DESIGN TOOLS THAT LEARN: A POSSIBLE CAD FUTURE

John S Gero
Key Centre of Design Computing
University of Sydney
Australia
john@arch.su.edu.au
http://www.arch.su.edu.au/~john

Abstract

This paper describes the concept of tools that learn. These are design support tools that acquire problem-specific knowledge as they are used to solve a problem. This knowledge is then reused in the solution of similar problems. The effect of this is that the tool is more efficient. An example is presented to demonstrate the idea.

1 Introduction

Computer-aided design has moved from meaning computer-aided drafting though computer-aided analysis to design synthesis support aids. Computer-aided drafting provides the means to document a completed or near completed design. It offered a number of advantages over manual drafting. Computer-aided analysis provides a means to determine the behaviours of a completed or partially complete design — you need to have a design before you can analyse it. In structural engineering design the development of matrix methods followed closely by the development of the finite element method provided a formal basis for the development of computer-aided analysis tools of considerable power. Other tools based on artificial intelligence techniques became available, tools such as code conformance checkers. Once analysis could be automated it became possible to develop design synthesis support aids. These tended to be aimed at parametric or routine design. In structural engineering we saw member design become automated. Once the context had been established the designs of frames and trusses could be automated. More recently we have seen other synthesis tasks having direct computer support.

In general, where the design decisions are based on agreed evaluations such as code conformance and direct cost then human designers have allowed those design subtasks to be automated using design synthesis tools. This has produced what we might call 'passive' computer-based design aids, passive in the sense that the designer does not interact with them. Such design aids produce design decisions which most human designers would also produce since the tasks they deal with are sufficiently circumscribed to minimise the range of possible solutions which could be of interest given the agreed evaluators. In one sense these aids are the computational counterpart of design tables which were used extensively for member design previously.

What all of these aids: computer-aided drafting tools, computer-aided analysis tools and 'passive' computer-based design aids have in common is that each tool is unchanged by its use. Every time to commence a new set of drawings on a computer-aided drafting system it takes the same effort. Every time you want to run a finite element analysis the program takes the same set of potential paths. Every time you want to synthesise a reinforced concrete column for a particular set of boundary conditions the program takes the same effort. There are clear and obvious benefits in having the tools unchanged by their use as this makes them independent of their use and they can be used with any arbitrary problem.

However, I claim that there are also significant disadvantages in having the tools unchanged by their use. Each design that a designer works on adds to the experience of the designer, in this sense the designer learns from each design. For example, the designer in analysing a structure using finite element techniques may find that a certain type of element in a particular set of configurations produces much better results than the standard element. A designer working on a hospital layout using a computer-based layout planning aid may find that certain layouts of spaces produce better results than others. When each of these designers next tackles a similar design task it would be useful if the same tools now had knowledge about the fact that a

certain type of finite element in a particular set of configurations produces much better results than the standard element and that certain layouts of spaces produce better results than others. The effect of this would be tools which are increasingly useful to the designer. In order for this to occur the tools would have to learn which are the beneficial aspects of the designs being produced. There is some relationship to this idea with case acquisition in case-based design systems [1] and in systems which utilise constantly reinforced neural networks [2].

2 Tools that learn

2.1 Learning

Learning implies the acquisition or restructuring of knowledge rather than the simple acquisition of facts. Computer-based or machine learning is comprised of four principal paradigms [3]: the inductive paradigm, the analytic paradigm, the genetic paradigm and the connectionist paradigm. All of these paradigms are used within a shared view of the role of machine learning in design: namely that of "learning to perform existing tasks better using available tools" where the tools themselves are unchanged. Here we want to use learning as a means to change the tool so that the tool performs better in a defined environment. Learning is always concerned with improving the quality of the knowledge that is to be used. A number of different teleologies drive machine learning, not all of which are directly applicable in design. The foci in machine learning in design include: concept formation [4], learning high-level relationships [2] and increasing the efficacy of the available knowledge [5].

