TOWARDS A MODEL OF EXPLORATION IN COMPUTER-AIDED DESIGN

J. S. Gero

This paper draws a distinction between search and exploration in design. Search is a process for locating values of variables in a defined state space whilst exploration is a process for producing state spaces. The paper proceeds to elaborate two exploration processes applicable in computer-aided design: emergence and evolutionary combination.

1. Introduction

Design distinguishes itself from other complex human intellectual activities in a number of important and significant ways. Design is that activity which is primarily concerned with changing the world.

Design can be initially characterised as an intentional, purposeful activity. This implies that we can model design as being goal-oriented, but as we shall see unlike problem solving which is also a goal-oriented activity, the goals in designing are not fixed and determining them is part of designing. Design is also a constrained activity, i.e. there are constraints which limit the designer's ability to achieve goals. Some of these constraints are based on the behaviour of the physical world, others are based on the designer's perceptions and interpretations of design situations, others are implicit in the representations and processes utilised. Design can be conceived as a decision making activity. This implies that choices exist; in computational terms this implies variables, the values of which have to be decided. (Variables in design include both descriptors and the relationships between them.) As is suggested later the variables themselves may be treated as the values for other variables. Search is the common process used in finding values for symbolic variables. We will be suggesting that the process of determining the space within which to search is a form of exploration. In addition to exploration and search design involves learning where learning implies a restructuring of the knowledge used. What defines the goals, the constraints, the knowledge, the processes and the focus of design depends on the context within which the designer operates. This context depends largely on the designer's perceptions of what makes up the context. These perceptions change as design occurs resulting in a perceived change of context.

From a computational viewpoint design can be treated paradigmatically as a process of producing variables, determining relationships between them and finding values for some of those variables such that useful values are determined for some other of those variables. In design the variables can be conveniently categorised into a number of classes, the three most significant

of which are those variables which define structure, those variables which define behaviour and those variables which define function. The variables used to describe structure are also often called design or decision variables, whilst those used to describe behaviour are also often called performance, objective or criteria variables.

One useful way to provide a framework for design is through the conceptual schema design prototypes (Gero, 1987; 1990) which articulates a function-behaviour-structure + knowledge + context framework. Thus, the state space representation of designs has three subspaces or abstractions: the structure space, S (often called the decision space); the behaviour space, B (often called the performance space); and the function space, F (which defines the artefact's teleology). Figure 1 shows these three subspaces which constitute the state space of designs.

Whilst there are transformations which map function to behaviour and vice-versa and structure to behaviour and vice-versa, there are no transformations which map function to structure directly. This is a version of the no-function-in-structure principle (de Kleer and Brown, 1984; Gero, 1990) where the teleology of an artefact is not found in its structure but is a contextual interpretation of its behaviour. The corollary: no-structure-in-function also holds. This may, at first glance, be counter-intuitive. The reason is that in human experience once a phenomenological connection between function and structure is made it is hard to unmake it. Thus, it appears that there is some sort of causal transformation between function and structure rather than a phenomenological one. The transformation which maps structure to behaviour is a causal one.

Often only the structure and behaviour spaces are considered in computational models although function provides an important articulation of ideas about design. Typical computational models of design can be grouped based on the processes they utilise under such processes as simulation, optimization, generation, decomposition, constraint satisfaction, and more generally search and exploration. All of these share one concept in common, namely that structures are produced in a design process and their resultant behaviours are evaluated. It is only recently that the function of the artefact being designed is beginning to be brought into the computational model (Ganeshan et al., 1991; Garcia and Howard, 1991; Rosenman, 1991; Rosenman et al., 1993).

In this paper we will focus on the exploration aspect of designing from a computational viewpoint. We will provide formalisable computational constructs for exploration and its supporting processes. In particular, we will present and examine two generic processes which allow exploration to occur. The first is emergence and the second is combination. As we shall see although these processes are very different their application results in changes which match the notion of exploration we will be using in this paper.

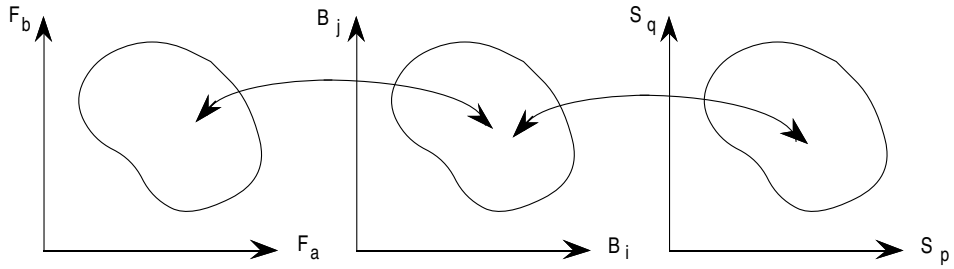

Fig. 1. The three subspaces of function, F, behaviour, B, and structure, S, which constitute the state space of designs

Logan and Smithers (1989) and Smithers (1992) have presented cogent arguments as to why design is not search and why design is more like exploration. They have presented a process model of design as exploration and have given hand-worked examples of designing in a blocks world. In this paper we aim to characterise exploration initially in terms of a state-space representation before developing and describing the computational constructs of interest.

2. Search and Exploration

Since there is very little work on computational characterisations of function, function variables and their processes, we will restrict our discussions, in general, to structure and behaviour only.

2.1. Search in design

Search, as a computational process, requires that the state spaces of behaviour and structure be well-defined, i.e. that all the states be directly specifiable a priori. In design this implies that the variables which define the structure and the behaviour are known a priori as are the relationships between them. Search then determines feasible, satisficing or, in appropriate circumstances, optimal values for structure variables which produce desired behaviours, Figure 2. In those situations where the values of the variables are in the domain of natural numbers the search problem can be cast as an optimal design problem. Under these conditions the optimization processes are the search processes. When the values of the variables are in the domain of symbols then a wide variety of artificial intelligence approaches based on various forms of reasoning are available to search for symbolic values.

