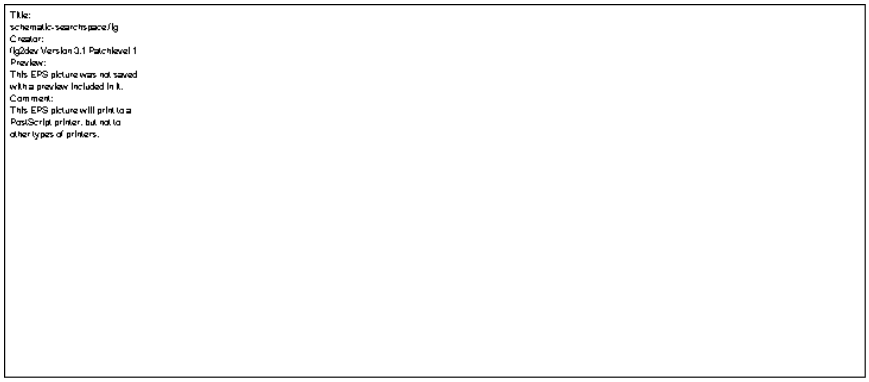

Figure 9: Effect of evolving representation: (a) original representation, interesting design states (black dots) are far apart, producing a large search space; and (b) evolved representation, the interesting design states are moved close together, resulting in a much smaller search space.

Analysis of conventional genetic algorithms shows that the convergence time usually grows with the length of the genotype, but this analysis relies on fixed-length genotypes and a fixed alphabet. Some numbers from the example in Section 3 may give a rough estimate: In that example, 366 evolved genes were created. Expressed in basic genes, these evolved genes have a medium length of about 17. Imagine a feature that can be described by two of these high level genes. To find this sequence among all genotypes of the same length, the probability is $\frac{1}{366^2} = \frac{1}{133956}$. If this feature were to be expressed in terms of the four different basic genes only, it would need a genotype of length 34. The probability here is $\frac{1}{4^{34}} = \frac{1}{2.9exp20}$. Since evolutionary search is not a blind search the actual difference is less.

Using Evolved Representations in Case-Based Design

As described in the introduction to this section, the idea of applying evolving coding in case-based design is that the representation evolves so that the search space focusses onto one design case, and can then be used again in a varied context (the new cases). This way, knowledge about the design case that is incorporated in the evolved genes can be used for the new case, without restricting the possibility of adapting to new requirements. Figure 10 illustrates this: starting with some basic building blocks, the system develops a description of the design, the frame with four squares. During

that process, evolved genes are created, e.g. those that represent two square shapes. The evolved representation (with the squares as basic building blocks) is then used to create a new design that is appropriate to the new design requirements.

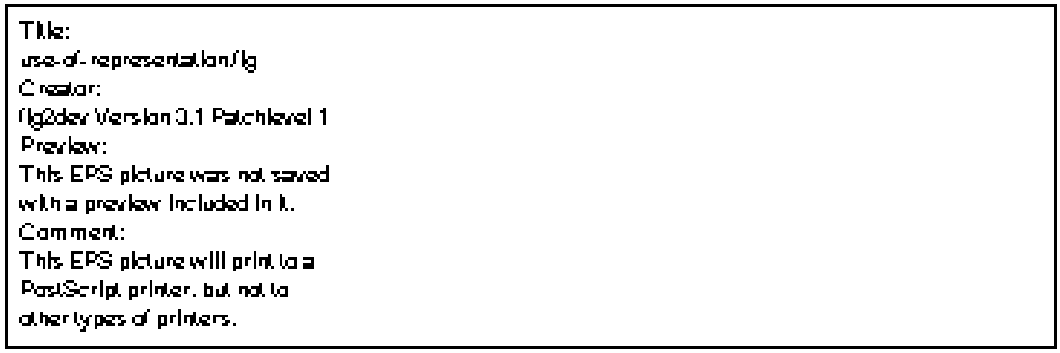

Figure 10: Use of evolved representation in case-based design

The final shape demonstrates an important point: the system can and will use knowledge derived from the design case (the two squares on top), but it can also use the basic building blocks (bottom part). The adaptations that are possible are therefore only restricted by what can be done with the basic coding. Imagine an application where finally one evolved gene is created that describes the whole design case. If this evolved representation is used for a new case, the amount of knowledge used from the design case can vary from

- 100%, using the single gene, no design variables is adapted; to
- 0%, using only basic building blocks, all design variables are adapted;

or any mixture of evolved genes of different complexity and basic genes, depending on how similar the new design requirements are to the design case.

