Improving Pareto Optimal Designs Using Genetic Algorithms

John S. Gero
Key Centre of Design Computing
Department of Architectural and Design Science
University of Sydney
NSW 2006 Australia
john@arch.su.edu.au

Sushil J. Louis
Department of Computer Science
Mackay School of Mines
University of Nevada
Reno, NV 89557
sushil@cs.unr.edu[1]

Abstract

Pareto optimal designs are the best designs that can be produced for a given problem formulation for a given set of criteria when the criteria are not combined in any way. If the goal is to improve the performance in those criteria then it is possible to manipulate the problem formulation to achieve an improvement. The approach adopted is to encode the formulation in a genetic algorithm and to allow the formulation to evolve in the direction of improving Pareto optimal designs. A set of rules (in the form of a shape grammar), the execution of which produces a design, is encoded as the genes in a genetic algorithm. However, the rule set is allowed to evolve, not just the order of execution of rules. We present an example demonstrating both the approach and its utility in improving Pareto optimal designs.

1 Introduction

Design concerns the development of the description of an artifact to satisfy a set of requirements. These requirements can be cast as objectives which point toward directions of

[1]This work was carried out at the Key Centre of Design Computing while on leave from the Department of Computer Science, Indiana University.

improving behavior. When there are multiple objectives or criteria to be optimized, Pareto optimality provides a principled means of choosing directions of improving behavior. Pareto optimality follows the principle of non-dominance of solutions and produces a Pareto optimal design such that there are no other designs superior in all criteria. This paper aims to improve Pareto optimal designs.

The problem of design optimization – multi-criteria or single criterion – can be cast as search through a state space of viable designs. A design that is optimal in one state space may no longer be optimal in a changed state space. Changing the state space corresponds to a change in the problem formulation and can lead to the discovery of "more"-optimal designs. This paper improves Pareto optimal designs by allowing the state space itself, as well as the designs subsumed by the state space, to evolve in the direction of improvement. A genetic algorithm encodes both the problem formulation and the possible design solutions circumscribed by the problem formulation and searches through an evolving state space for Pareto optimal designs. Note that the Pareto optimal design found at the end of the computation may be a "more"-optimal design than any that were circumscribed in the original problem formulation.

We start with the genetic algorithm searching a space in which the problem formulation is fixed. Next, the problem formulation is allowed to evolve during the search for improved solutions. In other words the genetic algorithm searches through possible problem formulations while searching for a good solution. In our sample two-criteria optimization problem, designs that could not be described within the original problem formulation arise over the course of the genetic algorithm's search leading to new and improved designs.

The next section discusses the different kind of changes that can be made to state spaces clarifying the concepts of additive and substitutive state space changes. We present the sample problem used in this paper and introduce shape grammars in section three. Genetic algorithms provide the computational framework for changing state spaces while searching for new improved designs and are introduced in section four. The last two sections present and discuss results and provide conclusions and directions for further research.

2 Optimization and Design State Spaces

The framework of design prototypes divides a state space into three subspaces corresponding to a space of structures, a space of behaviors associated with structures and a space of functions associated with behaviors.[2] Figure 1 shows the mappings between these subspaces. Optimizing a design can be represented as a process which changes one of the subspaces

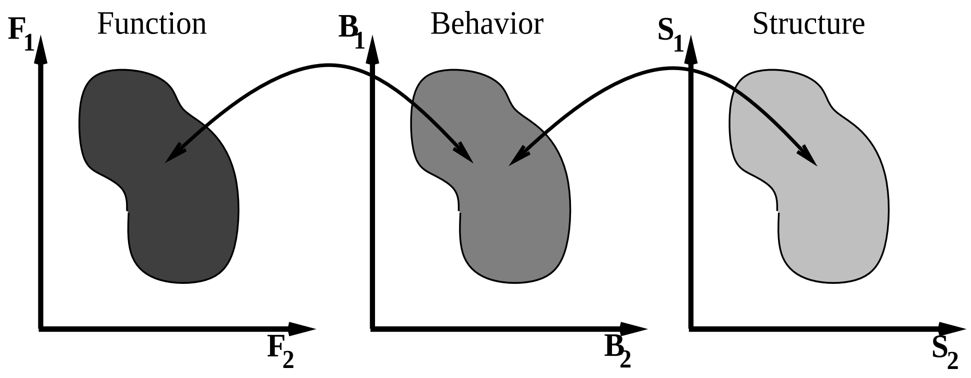

Figure 1: Structure, Behavior and Function spaces and the mappings between them.

(usually the structure space is changed). There are two interesting classes of change to state spaces: addition and substitution. Addition, which can be achieved by adding (design) variables, results in a new state space, S_n, that subsumes the original space, S_0, that is: $S_0 \subset S_n$ and $S_0 \bigcap S_n \neq \phi$ (figure 2). Substitution, which involves the replacement of some variables by other variables, results in a space, S_n, that is different from the original, S_0, such that $S_0 \not\subset S_n$ (figure 3).

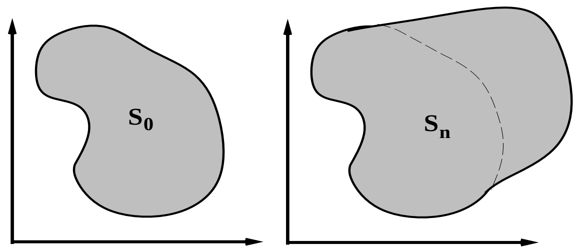

Figure 2: The effect of additive processes on a state space.