How does search contribute to design? In that aspect of designing when all the variables (at some level of granularity) have been produced and the relationships between them are known then search is a most appropriate process. Whilst search does not provide an adequate paradigm for design

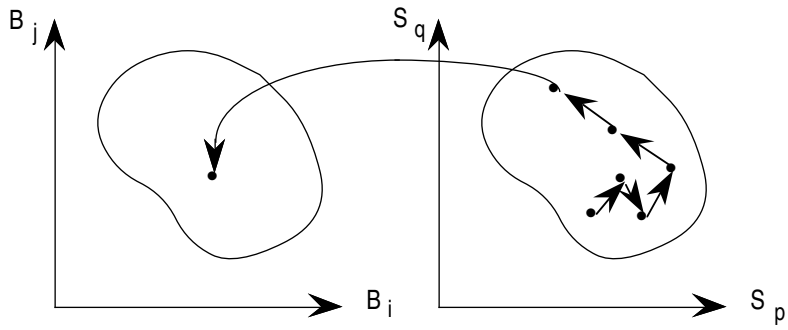

Fig. 2. Search is carried out within a fixed design state space. The structure space is searched, guided by the performance in the behaviour space.

such situations do occur at the base of many design activities. Search does form the basis of that class of design defined as 'routine design'. Thus, in a sense, search is a foundational, necessary but not sufficient, process for any computational model of design. What search fails to deal with is the process of producing variables and determining relationships between them.

2.2. Exploration in design

Exploration in design can be characterised as a process which creates new design state spaces or modifies existing design state spaces. New state spaces are rarely created de novo in design rather existing design state spaces are modified. The result of exploring a design state space is an altered state space.

For a given set of variables and processes operating within a bounded context or focus any computational model will construct a bounded (although in some cases countably infinite) state space. Exploration in design can be represented in such a state space by a change in the state space. Exploration maps onto the concept of non-routine design (Gero and Maher, 1992). Any of the subspaces in Figure 1 for function, behaviour or structure could be changed although, in general, in design it is the structure space that is changed. Exploring the structure space introduces new structure variables which may, although not necessarily, introduce new behaviours associated with those variables. Exploring the behaviour space introduces new behaviour variables which may, although not necessarily, introduce new functions and structures associated with those variables. Exploring the function space introduces new function variables which may, although not necessarily, introduce new behaviours associated with those variables, i.e.

$$S_{\text{new}} \longrightarrow \begin{cases} B_{\text{exisiting}} \\ B_{\text{new}} \end{cases}$$

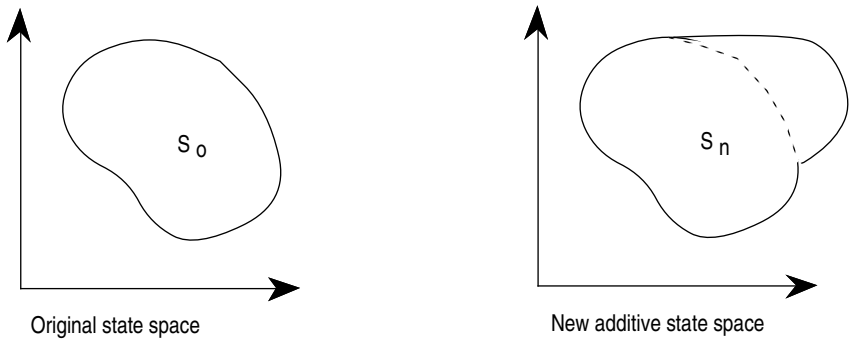

Fig. 3. The additive state space view.

$$B_{\text{new}} \longrightarrow \begin{cases} S_{\text{exisiting}} \\ S_{\text{new}} \end{cases}$$

$$B_{\text{new}} \longrightarrow \begin{cases} F_{\text{exisiting}} \\ F_{\text{new}} \end{cases}$$

$$F_{\text{new}} \longrightarrow \begin{cases} B_{\text{exisiting}} \\ B_{\text{new}} \end{cases}$$

where $\longrightarrow$ means implies.

There are two classes of state-space change possible: addition and substitution. This is based on Stevens' two forms of psychological representational scales (Stevens, 1957). The additive view is presented conceptually in Figure 3 where the new state space S_n totally contains the original state space S_o, i.e. $S_o \subset S_n$ and $S_n - S_o \neq \emptyset$

The implication of the additive view is that variables are added to the existing stock of variables. Gero and Kumar (1993) have demonstrated how the addition of structure variables allows design spaces that contain infeasible behaviour spaces to be made feasible. Further, they demonstrated how the addition of structure variables can improve the behaviour of an already optimized design.

The substitutive view of state-space change is presented conceptually in Figure 4 where the new state space S_n does not cover the original state space S_o, i.e. $S_o \not\subset S_n$.

The implication of this substitutive view is that some existing variables are deleted and others added. There is no nexus between the number of existing variables deleted and the number of new variables added. As will be seen later this view matches the concept of emergence.