Related work

In (Gero & Kazakov 1995), a formal description of a very similar approach can be found but it has not been applied to case-based design. Automatic organization into hierarchical structures has also been used in the context of genetic programming (Koza 1994, Angeline & Pollack 1994). As opposed to the application presented here, the basic function set in the genetic programming case is already relatively complex, and the abstraction by function generation usually does not exceed one or two levels of abstraction.

3. Example of Evolving Representation

In this example, the design case is given as a drawing. The process starts with a

representation that can be used to describe any kind of drawings, and gradually evolves this representation so that a focus is created, with the design case in the centre.

The process

As described in Section 2, the coding is allowed to evolve by identifying successful gene combinations and combining them into a new evolved gene. Successful for a gene combination means that it appears often in the genotypes of successful individuals, i.e. individuals with high fitness. It follows that if we want to create a coding that describes the design case, we have to use a fitness function that rewards individuals depending on how good they describe the design case. They have to describe the case both

- as exactly as possible, i.e. the use of any part of the evolved representation should not lead to individuals that cannot match the design case, and

- as completely as possible, i.e. as many aspects of the design case as possible are covered by the evolved representation

Generally, the fitness of an individual therefore depends on how much information about the design case it contains. To satisfy the first requirement, individuals that in any way do not match the design case are not allowed. In the example of floor plan layouts described below, the phenotypes (and the design case) represent line drawings, composed from elementary line segments (horizontal and vertical lines of one unit length). Every phenotype can be seen as a 'rubber stamp' that can be used to print different parts of the design case. The 'stamp' can be used only at positions where it would not create any lines (or line segments) that are not present in the design case. During the evolution, the 'stamps' get bigger and bigger, with the extreme case that at the end one 'stamp' (phenotype) is created that prints (describes) the whole case.

The fitness calculation for a new individual involves the following steps:

1. Transform the genotype of the individual into a phenotype, i.e. line-drawing.

2. Find all positions where the phenotype 'matches' the design case. A match is declared if and only if for all line segments in the phenotype there is a corresponding line segment in the design case (but not necessarily the other way round).

3. At all matching positions, draw the phenotype as a partial drawing.

4. Create the sum of the weights associated with all line segments in the design case that are represented in the partial drawing (see below for a

description how the weights are calculated).

5. This sum is the current fitness value for the individual. Whenever the weights for the segments in the design case are recalculated, the fitness values of all individuals in the population have to be updated.

As an example, Figure 11(a) shows a design case with associated weights, and Figure 11(b) shows two different individuals. Individual A does not fit the design case, while Individual B can be used to draw parts of it, as shown in Figure 11(c). The fitness of this individual would be 27, Figure 11(d).

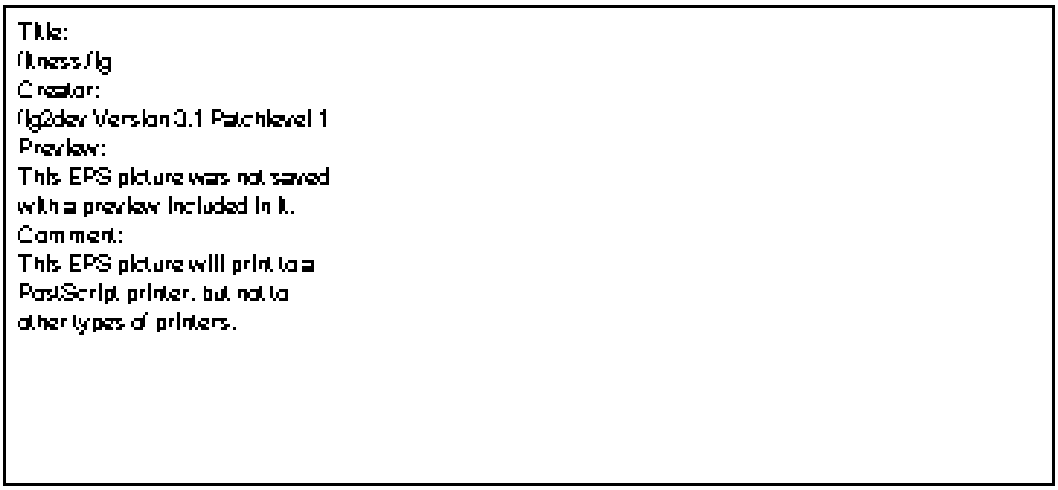

Figure 11: Example of fitness calculation: (a) the design case with associated weights, (b) two individuals, (c) parts of design case covered by Individual B at different positions, (d) total parts drawn by Individual B, resulting in a fitness of 27.