In this paper we model a substitutive process using a genetic algorithm to change design variables and search for an optimal design in the changed state space. The results in section 5 indicate that designs that are better than the optimal design in the original state space can be found.

The following sections introduce the sample problem, shape grammars, genetic algorithms and show how to encode the sample problem for genetic optimization.

3 A sample optimization problem

Consider the problem of generating the optimal topology of a beam section. Our two optimization goals are: 1) to minimize the perimeter (decrease surface area) and 2) to maximize the moment of inertia of the beam section (second moment of area) to improve its stiffness. The two criteria have been chosen such that a tradeoff needs to be made. A change for the better in one criteria usually leads to a change for the worse in the other. We use a *shape grammar* (described below) to embody the set of rules used to generate possible cross sections. Following a fixed length sequence of such rules generates a beam section. The moment of inertia and perimeter can now be calculated and used in the optimization process (see section 5).

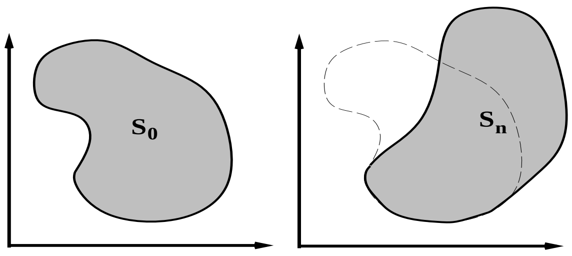

Figure 3: The effect of substitutive processes on a state space.

3.1 Shape grammars

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.[7] 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 $V_t \bigcup V_m$. 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 $V_t \bigcup V_m$ 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.

3.2 An example shape grammar

A shape grammar for generating beam sections is displayed in figure 4. Each rule has a left hand side (lhs) and a right hand side (rhs) separated by an arrow. When applying a rule we replace the cell corresponding to the lhs by the rhs of that rule. For example, starting with a cell labeled $\boxed{\text{A}}$ and applying rules $0, 2, 3$ in sequence would result in a beam section that looked like the one in figure 5.

Thus the finite rule sequence $0, 2, 3$ gives rise to a shape that can be interpreted as a beam section with an associated moment of inertia and perimeter. Cast in this way, the problem becomes the generation of a rule sequence that will optimize our two criteria.

With a fixed shape grammar (figure 4) and a fixed length sequence of rule applications, we are searching in a finite state space. Points in the state space represent beam section topologies, and we are searching for a topology that maximizes moment of inertia and minimizes perimeter.

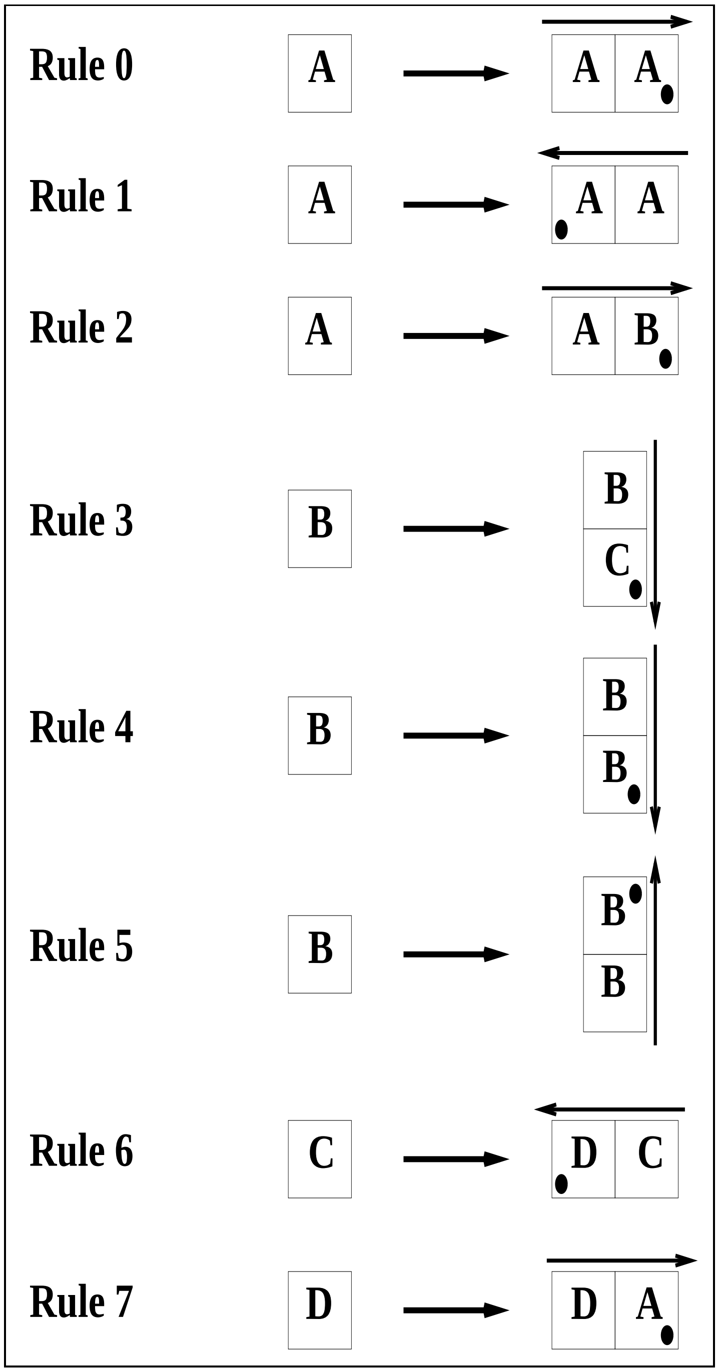

Figure 4: These eight rules comprise the fixed shape grammar.