Thus, exploration precedes search and it, effectively, converts one formulation of the design problem into another. If the final formulation is the

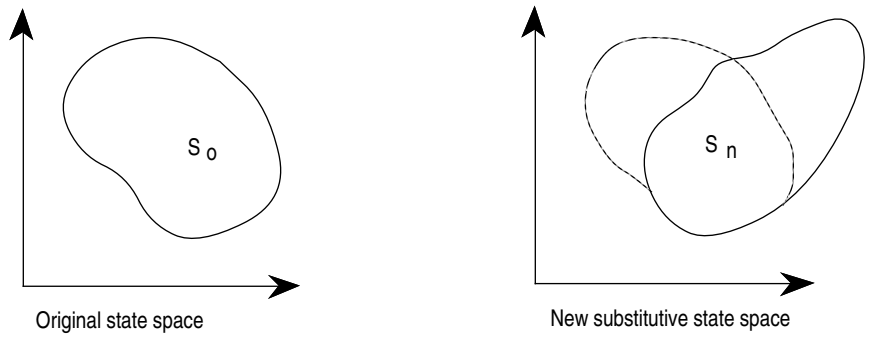

Fig. 4. The substitutive state space view.

accepted one we say that it is well-structured in contrast to the previous formulations which we would call ill-structured. This notion of structuring refers not to the definedness of any formulation but to the distance from the final formulation. The reason for the distinction between definedness and structuring is to recognise that all computational constructs need to be well defined but need not match the semantics of what is being represented. Part of designing involves determining what to design for (function or teleology), determining how to measure satisfaction (behaviour), and determining what can be used in the final artefact (structure). Exploration is the process which supports these design determinations. Having made these decisions via exploration then search takes over.

The next sections introduce and describe two computational processes which can be used as exploration processes.

3. Emergence as Exploration in Design

3.1. Emergence

A property that is not represented explicitly is said to be an emergent property if it can be made explicit. There are three views of emergence: computational emergence; thermodynamic emergence; and emergence relative to a model (Cariani, 1992). *Computational emergence* is the view that novel behaviours can emerge as a result of local computational interactions (Forrest, 1990; Steels, 1991; Langton, 1989). This is one of the approaches to the field of artificial life. *Thermodynamic emergence* is the view that thermodynamic theory may be used to describe how new, stable behaviours and structures can arise at loci removed from known equilibrium. *Emergence relative to a model* sees emergence as a deviation of the structure or behaviour of a system from an observer's model of it. It is this latter view of emergence that we wish to use in this paper.

We suggest that emergence is one computational process which is capable

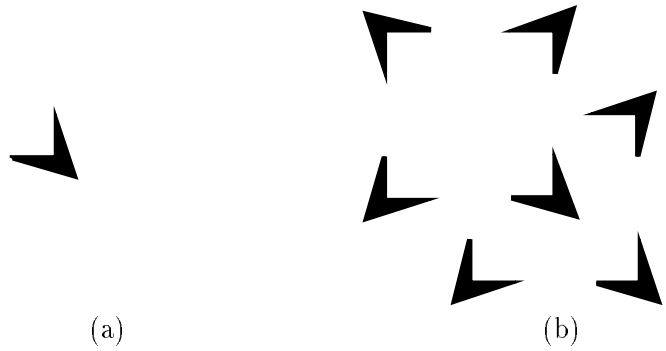

Fig. 5. (a) Single visual form; (b) seven copies of the single form in particular locations such that a number of visual forms emerge.

of supporting exploration in design. It is a process capable of modifying the state space under consideration and matches the concept of a designer changing his or her context. Emergence, in the way described in this paper, matches the substitutive state space view.

Emergence is a well recognised phenomenon in visual representations of structure although it is not limited to that field. We will use visual emergence as the vehicle to introduce and describe the computational process suggested. Consider the visual form in Figure 5(a), if a designer locates seven of these in a particular configuration as shown in Figure 5(b) it is possible to see, cognitively, a number of emergent forms.

One computational model of form emergence is based on notions drawn from fixation where a particular representation is taken to prevent other views from being 'seen'. Thus, the model separates out representation, schemas in representations and processes operating on those representations and introduces an alternate representation which eliminates the fixation (Gero and Yan, 1993). More will be said on this later in the paper. It is sufficient to state at this stage that at least one implemented computational model exists for shape or form emergence capable of 'discovering' the two emergent squares in Figure 5(b).

3.2. Emergence in design

Emergence plays an important role in design both within a domain and within a design. Emergence allows for new views of existing situations to come into being and thus becomes a way of changing the direction or focus of a design. More formally, emergence allows for new instances of the schema under consideration to be found; instances which were not previously represented. For example, consider Figure 6, the two triangles labeled $T1$ and $T2$ have been drawn, triangle labeled $T3$ is a third triangle which was not

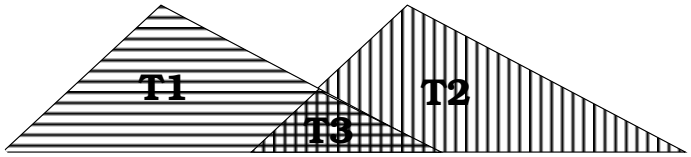

Fig. 6. The two triangles $T1$ and $T2$ are drawn whilst triangle $T3$ emerges.

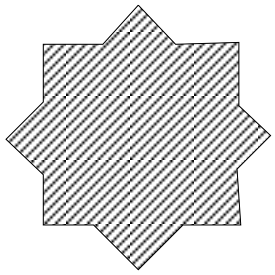

Fig. 7. A sixteen-sided form produced using a schema of a sixteen-sided form.

drawn but emerges. Triangle $T3$ is a new instance of a triangle schema and it now can be used in the design.

Emergence allows for new schemas to emerge, schemas which were not in the original representation. For example, consider Figure 7 which shows a sixteen-sided form. Figure 8 shows two squares which emerge from the original form. The square is a new schema discovered using a form of data-driven or feature-driven search to produce it.

The emergent squares now provide design opportunities not available when the form was viewed as a sixteen-sided form only. For example, the two squares may be rotated with respect to each other or they may be separated and so on. What is important is that a new design space has been

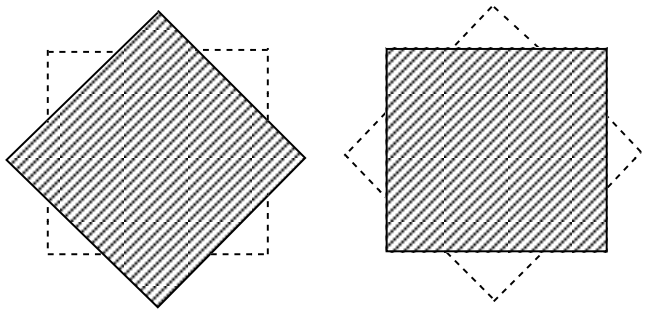

Fig. 8. Two four-sided forms which emerge from the sixteen-sided form shown in Figure 7, discovered using an emergence process (Gero and Yan, 1993).