This fitness alone would lead to a convergence of individuals that describe only some aspects of the design case. To prevent this, another analogy from evolution in nature is used. The different aspects of the design case are seen as a resource (e.g. food) that has to be shared between all individuals using it. Individuals therefore get high rewards if they describe aspects of the case that are covered only by few other individuals (evolutionary niches), and only little additional fitness for aspects that are described by many individuals.

To create the 'niching' effect, every line segment in the design case has a weight associated with it. This weight is calculated regularly as a fixed value divided by the number of individuals in the population that can be used to 'stamp' that segment. If e.g. only two individuals code for a 'stamp' that can be used for a certain part in the design case, both get 50% of the constant value as fitness for that part. If 20 individuals do so, every one gets only a fitness of 5%. This automatically stops all trends of convergence towards only some features in the design case (e.g. only horizontal lines).

The fitness of an individual therefore depends on

- its length

- the number of places where it 'matches' parts of the design case

- the number of other individuals that can be used to describe the same line segments.

Since the goal during the development of a representation is variety and not optimization, all offspring created in the variation function are kept if they:

- are not empty, i.e. they contain some information about the design case. In the coding show in Figure 12, this means that they have to contain at least one (00) code

- match (as described above) the design case at least at one position

- no other individual already in the population has a genotype that codes for the same phenotype as the new individual. If such an individual exists, then the individual with the shorter genotype is kept. The use of evolved genes is hereby encouraged, since evolved genes usually lead to shorter genotypes. This is the only instance where an individual in the population can be replaced.

As described in Section 2, the basic coding has to be as simple as possible, excluding as much high-level-knowledge as possible, without restricting the set of possible solutions. We therefore 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 (see Figure 12(a)) Note that this is a different coding than the one used in the example in Section 2. In this coding, directions are relative to the previous direction, and unconnected lines are possible.

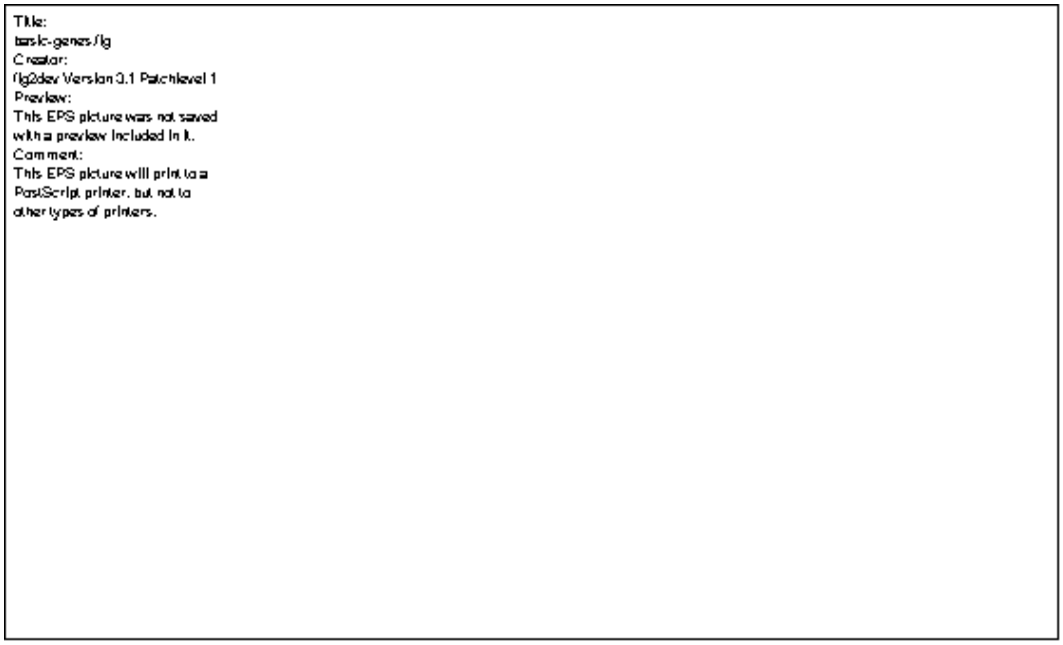

Figure 12: Basic coding, arrows show pen position and current direction before and after the gene is drawn (a) and example of its use in a genotype (b).

Genotypes are represented in the system as a list of genes. Figure 12(b) shows how the genotype ((00) (01) (00) (10) (11) (00)) is transformed into a phenotype (box) by assembling the corresponding line segments.