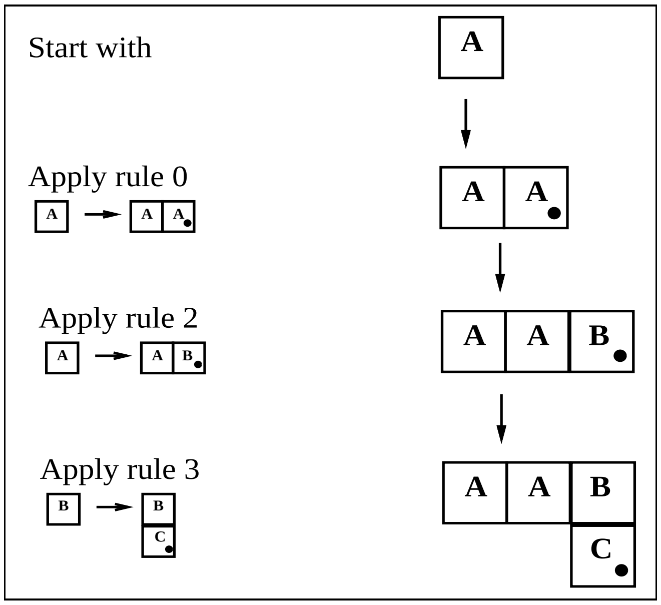

Figure 5: Applying rules 0, 2, 3 in sequence.

This is a search problem and as such, well suited for a genetic algorithm. The next section describes how to encode the problem for a genetic algorithm.

4 Genetic Algorithms

Genetic algorithms (GAs), originally developed by Holland,[4] model natural selection and the process of evolution. Conceptually, GAs use the mechanisms of natural selection in evolving individuals that, over time, adapt to an environment. They can also be considered a search process, searching for better individuals in the space of all possible individuals. In practice, individuals represent points in a state space, while the environment provides a measure of "fitness" that helps identify better individuals. Genetic algorithms are a robust, parallel search process requiring little information to search effectively. They have been used in function optimization since their inception, optimizing large poorly-understood problems that arise in many areas of science and engineering.[3,1]