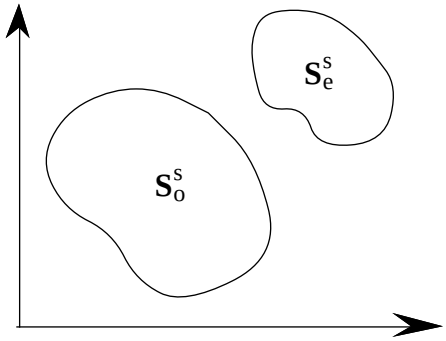

Fig. 9. The structure state spaces of the original and emergent forms.

constructed from these particular emergent forms. If we let be the state space of the structure of the original form and be the state space of the structure of the emergent square forms then

$$S_o^s \cap S_e^s = \emptyset$$

Figure 9 shows this conceptually. Even though the structure state space may not intersect, as we stated earlier, this does not necessarily mean that the behaviour and function state spaces of the emergent structures have any new variables introduced into them.

From this characterisation we can see that emergence meets the requirements of being an exploration process in that it changes the state space.

It is worth examining the differences between schemas and simple representations. A schema (in the Kantian sense) is the conceptual abstraction of entities. In this sense schemas allow for the structured representation of descriptions. For example we can have a schema for triangles and another for squares as well as one for polyline shapes. Before we discuss schema emergence we need to draw a distinction between structure emergence and schema emergence.

Let us consider again the situation in Figure 6. A new structure, $T3$, emerges which has the same schema as the original structures, i.e.

$$S_o^s \cap S_e^s \neq \emptyset$$

However, if we let S_o^s be the original schema and S_e^s be the schema of the emergent structure then there are three situations possible.

if $S_o^s \cap S_e^s \neq \emptyset$

and $S_o^s \cap S_e^s = S_o^s$

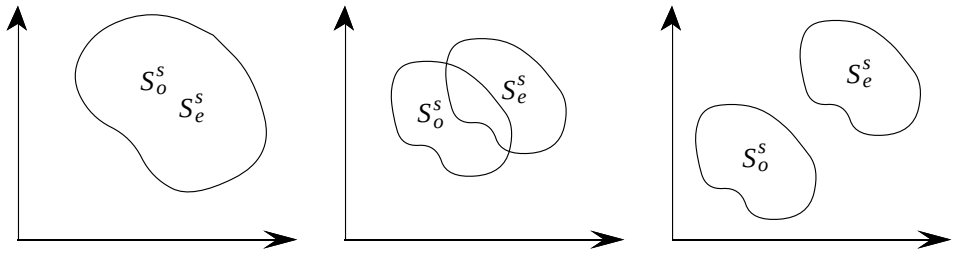

Fig. 10. State space representations of schemas:
(a) Emergent structure has same schema as original structure. (b) Emergent structure has schema related to schema of original structure. (c) Emergent structure has schema unrelated to schema of original structure.

then the schema of the original structure and the emergent structure are the same. This is the case for the emergent triangle in Figure 6.

if $S_o^s \cap S_e^s \neq \emptyset$

and $S_o^s \cap S_e^s \neq S_o^s$

then the schema of the emergent structure is different to that of the original structure but there is some overlap between them. This is the case for the emergent squares in Figure 8.

if $S_o^s \cap S_e^s \neq \emptyset$

then the schema of the emergent structure is both different to that of the original structure and there is no overlap between them. This is the case for the emergent squares in Figure 5.

These three cases are shown in Figure 10.

3.3. Process model of emergence

One approach to the development of a process model of emergence is to re-examine concepts associated with fixation. Fixation occurs when some aspect of the representation of a situation prevents us from viewing that situation in another way (Weisberg and Alba, 1981). We can see a demonstration of this in Edward de Bono's 'tennis game problem' (de Bono, 1972).

Suppose you need to schedule the games in a competition for your tennis club. You have 97 players registered. The first question to ask is how many games need to be scheduled and therefore to determine how long the competition will last. One way to determine how many games need to be scheduled is as follows. You need to know that each game produces one winner and one loser; only winners proceed to the next round to play other

Round	No of players	Bye player	No of games	Cumulative no of games
1	97	1	48	48
2	49	1	24	72
3	25	1	12	84
4	13	1	6	90
5	7	1	3	93
6	4	0	2	95
7	2		1	96

Fig. 11. Calculating the total number of games needed to be scheduled.

winners until you reach the final game. The winner of the final game is the club champion.

Since you need to have even numbers of players to produce games whenever you have an odd number of players, one player has a 'bye' and automatically goes into the next round as if he or she were a winner. The winners from each game go into the next round. These two rules are applied iteratively. Figure 11 shows the application of these rules to determine the number of games which need to be scheduled. Clearly, for each competition which has a different number of entrants a new set of calculations will be needed.

We can characterise the schema being used here as the 'winning game' schema, i.e. each game produces a winner who proceeds to the next round. Another schema may be characterised as the 'losing game' schema. Here, each game produces one loser. Irrespective of how often a player has won, a single loss forces them out of the competition. Only one player, the club champion, loses no games. Since each game produces one loser and only one player has no losses the number of games to be scheduled is: $97 - 1 = 96$ or, more generally, the number of players minus one.