The coding can be used to represent any kind of two dimensional shapes 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 straightforward. The example discussed here uses the coding to create floor plans, the two floor plans shown in Figure 13 are used together as the design cases.

Figure 13: Room plans used as design 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: we only have to consider pairs of successive genes in the creation of evolved genes. This is because any combination of genes that are not directly successive will always depend on the genes between them. For example, codes for straight lines in first and third position in the genotype give very different results, depending on if the gene in the second position introduces a line, a blank step, a left turn or a right turn between them.

With these simplifications, we can use the following algorithm to create evolved genes:

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 of the individual by its length. Since the fitness of an individual is directly related to the length of the phenotype it represents, this gives the approximate contribution of every single gene in its genotype. In the table, add this value to all pairs occuring 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 14(a) shows a part of the hierarchical composition of one of the evolved genes (# 363) from lower-level evolved genes and basic genes (numbers in brackets). From left to right, the two genes drawn above each other are always combined into a new evolved gene. This new gene is then used as one of the two genes in the next step (the lower one). The thin arrow shows the points where the genes are combined. The numbers of the evolved genes represent the order in which they where created in the system. In the early stages of the evolution, the sequence of two basic genes that code for straight lines (code (0 0)) is represented very often in successful individuals. The first evolved gene, gene #0 therefore represents this combination. The next combination of genes that is identified as successful in the population is the sequence of a left turn and the evolved gene #0; this combination forms evolved gene #1. Gene #2 is not shown here, the fourth evolved gene created (#3) represents the combination of evolved gene #1 with evolved gene #0. Later in the gene evolution, this gene is used together with evolved gene #7 to create evolved gene #60. One of the last genes created in the run this example is based upon is evolved gene #363, from a combination of evolved gene #243 and evolved gene #155.

Figure 14(b) shows how the individual with the genetic code (289 267 287 363 246

246) is composed from six evolved genes. Again, the thin arrows show the places where the genes are connected. First, evolved gene #289 is drawn, the vectors (little arrows in the drawing) give the direction and position of the drawing-'turtle' before and after the gene is drawn. Evolved gene #267 is then drawn so that its start-vector coincides with the end-vector of gene #289. This is repeated for the other four evolved genes in the genotype of this individual. Gene #363 is the same one that is shown 'disassembled' above. Gene #246 is used in two different directions.

Figure 15 shows more examples of evolved genes.

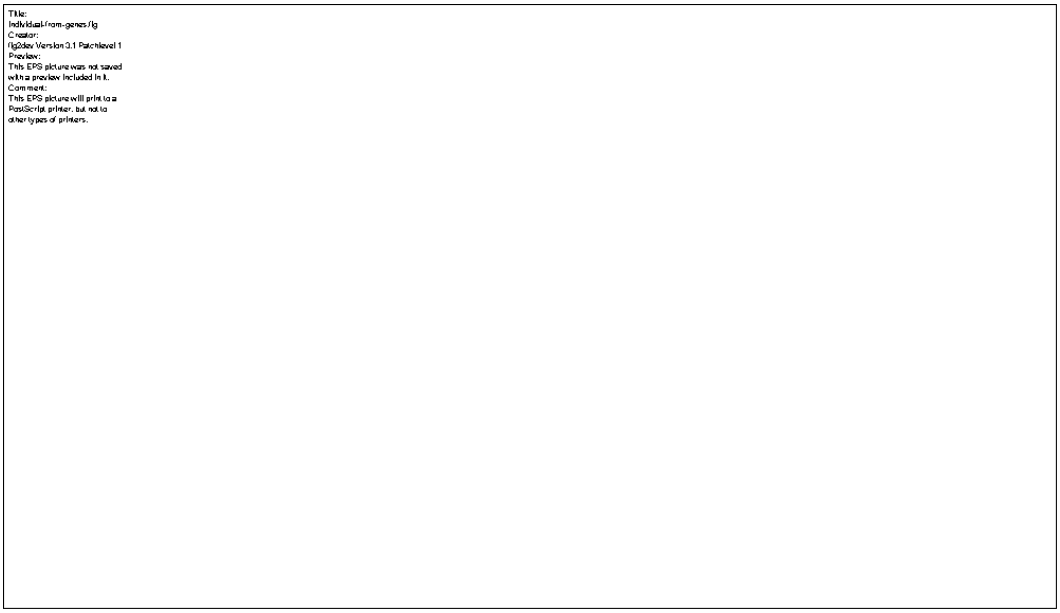

Figure 14: Evolved representation: (a) part of the hierarchical composition of the evolved gene 363 and (b) composition of the individual with the genotype (289 267 287 363 246 246).