As already noted, the motivational idea behind GAs is natural selection implemented through selection and recombination operators. 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.

```
$t = 0$
**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 stochastic 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 6 shows how crossover mixes information from two parent strings, producing offspring made up of parts from both parents. We note that this operator which does no table lookups or backtracking, is very efficient because of its simplicity.

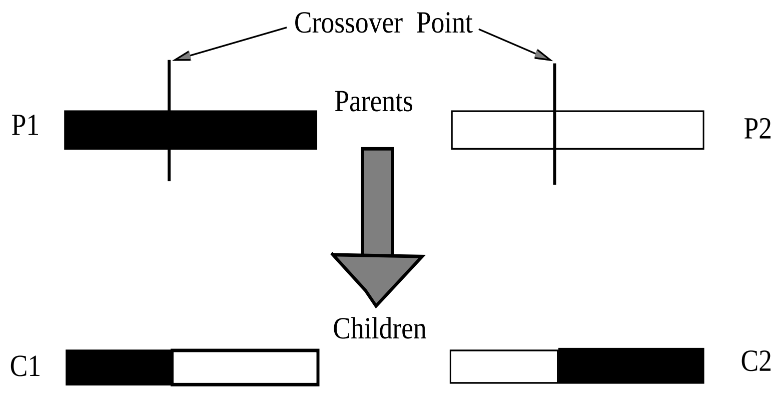

Figure 6: Crossover of the two parents A and B produces the two children C and D. Each child consists of parts from both parents which leads to information exchange.

Holland's fundamental theorem of genetic algorithms leads to the building block hypothesis which states that genetic algorithms work near-optimally by *combining* certain types of *building blocks* corresponding to partial solutions or designs.[3]

Since GAs work with binary strings, we describe an encoding used by the genetic algorithm on our two criteria optimization problem.

Consider the grammar in figure 4. There are four classes of composition rules, one associated with each of the cells labeled A, B, C, D. In total there are eight rules in these four classes. In our first attempt at an encoding, a fixed length sequence of these rules forms the genotype. Thus each member of the genetic algorithm's population is a string of (rule) numbers representing a rule application sequence. Applying the rules in sequence from left to right, produces a topology. Since genetic algorithms prefer binary strings, we then transform the rule application sequence into a binary string.

Consider an example in which we fix the number of rule applications at 3, then the string

$$2, 3, 6$$

corresponds to successively applying rules $2, 3$ and 6. Before we convert this into a binary string, the question of how many binary digits, or bits, are required for each decimal number

Decimal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Table 1: Mapping between decimal and three bit binary numbers

needs to be answered. Each position in the string can have values ranging from 0 to 7 corresponding to each of the rules in the grammar (figure 4). Therefore we use three (3) bits to represent the 8 possible rules ($2^3 = 8$). The mapping of the (decimal) numbers between 0 and 7 and the equivalent binary numbers is given in table 1.

The string $2, 3, 6$ which is interpreted as applying rule 2 followed by applying rule 3 and then rule 6, would be represented as the binary string

$$010011110$$

where each group of three bits represents a rule. Thus the first three bits (counting left to right) $010 = 2$ in decimal represents rule 2. The next three bits $011 = 3$ in decimal represents rule 3 and the last group of three bits $110 = 6$ in decimal represents rule 6. The binary string

$$001010011$$

represents another sequence of rule applications. With these strings as parents, crossover can now produce *non-viable* offspring as shown below

$$\begin{array}{lll} \text{Parents} & 0100111 \mid 10 & 0010100 \mid 11 \\ & \Downarrow & \\ \text{Offspring} & 0100111 \mid 11 & 0010100 \mid 10 \end{array}$$

Here the first offspring $010, 011, 111$ (for the convenience of the reader, the commas have been inserted to separate the groups of three bits) is not viable because rule $111 = 7$ cannot be applied after rule $011 = 3$ since the left hand side of rule 7 ($\boxed{\text{D}}$) does not match the right hand side of rule 3 ($\boxed{\text{C}}$).

In general non-viable offspring are produced when crossover results in individuals that do not belong to the search space of interest. In the context of grammars, if an individual represents a sequence of rule applications resulting in a shape, it may call for the application of a rule that is inappropriate in the current context. There are a few approaches to tackling problems of this kind. We choose to change the encoding to one in which non-viable offspring cannot occur.