Adopting the winning game schema prevents you from using the losing game schema which, in this case, is a superior approach to the solution of this simple problem. Fixation occurs when one schema prevents the use of another schema. We can extend the notion of fixation to be applied to those situations where any form of representation of features prevents the representation of other features. This will allow us to utilise a single conceptual framework to describe process models for both structure and schema emergence.

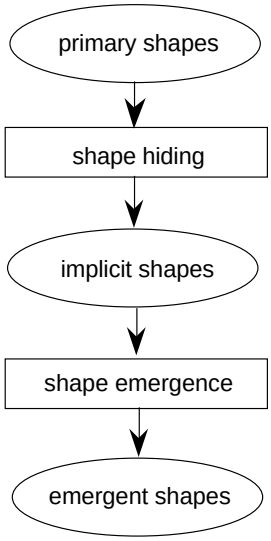

Fig. 12. A process model of shape emergence. Rounded blocks show representations, squared blocks show processes.

3.3.1. Structure emergence

A process model for structure emergence has been developed by Gero and Yan (1993) for the emergence of shapes. It has two fundamental steps: shape hiding and shape emergence. Shape hiding changes the representation of shapes such that the original or primary shapes are no longer specifically represented, i.e. are hidden. To do this, an alternate representation is used. This produced implicit shapes only. Shape emergence applies shape schemas to the implicit shapes to determine whether new shapes (additional to the primary shapes) can emerge. Figure 12 provides an outline of this process.

This can be generalised beyond shapes as shown in Figure 13.

Whilst such a model has been developed and implemented for two-dimensional closed shapes it does not appear to have been implemented elsewhere although we can readily conceive of its application in other domains. Consider the domain of structural engineering applied to buildings. The engineer commences with a set of parallel frames which are used to support both vertical and lateral loads, Figure 14.

In order to provide lateral stability the engineer adds horizontal bracing at each floor to produce the engineering structure shown in Figure 15. Now another set of frames emerges, frames which are not explicitly represented as frames initially, Figure 16. Subject to an appropriate representation these emergent frames can be found using the same schema as for the primary frames.

The model in Figures 12 and 13 can be described as follows:

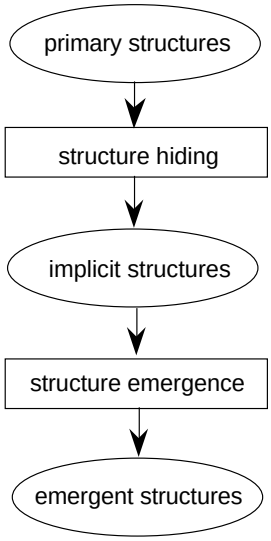

Fig. 13. A process model of structure emergence.

where α = subscript
a = subscript for alternate
e = subscript for emergent
o = subscript for original or primary
R_α = representation
S^s_α = a structure
S^s_k = a schema

The primary or original structure can be described in the original representation as

$$S^s_{\alpha=0}(R_{\alpha=0} \mid S^s_{\alpha=0})$$

The primary structure can be described in an alternate representation as

$$S^s_{\alpha=0}(R_{\alpha=a} \mid S^s_{\alpha=0})$$

and all the implicit instances of the schema in that representation can be described as

$$S^s_{\alpha=e}(R_{\alpha=a} \mid S^s_{\alpha=0})$$

where $S^s_{\alpha=0} \subset S^s_{\alpha=e}$.

However, it is possible to provide different schemas so that other structures may emerge

$$S^s_{\alpha=e}(R_{\alpha=a} \mid S^s_{\alpha=a})$$

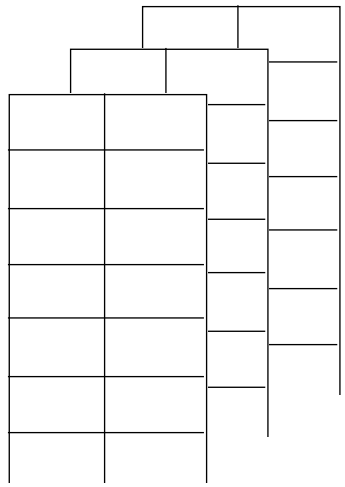

Fig. 14. Building structure as a set of parallel load resisting frames.

3.3.2. Schema emergence

The same concepts may be applied to the emergence of schemas although different processes are required. Figure 17 shows a process model of schema emergence analogous to the structure emergence process model of Figure 13.

The primary or original schema can be described in the original representation and original structure as

$$S^s_{\alpha=0}(R_{\alpha=0} \mid S^s_{\alpha=0})$$

The original schema can be described in an alternate representation as

$$S^s_{\alpha=e}(R_{\alpha=a} \mid S^s_{\alpha=0})$$

or

$$S^s_{\alpha=e}(R_{\alpha=a} \mid S^s_{\alpha=0})$$

New schemas can be derived from the new representation and possible alternate structures and can be described as

$$S^s_{\alpha=e}(R_{\alpha=a} \mid S^s_{\alpha=a}) \longrightarrow S^s_{\alpha=e}(R_{\alpha=a} \mid S^s_{\alpha=e})$$

Processes for structure emergence have been developed for the case of structures of two-dimensional closed shapes. A data-driven process for schema emergence has been developed for schemas describing two-dimensional closed shapes (Gero and Yan, 1993). This process can be described in terms of the concept of schema classes. A schema class is defined by the set of common attributes of a class of schemas. Individual schemas are instances of the

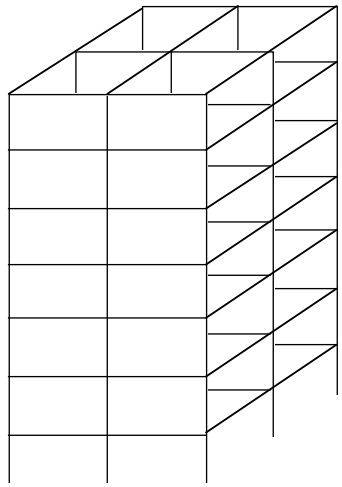

Fig. 15. Building structure with horizontal lateral bracing added.

schema class with additional constraints. In this case the common attributes are that all the individual schemas in the class describe two-dimensional closed shapes. Instances of the schema define a state space of structures. A schema class is used to find schemas implicit in a representation.