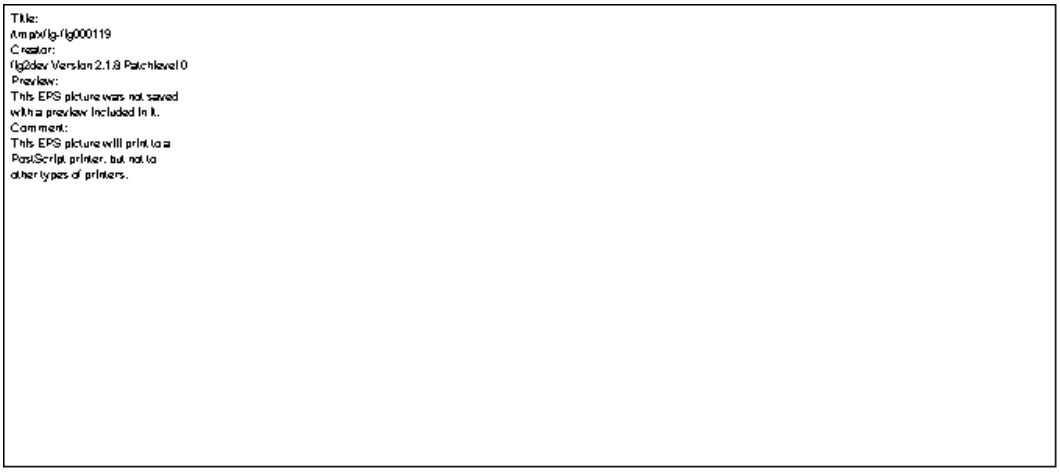

Figure 15: Evolved representation: phenotypes represented by evolved genes created from the design cases.

Examples

Example 1: minimizing wall length

In the run used for these examples, about 12000 individuals have been created, together with 366 evolved genes.

After a representation based on the design cases in figure 13 has been developed, it is used for new, different design requirements. A standard evolutionary algorithm is used, where the design requirements are coded into the fitness function. The representation is not evolving anymore, instead the full set of high level genes learned from the design case is used, together with the original basic genes.

As an example, the new design requirement was set to create a floor plan with minimal overall wall length, while at the same time fulfilling three additional design requirements:

- no walls with ``open ends'', that is walls that do not build a closed room
- fixed number of rooms
- fixed room areas, that is the sizes of the rooms ordered from largest to smallest were specified (e.g. biggest room 200 units, middle room 100 units, smallest room 75 units, for a three-room room-plan)

The additional requirements were given higher priority than the minimization of the wall length. The fitness of any design that violates any of the requirements is reduced accordingly (e.g. five instead of six rooms would lead to only 58% of the fitness). An exception to this is that small deviations from the room size (5 units per room in the examples below) do not incur a penalty.

Figure 16 shows the result of one run, after 100000 crossovers (producing 200000 new individuals) were performed. The population size was 1000 individuals. The number of rooms was set to six, with the areas 200, 150, 100, 75, and 50. All 366 of the evolved genes created from the design in Figure 13 have been used, together with the four basic genes. 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. At this stage, the best individuals seem to be variations of two basic layouts. The ability to vary layouts can be best seen in the last three layouts, where only the position of the two eastern rooms changes. All designs have six rooms, but in every design shown one of the rooms deviates about 10 to 15 units from the specified size. Since the restriction of the room sizes was given much higher priority than the minimization of the wall length, the system first tried to achieve correct room sizes. Only certain room forms

give the required sizes, this explains why the resulting plans are generally less square shaped than one might expect.