In the new encoding, we let the interpretation of a number in the string *depend* on the right hand side of the last applied rule. (In the following discussion we use the numbers **0 - 7** in boldface to identify the 8 rules of our shape grammar in figure 4.) In our new encoding the first number in the string specifies the starting label, that is, whether the starting cell is of type $\boxed{\text{A}}$, $\boxed{\text{B}}$, $\boxed{\text{C}}$ or $\boxed{\text{D}}$. The remaining numbers specify an application sequence. Consider the string $0, 2$. The first number 0 fixes the starting cell as a cell of type $\boxed{\text{A}}$, The second number 2, now refers to the second rule *with an* $\boxed{\text{A}}$ *on the left hand side*. Therefore, in this encoding the number 2 in the second position of the string, refers to rule **1**, that is,

$$\boxed{\text{A}} \longrightarrow \overset{\longleftarrow}{\boxed{\bullet\ \text{A}}}\boxed{\text{A}}$$

Thus the interpretation of the numbers as rules is context dependent. A number's interpretation depends on the result of the previous rule's application. This context dependent encoding is summarized below.

1. if an $\boxed{\text{A}}$ is the match for the left hand side

- 0 and 3 $(00, 11$ in binary) refer to rule **2**

$$\boxed{\text{A}} \longrightarrow \overset{\longrightarrow}{\boxed{\text{A}}\boxed{\text{B}\bullet}}$$

- 1 (01 in binary) refers to rule **0**

$$\boxed{\text{A}} \longrightarrow \overset{\longrightarrow}{\boxed{\text{A}}\boxed{\text{A}\bullet}}$$

- 2 (10 in binary) refers to rule **1**

$$\boxed{\text{A}} \longrightarrow \overset{\longleftarrow}{\boxed{\bullet\ \text{A}}\boxed{\text{A}}}$$

2. if a $\boxed{\text{B}}$ is the match for the left hand side

- 0 and 3 $(00, 11$ in binary) refer to rule **3**

$$\boxed{\text{B}} \longrightarrow \begin{array}{c}\boxed{\text{B}}\\ \boxed{\text{C}\bullet}\end{array} \downarrow$$

- 1 (01 in binary) refers to rule **4**

$$\boxed{\text{B}} \longrightarrow \begin{array}{c}\boxed{\text{B}}\\ \boxed{\text{B}\bullet}\end{array} \downarrow$$

- 2 (10 in binary) refers to rule **5**

$$\boxed{\text{B}} \longrightarrow \begin{array}{c}\boxed{\text{B}\bullet}\\ \boxed{\text{B}}\end{array} \uparrow$$

3. if a $\boxed{\text{C}}$ is the match for the left hand side

- $0 - 3$ (00 to 11 in binary) all refer to rule **6**

$$\boxed{\text{C}} \longrightarrow \overset{\longleftarrow}{\boxed{\bullet\ \text{D}}\boxed{\text{C}}}$$

4. if a $\boxed{\text{D}}$ is the match for the left hand side

- $0 - 3$ (00 to 11 in binary) all refer to rule **7**

$$\boxed{\mathrm{D}} \longrightarrow \overrightarrow{\boxed{\mathrm{D}}\boxed{\mathrm{A}\bullet}}$$

With this encoding not only do we solve the problem of non-viable offspring we also reduce the number of bits needed to encode a shape as we use only two bits per rule instead of three. Figure 7 depicts the mapping from a rule application sequence to a shape using this encoding.

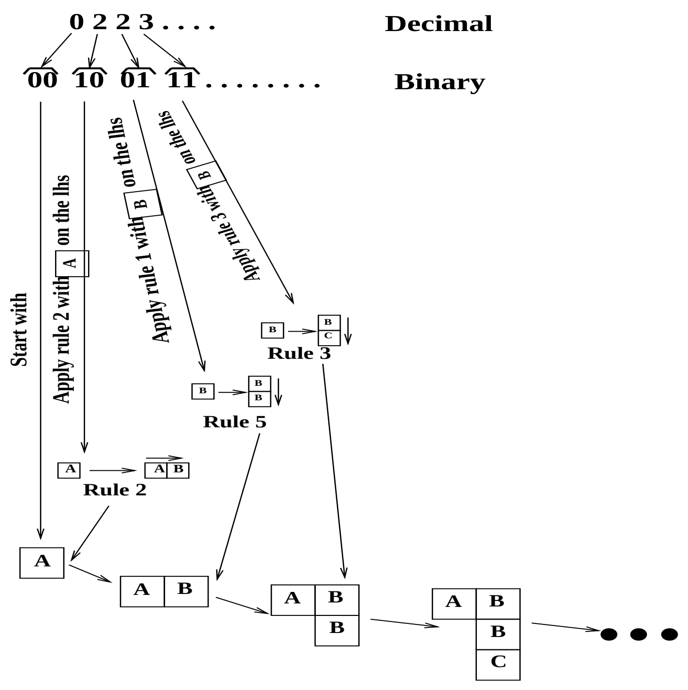

Figure 7: How the genetic algorithm interprets a binary string to generate a topology for the beam section

5 Results

We start with finding the topology that will optimize our two criteria for a fixed shape grammar. The set of optimal structures for this fixed grammar defines a attainable criteria set corresponding to a space of behaviors. The goal is to find the execution order of the grammar rules which will optimize a set of behaviors. In this fixed scheme, additive and substitutive processes are absent.

The grammar is given in figure 4. Thus starting with a label and selectively applying the rules, we can arrive at a topology for a beam section after a predetermined number of rule applications. From this grammar, shapes of interest can be derived after *nine* rule applications, we therefore fix the number of rule applications at nine. Each of the cells in the grammar has unit length and unit width. Weights, associated with labels, figure in the calculation of the moment of inertia simulating larger cross sectional areas, or different materials and are given below:

- $A \Rightarrow 1\ unit$
- $B \Rightarrow 2\ units$
- $C \Rightarrow 3\ units$
- $D \Rightarrow 4\ units$

Using the weights and cell dimensions, we can calculate the perimeter and moment of inertia of the shapes generated by the genetic algorithm. The problem is therefore to find the topology of a beam section such that the following two criteria are optimized:

- The moment of inertia (second moment of area) of the beam section which has to be maximized.The moment of inertia is defined about the center of the beam section and is given by
$$MI = \sum_{\text{all cells}} \frac{(\text{area of cell}_i)(\text{weight of cell}_i)}{(\text{distance of cell}_i\text{ from center})^2}$$
- The perimeter of the beam section which has to be minimized.

subject to the constraint that only nine cells be used.

5.1 Optimization in a fixed state space

The genetic algorithm starts with a randomly generated initial population of size 50. We select two random individuals in the population for mating and recombine them to produce two offspring. The set of two parents and two offspring form the set of solutions from which we find the non-dominated solutions (the Pareto set) through exhaustive testing. Two individuals from this Pareto set participate in the next generation of the genetic algorithm. The GA completes a generation when repeated recombination and selection fills up the fixed size population of 50. 100 generations are sufficient for the genetic algorithm to converge. Since the GA is a probabilistic algorithm we report results averaged over a number of runs of the GA with different random seeds. Results from a number of runs are usually used in studies of genetic algorithm applications since the algorithm is probabilistic in nature and a single run may not provide a true picture. The population size was chosen based on the computing resources available and the need to have sufficient diversity at initialization (see Louis and Rawlins[5] for details on population sizing and convergence time).

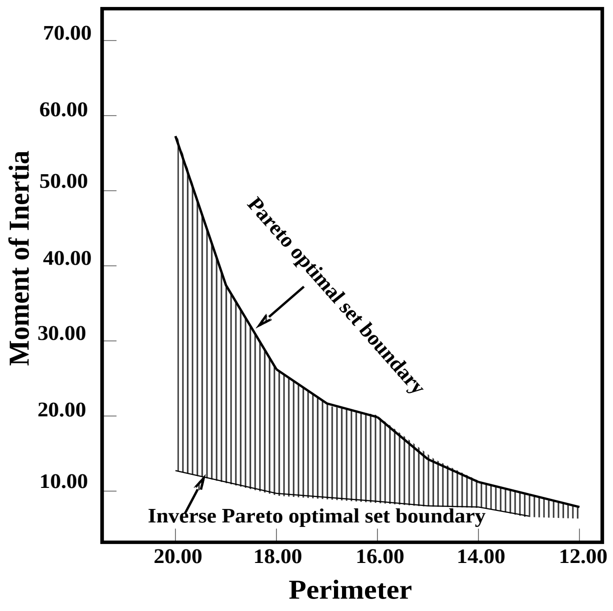

Figure 8: The attainable criteria set for a fixed grammar. This corresponds to optimizing with a fixed problem formulation.

The results of the optimization by the genetic algorithm are depicted graphically in figure 8. The Pareto optimal boundary is the line connecting the points of the Pareto set produced at the end of computation. Similarly, the inverse Pareto optimal boundary connects the inverse of the Pareto optimal set, i.e., the solutions obtained by reversing the signs of the criteria and optimizing again.[6] The space bounded by the Pareto optimal set and its inverse is the feasible solution space and is often called the *attainable criteria set*. This is the shaded region in figure 8.

5.2 Optimization in an evolving state space

To find "more"-optimal solutions, we allow the problem formulation to change by allowing the shape grammar to evolve in the direction of improving behavior. New grammars are generated from the original grammar through mutation and crossover of rules. That is, rules from the original grammar can serve as a basis for the generation of new grammar rules and are produced by cutting and splicing the original rules. Consider the two rules labeled "parents" in figure 9, choosing a crossover points as indicated produces the rules labeled "offspring". Different crossover points produce different "offspring" rules leading to a number of different grammars. Those grammars that are used to produce better (according to our two criteria) structures proliferate along with the structures. Recombination therefore allows the restructuring of the problem formulation, playing an important part in generating structures that were not possible with the original problem formulation and that are "more"-optimal.