Take for example a representation of shapes based on bounding lines. There may be a schema associated with three bounding lines and some constraints called a triangle. This schema can be used as a means to search the data in that representation for matching instances. Alternately, we may define a schema class based only on bounding lines closing and search the data for any member of that schema class and find, for example, closed shapes with four bounding lines, two of which are parallel. This becomes a 'new' schema which can be used to search the data for instances of itself.

3.4. Emergence as exploration in design

We treat exploration as being concerned with determining spaces within which search then takes place. Exploration may be considered as being akin to determining which variables are going to be used in describing a design. Structure emergence takes an existing structure state space and modifies it by adding to it. The modification can involve the introduction of both new descriptor variables and new relationship variables. The modification can be either additive or substitutive depending on how it is treated. Schema emergence also takes an existing state space and modifies it. Both of these forms of emergence meet the definition of exploratory processes.

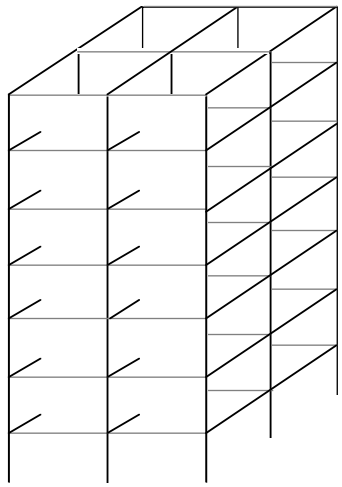

Fig. 16. Building structure with emergent frames highlighted.

4. Evolutionary Combination as Exploration in Design

4.1. Evolutionary combination

Combination is one way of changing a state space. A number of conceptually similar processes exist for combination largely based on analogy (Qian and Gero, 1992) and case-based reasoning (Riesbeck and Schank, 1989). Another basis of change produced by combination leads to the concept of evolution and an analogy with natural evolution and evolutionary processes. Woodbury (1989) was one of the early proponents of using formal models of evolution in design, although the concept has been discussed informally for some time (Steadman, 1979).

One common interpretation of computational models based on the genetic processes of crossover ($\equiv$ combination) and mutation is that of a model of search (Goldberg, 1989). However, it will be argued in the next section that the field of genetic algorithms can be applied to exploration as well as search. We suggest that evolutionary combination is one computational process which is capable of supporting exploration in design. It is a process capable of modifying the state space under consideration and matches the concept of a designer changing his or her context. Evolutionary combination, as described in this paper, matches the substitutive state space view.

In genetic algorithms, a population of 'organisms' (usually represented as bit strings) is modified by the probabilistic application of the genetic operators from one generation to the next. The basic algorithm where $P(t)$ is the population of strings at generation t, is given below.

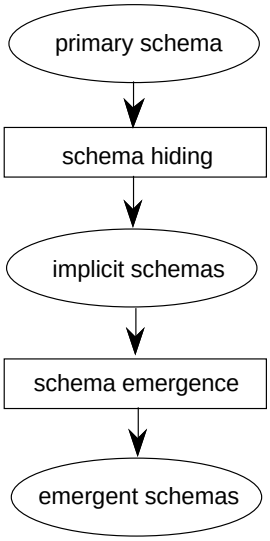

Fig. 17. A process model of schema emergence.

$t = 0$
initialize $P(t)$
evaluate $P(t)$
while (termination condition not satisfied) **do**
begin
  select $P(t+1)$ **from** $P(t)$
  recombine $P(t+1)$
  evaluate $P(t+1)$
  $t = t + 1$
end

Evaluation of each string which corresponds to a point in a state space is based on a fitness function that is problem dependent. This corresponds to the environmental determination of survivability in natural selection. Selection is done on the basis of relative fitness and it probabilistically culls from the population those points which have relatively low fitness. Recombination, which consists of mutation and crossover, imitates sexual reproduction. Mutation, as in natural systems, is a very low probability operator and just flips a specific bit. Crossover in contrast is applied with high probability. It is a structured yet randomized operator that allows information exchange between points. Simple crossover is implemented by choosing a random point in the selected pair of strings and exchanging the substrings defined by that point. Figure 18 shows how crossover mixes information from two parent strings, producing offspring made up of parts from both parents.

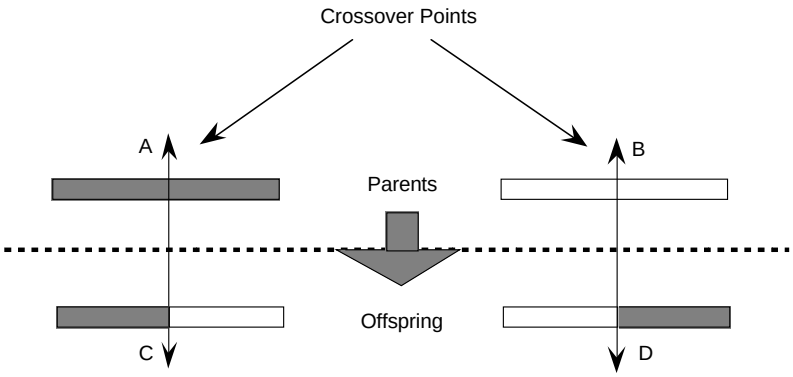

Fig. 18. Crossover of the two parents A and B produces the two children C and D. Each child consists of parts from both parents which leads to information exchange.