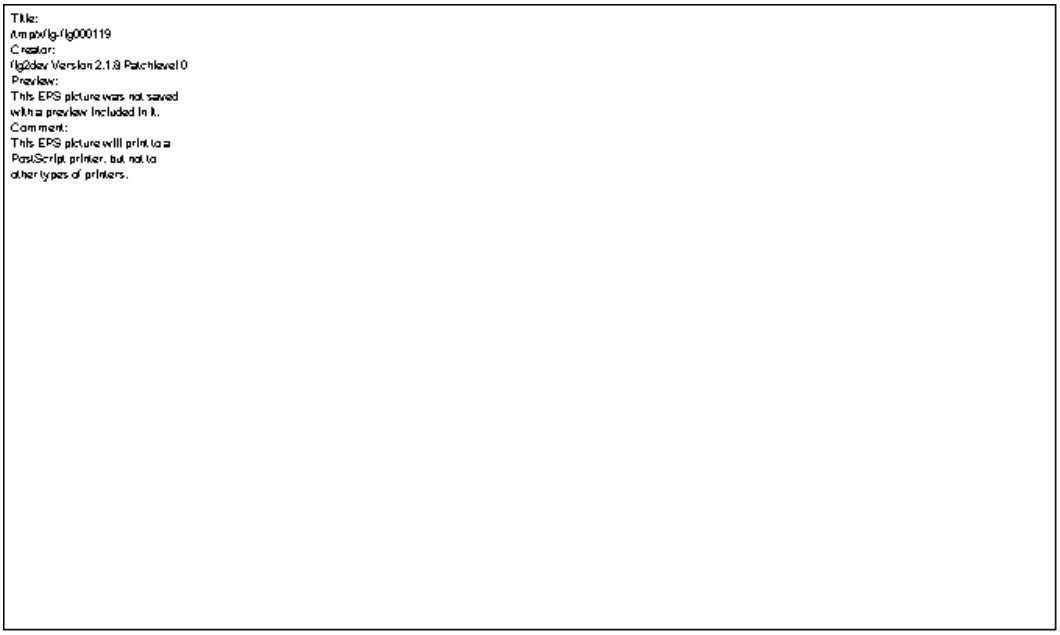

Figure 16 New floor plans, using design knowledge from the design cases. See text for design requirements.

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 design case. An analysis of the lengths of genes used in the genotypes of the final population can therefore show how much information from the design case is used in the new designs. Figure 17 shows a histogram for the example application. The peak at the left contains the four basic genes, which together are used about 900 times in the final population of 500 individuals. Many of these occurrences are left and right turns, 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. Most of the evolved genes in the population have lengths of around 12, but the final designs also use a considerable number of evolved genes with lengths around 40. This shows that the new designs indeed make use of the evolved representation to a very large degree.

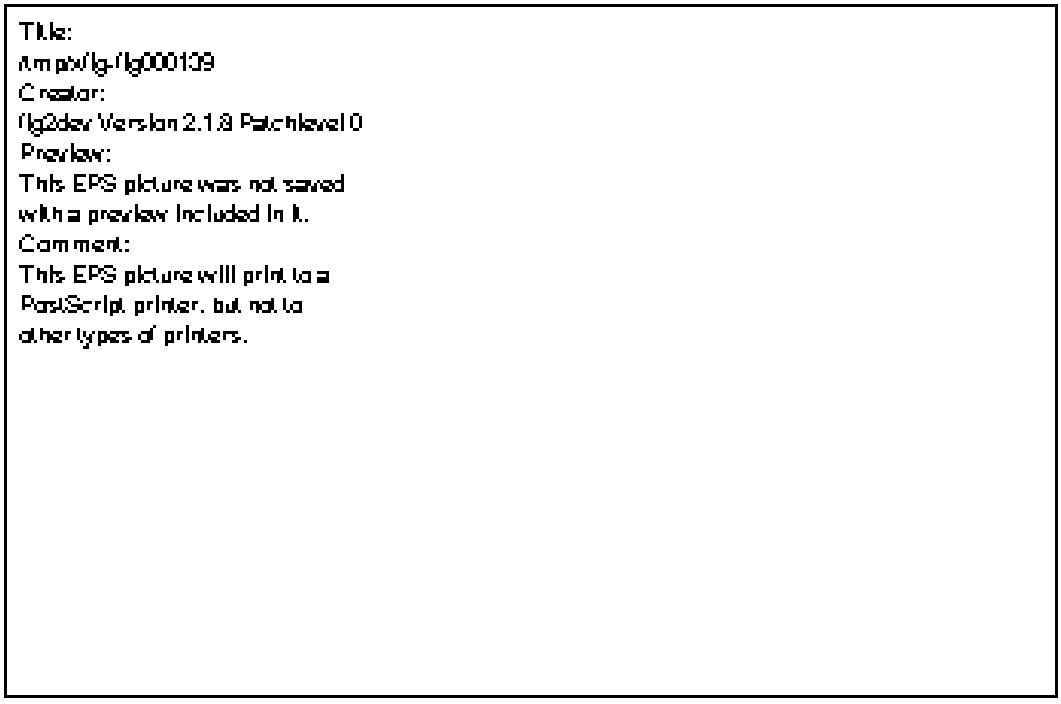

Figure 17: Histogram of the lengths of evolved genes (in basic genes) used by the new designs.

Example 2: reduced gene set

In this run, only a subset of the evolved genes were introduced into the population. The evolved genes used were the first 20 that were created from the layouts in Figure 13. Since the genes evolved first, they have a comparatively low complexity, on average they encode 6.25 basic genes. All other parameters and the design requirement are the same as in Example 1. The resulting designs are shown in Figure 18.