Neither the commencing grammar nor the length of the rule application sequence is significant. The ability to start with any grammar and *evolve new grammars* is the significant concept. The insight here is that *changing the problem formulation* can produce "more"-optimal solutions. We chose to use genetic algorithms to implement this insight because, unlike other search algorithms, they allow the *combination* of partial solutions into more complete solutions.

When we allow the grammar itself to evolve, we need to encode the rules within the genotype. Once again we chose an encoding that does not allow non-viable offspring. Making the grammar implicit, we encode a sequence of directions (up, down, left or right) and labels (A, B, C, or D) to specify an individual. This added flexibility means we require 6

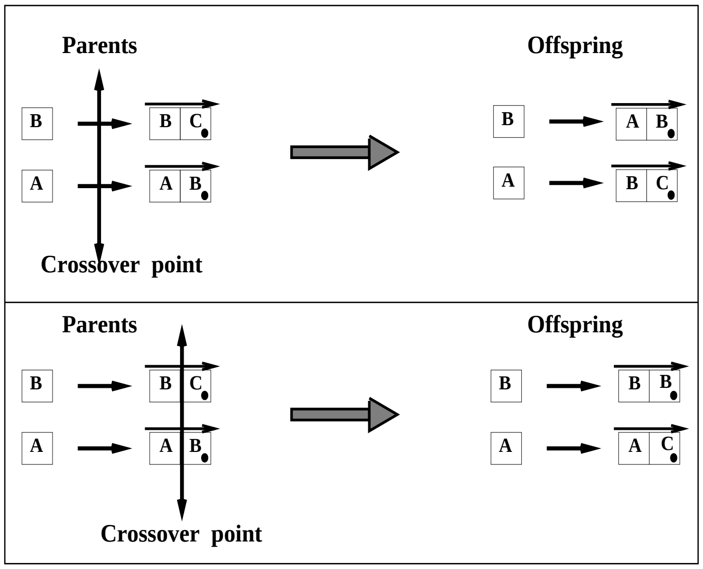

Figure 9: Generating new grammar rules by crossover.

bits to specify the next move in generating a shape, 2 bits for direction and 2 bits each for a label. The rules that we use are now of the form:

$$\boxed{X} \longrightarrow \begin{array}{ccc} & \uparrow & \\ \leftarrow & & \rightarrow \\ & \downarrow & \end{array} \boxed{Y}\ \boxed{Z}$$

where the cell label X is specified by the last applied rule, while the direction and new cell labels Y and Z are specified in the current move. In our encoding, we use the numbers from 0 to 3 to specify the four types of cells

$$\begin{array}{l} 0 \longrightarrow \boxed{A} \\ 1 \longrightarrow \boxed{B} \\ 2 \longrightarrow \boxed{C} \\ 3 \longrightarrow \boxed{D} \end{array}$$

and use the same strategy to specify directions

$$0 \longrightarrow \downarrow$$
$$1 \longrightarrow \uparrow$$
$$2 \longrightarrow \rightarrow$$
$$3 \longrightarrow \leftarrow$$

For example the string

$$0, 2, 3, 1, 1, 2, \ldots$$

specifies the following moves: For initialization, we ignore the first two numbers which specify a direction and a cell label and start off with 3 which represents $\boxed{\text{D}}$. The next three numbers: $1, 1, 2$ tell us to go up (1) after replacing the previous cell ($\boxed{\text{D}}$) by the cell denoted by 1 which is $\boxed{\text{B}}$. Once we have replaced the previous cell we go up (as already indicated) and label the cell $\boxed{\text{C}}$ as indicated by the number 2. We continue in this way for a total of nine moves. The result of the first move is shown below.

$$\begin{array}{ccc} & & \boxed{\text{C}} \\ \boxed{\text{D}} & \Rightarrow & \boxed{\text{B}} \end{array}$$

With this encoding, crossover always results in a legal string and allows changing the problem formulation while optimizing the topologies according to the two criteria.

The genetic algorithm now optimizes both the grammar and the topologies generated according to the two criteria. The Pareto optimal boundary and the inverse Pareto optimal boundary along with the attainable criteria set is shown in figure 10. The attainable criteria sets resulting from optimization with the fixed grammar and the evolving grammar are different. The difference is apparent when we superpose the two sets as in figure 11. The attainable criteria set produced by the evolving grammar contains topologies that are "more"-optimal, with reference to our two criteria, than those produced by the fixed grammar.

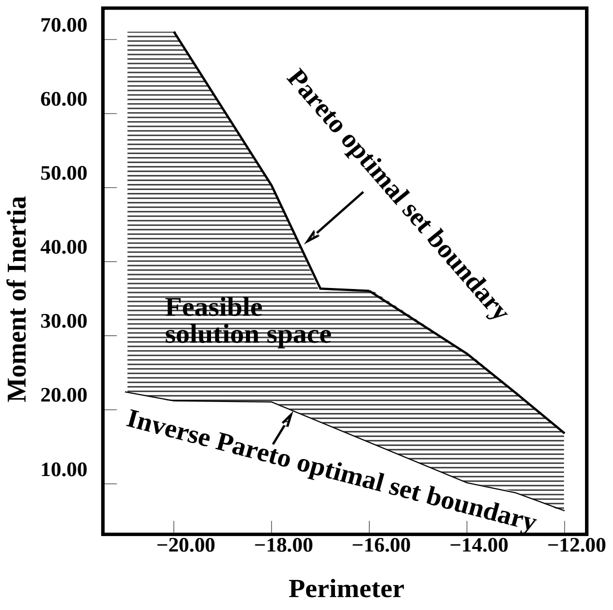

Figure 10: The attainable criteria set for an evolving grammar. This corresponds to allowing the problem formulation to change while optimizing.

6 Discussion

Figure 11 shows the attainable criteria set for our fixed grammar and a grammar that is allowed to evolve. The horizontally shaded area is the attainable criteria set defined by the evolving grammar and the vertical shaded area is the attainable criteria set for the fixed grammar. The two sets are not equal and the attainable criteria set depicting feasible solutions for the evolving grammar shows a marked improvement. Let S_0 denote the behavior space (attainable criteria set) for the structures defined by our fixed grammar (fixed problem formulation), and S_n the attainable criteria set when the grammar is allowed to evolve (evolving problem formulation). Figure 11 indicates that:

$$S_0 \not\subset S_n$$

and

$$S_0 \bigcap S_n \neq \phi$$

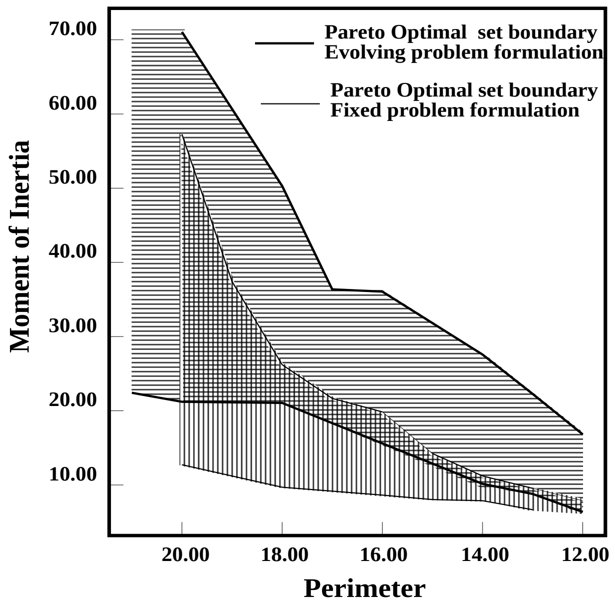

Figure 11: Comparing attainable criteria sets when the problem formulation is fixed and when it is allowed to change.

This corresponds to a substitutive process. All genetic algorithm parameters were the same for the runs with the fixed grammar and the runs with the grammar that is allowed to evolve. The reason for the difference and improvement follows from the observation that the heaviest label D is now allowed to propagate in the vertical direction, thus increasing the moment of inertia. Since perimeter does not depend on weight, but only on shape, the genetic algorithm converges upon grammars that increasingly use D labeled cells, leading to larger values for the moment of inertia. Figure 12 shows one grammar that was learned by this evolutionary process. This grammar is capable of producing better designs than the commencing grammar and corresponds to a new problem formulation. Figure 13 shows some of the beam sections produced during the learning process. These could not be produced with the original grammar.

It is well recognized that there is a nexus between problem formulation and problem solution. Optimization processes depend on a fixed problem formulation although, increasingly, the formulation is being expanded to include more decision variables in the optimization process. This can be seen in the area of structural optimization as an example. The early formulations were concerned with optimizing the cross-sectional area of the component members. This was then expanded to include not just the cross-sectional area but also the geometry of the structure as decision variables. Then, the formulation was further expanded to include the topology of the structure as a a decision variable. The changes in formulation all occurred manually.

What has been presented here is an approach to automatic reformulation of a design optimization problem, a reformulation that guarantees to improve the optimal solution set. In the example presented, neither the commencing grammar nor the length of the rule application sequence is significant. The ability to start with any grammar and evolve new grammars is the significant concept. The insight here is that *changing the problem formulation* can produce "more"-optimal solutions. We chose to use genetic algorithms to implement this insight because, unlike other search algorithms, they allow the combination of partial solutions into more complete solutions. The concept is generic and potentially widely applicable to Pareto optimization design problems that can be cast as evolutionary systems.

Acknowledgements

This work was funded by grants from the Key Centre of Design Computing and the Australian Research Council.

References

1. Adeli, H. and Cheng, N. T. (1994). Integrated genetic algorithm for optimization of space structures, *ASCE Journal of Aerospace Engineering* **6**(4): 315–328.

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

3. Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts.

4. Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbour, Michigan.

5. Louis, S. J. and Rawlins, G. J. E. (1992). Syntactic analysis of crossover in genetic algorithms, *in* D. Whitley (ed.), *Foundations of Genetic Algorithms–2*, Morgan Kaufmann, San Mateo, CA, pp. 141–152.

6. Radford, A. D. and Gero, J. S. (1988). *Design by Optimization in Architecture, Building and Construction*, Van Nostrand Reinhold, New York.

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

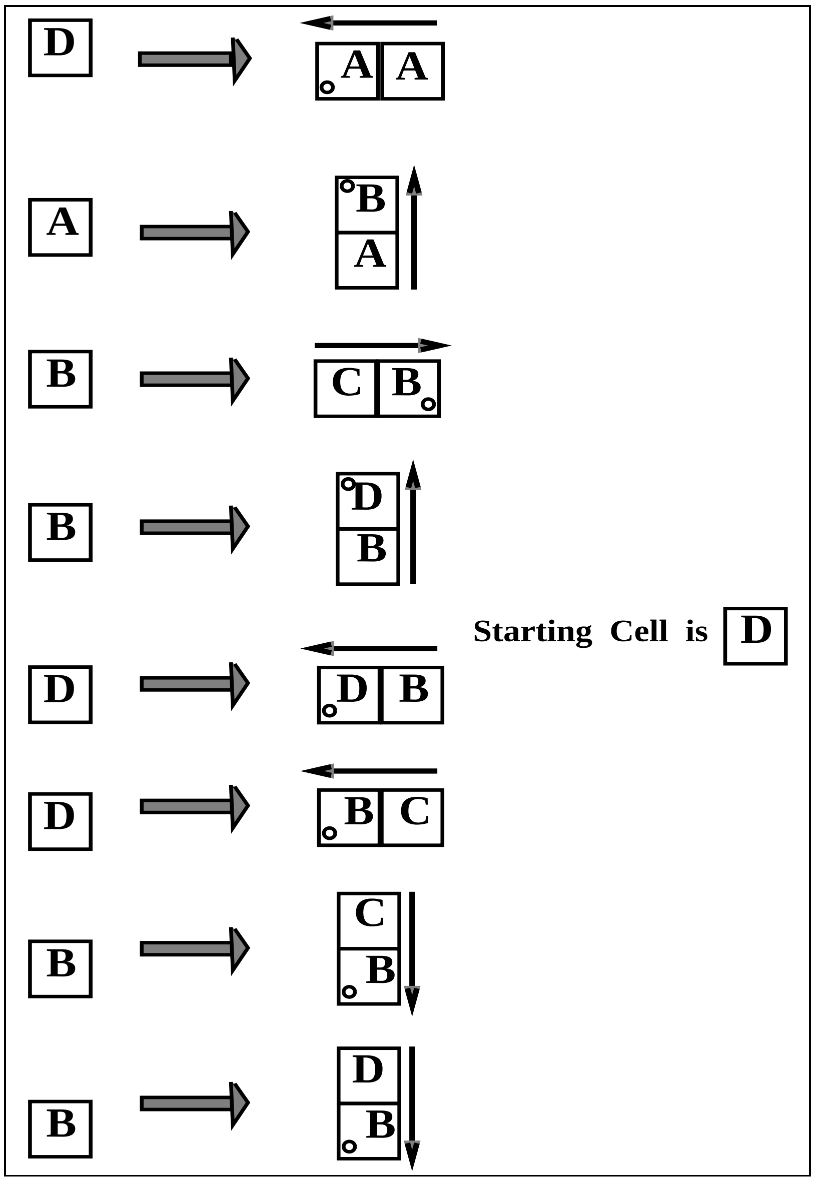

Figure 12: A new problem formulation. A new grammar produced by the genetic algorithm that results in “more”-optimal beam sections.

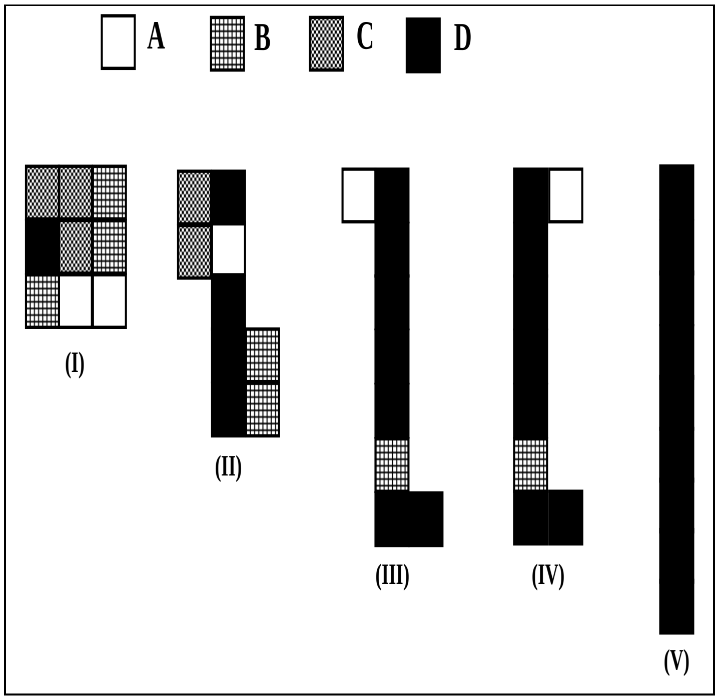

Figure 13: Beam sections produced during optimization while changing the problem formulation