4.2. Evolutionary combination in design

When using the concepts embodied in genetic algorithms it is necessary to distinguish the genotype where the genetic material is represented from the phenotype which is the structure resulting from the expression of the genotype in some form. In natural genetics the genotype contains the chromosomes with their constituent genes and the phenotype is the resulting organism. One of the challenges in using genetic algorithms in design is to find an appropriate representation for the genotype. A number of researchers have suggested that shape grammars may be a useful representation.

Shape grammars were introduced into the architectural literature as a formal method of shape generation. They provide a recursive method for generating shapes and are similar to phrase structure grammars, but defined over alphabets of shapes and generate languages of shapes (Stiny and Gips, 1978). A set of grammatical rules map one shape into a different shape. These rules define the set of possible mappings or transformations. More formally, a shape grammar is the quadruple (V_t, V_m, R, I). Where V_t is a set of terminal shapes or terminals and V_m a set of nonterminal shapes or markers. V_t and V_m provide the primitive shape elements of a shape grammar. R is a set of rules consisting of two sides, each side of which contains members of $V_t \cup V_m$. If the left hand side of a rule matches a shape, applying the rule results in replacing the matching shape with the right hand side of the rule. I is the initial shape, a subset of $V_t \cup V_m$ and starts the shape generation process. This models a design system where the rules embody generalized design knowledge and a sequence of rule applications generates a design.

We model routine design with a fixed set of shape grammar rules and encode the possible execution order (application sequence) of these rules for manipulation by the genetic algorithm. The set of optimal structures

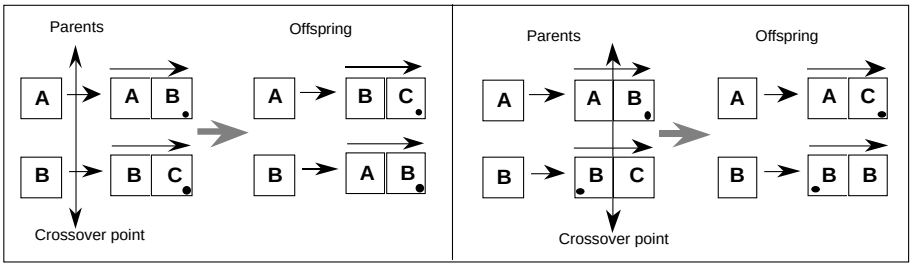

Fig. 19. Generating new grammar rules by crossover, different crossover points produce different offspring rules.

for this fixed grammar defines a space of feasible solutions corresponding to a space of behaviours. The goal is to find the execution order of the grammar rules which will optimize a set of behaviours. In this fixed scheme, additive and substitutive processes are absent. However, when in addition to the application sequence, we allow the grammar itself to be encoded for manipulation by the genetic algorithm, we evolve new grammars and associated rule application sequences to improve on the best possible designs that could be generated by the fixed grammar. That is, instead of optimizing a plan of application of some fixed set of rules, the computational model evolves new rules and optimizes application sequences for those new rules to generate novel and 'more optimal' solutions. By more optimal we mean that the optimal behaviours produced by the application of the new rules are better than those produced by the application of the original rules.

The primary process in genetic algorithms is crossover which combines parts of the genotypes of the two parents-this is the basis of evolutionary combination in design.

4.3. Process model of evolutionary combination

In order to change the state space the grammar needs to be changed. New grammars are generated from the original grammar, through mutation and crossover of rules. That is, rules from the original grammar serve as a basis for the generation of new grammar rules and are produced by cutting and splicing, i.e. combining, the original rules. Consider the two rules labeled 'parents' in Figure 19, choosing the crossover point as indicated produces the new rules labeled 'offspring.' Different crossover points produce different 'offspring' rules leading to a number of different grammars. Recombination therefore plays an important part in the process of learning, helping to generate different grammars and thus different structures and behaviour spaces to explore.

When modeling a design process using a (shape) grammar and its associ-

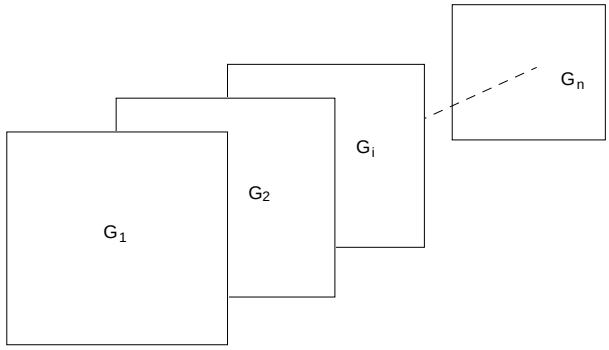

Fig. 20. New state spaces generated when the grammar G_i is evolved into the grammar G_{i+1}.

ated language, we are restricted by the choice of grammar. The design task in this situation is a planning task: to plan a sequence of rule applications that will generate a desired behaviour from the resultant shape. In this case the structure and behaviour spaces are fixed and defined by the grammar. We can encode the task for the genetic algorithm by numbering the rules and representing an individual as a finite length string of these numbers. However, such an encoding can result in nonviable individuals since a rule application called for in an individual may not match its left hand side with any part of the shape generated so far. Instead we use an encoding where the interpretation is context dependent. We allow the grammar itself to evolve while generating a sequence of rule applications. Since every grammar defines a state space, the genetic algorithm now explores a number of structure and behaviour spaces in parallel. This expands the number of possible designs and in the case of our system, produces better, more optimal shapes and associated behaviours that were not possible before.

4.4. Evolutionary combination as exploration in design

Since every grammar defines a state space, evolutionary combination which produces new grammars explores a number of structure and behaviour spaces in parallel, Figure 20.

Evolutionary combination can be extended through the use of mutation. In genetic algorithms all mutations are homogeneous, i.e. they produce rules which could possibly have been produced by evolutionary combination. However, as discussed earlier it is possible to produce mutations which are heterogeneous, i.e. they produce rules which could not possibly have been produced by evolutionary combination. Such a situation opens up another dimension of exploration since it requires additional knowledge in order to produce the phenotype and possibly check for its behaviour.