Figure 18: New floor plans, using a reduced set of evolved genes; requirements as for Example 1.

As in Example 1, the resulting layouts have six rooms, but compared with the previous example, the overall wall lengths are slightly shorter (114 in the best

individual compared with 117). However, in these layouts the size of two rooms deviate by more than five units from the size required, leading to lower fitnesses. We are currently investigating the influence of the number and complexity of evolved genes used. Due to the inherent randomness of the process, a high number of runs are required to obtain conclusive results. So far, it seems that both too few and too many evolved genes can slow down the generation of good designs.

Example 3: room plans with four rooms

In this example, the number of rooms required was changed to four, while the other fitness criteria remained unchanged from Example 1. Again, 100000 cross-overs were performed, and the full set of evolved genes was used. The results are shown in Figure 19.

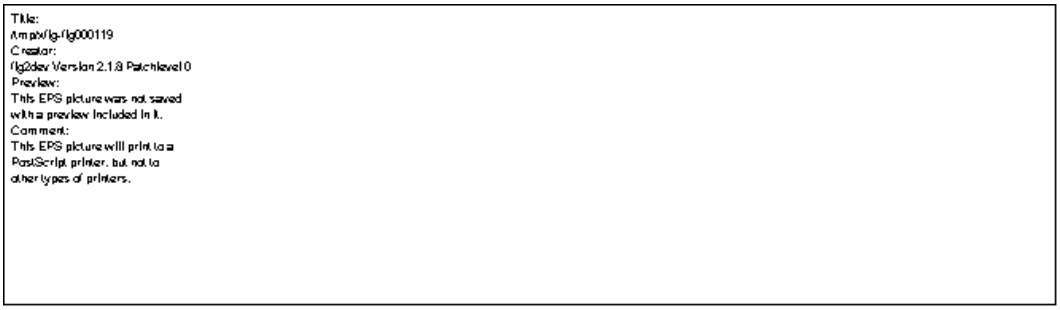

Figure 19: Result of a run with varied fitness: creating four rooms; other requirements as for Example 1.

Example 4: maximizing overall wall length

In another run, the fitness function was modified to produce high fitness if the overall wall length is maximized, while the additional design requirements remained as specified in Example 1. As illustrated in Figure 20, the system used the fact that the gain in wall length from an additional small room was higher than the penalty for a wrong room number count. The size of the additional room is 4 units, and therefore incurs no additional penalty for a wrong room size. Again, the system produced a number of variations of one basic layout, in this case by placing the small room in different positions.

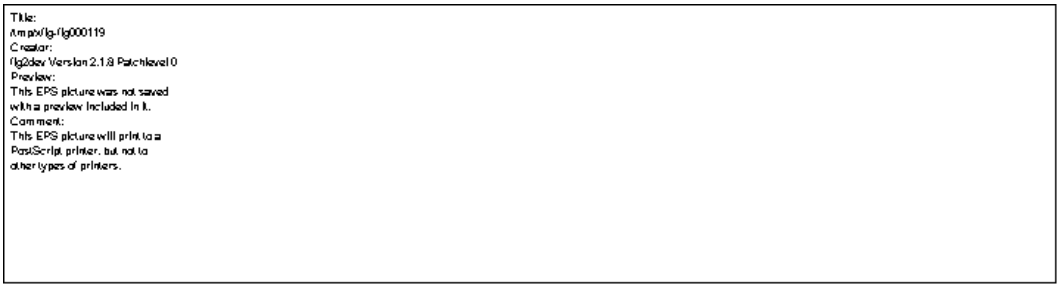

Figure 20: Results of a run with varied fitness: maximizing overall wall length; other requirements as for Example 1.

4. Discussion

The examples demonstrate the effectiveness of the genetic engineering approach to case-based design. It is possible to treat the evolution of a representation of a set of design cases as the equivalent of the development of a set of variables to describe a design or a set of designs. Each set of evolved genes dramatically biases the probabilities of finding a particular design which is an adaptation of the cases. In this sense we have a model of design exploration in a closed computational world. Each set of evolved genes can be considered to strongly define a subspace within the state space of all possible designs such that the designs which are produced from that representation will lie within the subspace. Thus, the evolution of the representation and its subsequent use matches our concept of creative design. Novel designs exhibit features of the existing designs but do so in unexpected ways.

If we compare the results shown in Figure 16 with those in Figure 20 we can see how altering the fitness dramatically changes the adaptation within this new space of possible designs. This is the same as an optimization formulation of a design problem where the variables are kept but the objective function is changed. It is the variables which define the state space of possible designs. the evolved genes play a dominant role in defining the probabilities which circumscribe the state space of likely designs. Each set of evolved genes defines a different state space of likely designs.

An interesting development of the approach described here is not simply to reuse evolved genes which are evolved from the initial elementary gene set but to allow ``true'' genetic engineering by inserting evolved genes with known phenotypic characteristics from another design or even another domain of designs. In this other design domain the elementary genes which are used in the evolution of the evolved gene may well be different. The effect of this is to introduce new genes into the representation in manner similar to the way analogy introduces new variables into a design. This would directly alter the state space of possible designs capable of being produced using these genes, thus producing potentially highly creative designs.

A more sophisticated form of genetic engineering would involve not just locating gene groups which produce high fitness behaviours but determining whether there was a higher level structure to these genes, a structure which might be conceived of as a controller for those genes. A modification to that controller might have a more

significant effect than gene evolution itself. Such higher level structure of genes could be readily substituted from one design domain to another to produce different ``styles'' of designing.

Acknowledgments

This work is supported by a grant from the Australian Research Council and by a University of Sydney Postgraduate Research Award.

References

Angeline, P. J. Pollack, J. B. 1994. Coevolving high-level representations, *in* C. G. Langton (ed.), *Artificial Life III*, Vol. XVII of *SFI Studies in the Sciences of Complexity*, Santa Fe Institute, Addison-Wesley, Reading, pp. 55-72.

Bäck, T., Hofmeister, F. Schwefel, H.-P. 1991. A survey of evolution strategies, *International Congress on Genetic Algorithms ICGA'91*, Morgan Kaufmann, San Mateo, CA, pp. 10-17.

Coyne, R. D., Rosenman, M. A., Radford, A. D., Balachandran, M. \ Gero, J. S. 1990. *Knowledge-Based Design Systems*, Addison-Wesley, Reading.

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.

Flemming, U., Coyne, R., Glavin, T. Rychener, M. 1988. A generative expert system for the design of building layouts, *Technical Report EDRC-48-08-88*, Engineering Design Centre, Carnegie Mellon University, Pittsburgh, PA.

Fogel, D. B. 1994. An introduction to simulated evolutionary optimization, *IEEE Transaction on Neural Networks* **5**(1): 4-14.

Gero, J. S. 1990. Design prototypes: A knowledge representation schema for design, *AI Magazine* **11**(4): 27-36.

Gero, J. S. 1994. Towards a model of exploration in computer-aided design, *in* J. S. Gero E.Tyugu (eds), *Formal Design Methods for CAD*, North-Holland, Amsterdam, pp. 315-336.

Gero, J. S. Kazakov, V. A. 1995 . Evolving building blocks for design using genetic engineering: a formal approach, *in* J. S. Gero F. Sudweeks (eds), *Preprints Advances in Formal Design Methods for CAD*, IFIP - University of Sydney, Sydney, pp. 29-48.

Koza, J. R. 1994. Architecture-altering operations for evolving the architecture of a

mutil-part program in genetic programming, *Technical report*, Computer Science Department, Stanford University, Stanford, California 94305-2140 USA.

Langton, C. G. 1988. Artificial life, *in* C. G. Langton (ed.), *Artificial Life*, Vol. VI of *SFI Studies in the Sciences of Complexity*, Addison-Wesley, Reading, pp. 1-47.

Maher, M. L., Balachandran, M. B. Zhang, D. 1995. *Case-Based Reasoning in Design*, Lawrence Erlbaum, Hillsdale, NJ.

Maher, M. L., Zhao, F. Gero, J. S. 1989. Creativity in humans and computers, *in* J. S. Gero \ T. Oksala (eds), *Knowledge-Based Systems in Architecture*, Acta Scandinavica, Helsinki, chapter 13, pp. 129-141.

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.

Spears, W. M., Jong, K. A. D., Baeck, T., Fogel, D. B. de Garis, H. 1993. An overview of evolutionary computation, *Machine Learning: ECML-93*, Springer Verlag, Berlin, pp. 442-459.

Yaeger, L. 1994. Computational genetics, physiology, metabolism, neural systems, learning, vision and behaviour or PolyWorld: Life in a new context, *in* C. G. Langton (ed.), *Artificial Life III*, Vol. XVII of *SFI Studies in the Sciences of Complexity*, Santa Fe Institute, Addison-Wesley, Reading, pp. 263-298.

This is a copy of: 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.