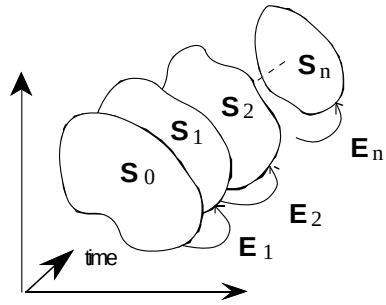

Fig. 21. Graphical model of state space exploration where S_i is state space i and E_j is exploration j.

5. Towards a Model of Exploration in Computer-Aided Design

We have distinguished the two concepts of search and exploration. Search may be treated as the process of looking for appropriate states in a state space conception of the design. Exploration may be treated as the process which determines and to an extent produces the state space within which to explore. In some sense then exploration may be conceived of as meta-search in that in computational terms all the state spaces which could possibly be produced by a set of exploration processes is determined a priori by the initial state space and those processes.

A simple model of exploration in computer-aided design can be graphically presented as in Figure 21 where new state spaces are produced by exploration processes and these state spaces are searched once they are produced.

Let S_i be the state space i which is defined by the variables and their relationships within it. In more general terms it could be defined by a set of schemas over those variables as well as the variables themselves. Let E_j be the exploration processes used to change S_{j-1} into S_j. Then a simple statement of the exploration model is

$$S_j = E_j(S_{j-1})$$

The state spaces are defined in terms of function, behaviour and structure although the emphasis in this paper has been on exploring via structure. Elsewhere, we have suggested how 'new' behaviours may be introduced (Gero and Maher, 1992).

References

Cariani, P.: 1992, Emergence and artificial Life, *in* Langton, C., Taylor, C., Farmer, J. D. and Rasmussen, S. (eds), *Artificial Life II*, Addison-Wesley, Reading, Massachusetts, pp. 775–797.

de Bono, E.: 1972, Personal communication, Forrest, S. (ed.), *Emergent Computation*, Elsevier, New York.

Ganeshan, R., Finger, S. and Garrett, J.: 1991, Representing and reasoning with design intent, *in* Gero. J. S. (ed.), *Artificial Intelligence in Design '91*, Butterworth-Heinemann, 737–755.

Garcia, A. C. B. and Howard, C.: 1991, Building a model for augmented documentation, *in* Gero, J. S. (ed.), *Artificial Intelligence in Design '91*, Butterworth-Heinemann, 723–736.

Gero, J. S.: 1987, Prototypes: a new schema for knowledge-based design, *Working Paper*, Architectural Computing Unit, Department of Architectural Science, University of Sydney, Sydney.

Gero, J. S.: 1990, Design prototypes: a knowledge representation schema for design, *AI Magazine*, **11**(4): 26–36.

Gero, J. S. and Kumar, B.: 1993, Expanding design spaces through new design variables, *Design Studies* **14**(2): 210–221.

Gero, J. S. and Maher, M. L.: 1992, Mutation and analogy to support creativity in computer-aided design, *in* Schmitt, G. N. (ed.), *CAAD Futures '91*, Vieweg, Wiesbaden, pp. 261–270.

Gero, J. S. and Yan, M.:1993, Shape emergence by symbolic reasoning, *Working Paper*, Key Centre of Design Computing, University of Sydney, Sydney.

Goldberg, D. E.: 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading.

Langton, G. L.: 1989, *Artificial Life*, Addison-Wesley, Reading.

Logan, B. and Smithers, T.: 1989, The role of prototypes in creative design, *Preprints Modeling Creativity and Knowledge-Based Creative Design*, University of Sydney, Sydney, Australia, 233–248.

Qian, L. and Gero, J. S.: 1992, A design support system using analogy, *in* Gero, J. S. (ed.), *Artificial Intelligence in Design '92*, Kluwer, Dordrecht, pp. 795–813.

Riesbeck, C. and Schank, R.: 1989, *Inside Case-Based Reasoning*, Lawrence Erlbaum, Hillsdale, New Jersey.

Rosenman, M. A.: 1991, Incorporating intent in design data exchange standards, *in* Gero, J. S. and Sudweeks, F. (eds), *Preprints IJCAI-91 Workshop on Artificial Intelligence in Design*, University of Sydney, Sydney, pp. 51–56.

Rosenman, M. A., Gero, J. S. and Hwang, Y-S.: 1993, Representation of multiple concepts of a design object based on multiple functions, *Management of Information Technology for Construction* (to appear).

Smithers, T.: 1992, Design as exploration: puzzle-making and puzzle-solving, *AID'92 Workshop on Search-Based and Exploration-Based Models of Design Process* (available from the Department of Artificial Intelligence, Edinburgh University), pp. 1–21.

Steadman, P.: 1979, *The Evolution of Designs*, Cambridge University Press, Cambridge.

Steels, L.: 1991, Towards a theory of emergent functionality, *in* Meyer, J.-A. and Wilson, S. W. (eds), *From Animals to Animats*, MIT Press, Cambridge, pp. 451–461.

Stevens, S. S.: 1957, On the psychophysical law, *Psychological Review*, **14**: 153-181.

Stiny, G. and Gips, J.: 1978, *Algorithmic Aesthetics: Computer Models for Criticism and Design in the Arts*, University of California Press, Berkeley and Los Angeles, California.

Weisberg, R. W. and Alba, J. W.: 1981, An examination of the alleged role of 'fixation' in the solution of several 'insight' problems, *Journal of Experimental Psychology*, **110**(2): 169–192.

Woodbury, R. F.: 1989, Design genes, *Preprints Modeling Creativity and Knowledge-Based Creative Design*, University of Sydney, Sydney, pp. 133–154.