Here we are particularly interested in the second of these foci – learning high-level relationships which exist in the solutions to design problems which are useful in producing the solutions' beneficial performance. If we can learn these then when we reuse the tool it will have different knowledge in it than before, knowledge which should make it easier for the tool to produce better preforming solutions more easily.

2.2 How can tools learn?

All computational systems, at some basic level, can be viewed as being comprised of two components: representation and process. Thus, tools can learn this useful design information either by restructuring their representations or by modifying their processes. Here we will concentrate on the former, although modifying processes is likely to be equally or more rewarding.

Take the case of *evolutionary systems* exemplified by genetic algorithms [6] where the representation is distinctly different at different stages of the solution process. At the base level is the genotype which is a collection of genes. The genotype is translated into the phenotype which is the direct representation of the design, ie the 'structure' of the design. The phenotype is then translated into a performance in the system's fitness, ie the 'behaviour' of the design. The performance in the fitness controls aspects of the genetic processes related to the genotype in the next generation. As in natural systems, the genes in design systems appear to bear no direct relation to the phenotype itself, ie they do not look in any way like the structure of a design and it is the mapping process which embodies them. This multilevel representation is extremely important in nature as it prevents disturbances to the phenotype (the natural organism) from being propagated to future generations. For example if you lose your arm in an accident, your children are not born missing that arm. For different reasons the multilevel representation is particularly useful also in design systems, as it allows us to manipulate the genotype quite separately from the phenotype.

The ability to manipulate the genotype provides us with the locus of learning. The question, however, remains how can tools learn? Simply running an evolutionary system to produce good designs does not change that system. There needs to be a process which has the capacity to change the genetic representation in such a manner that the next time the system is used on a similar problem it will perform better. The area in natural evolutionary systems which manipulates the genetic material is called *genetic engineering* [7]. We will develop an analog of the practice of genetic engineering in the genetics of natural organisms. Here the genetic engineer first classifies the population of phenotypes into two basic groups. The first group contains phenotypes (or in our case designs) which have a high level of performance in the fitness. The second group of phenotypes does not exhibit this high level of performance. The genetic engineer then

looks through the genotypes of these two groups and tries to find sets of genes in the genotypes of the high fitness group which all such genotypes exhibit and which do not occur in the low fitness group's genotypes, Figure 1.

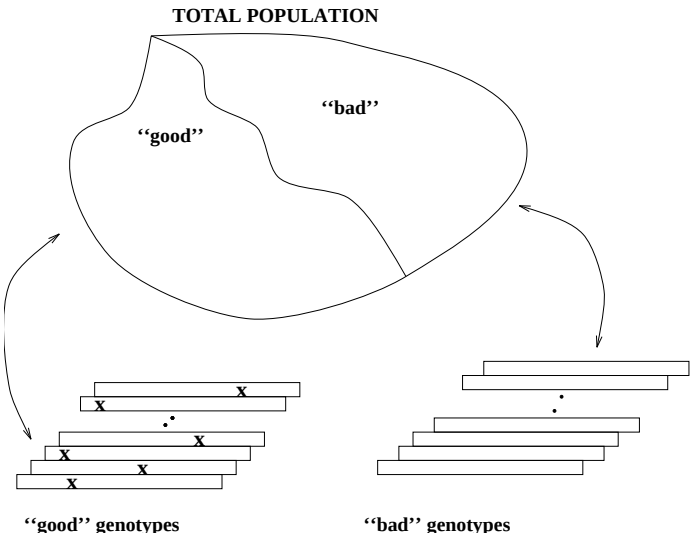

Figure 1. The genotypes of the "good" members of population all exhibit gene combinations, X, which are not exhibited by the genotypes of the "bad" members. These gene combinations are the ones of interest in genetic engineering (after [8]).

We can see this in a different way exemplified in Figure 2 which contains 10 designs produced using an evolutionary system. Under each design is the sequence of genes used to produce it and next to each design is its performance in the fitness.

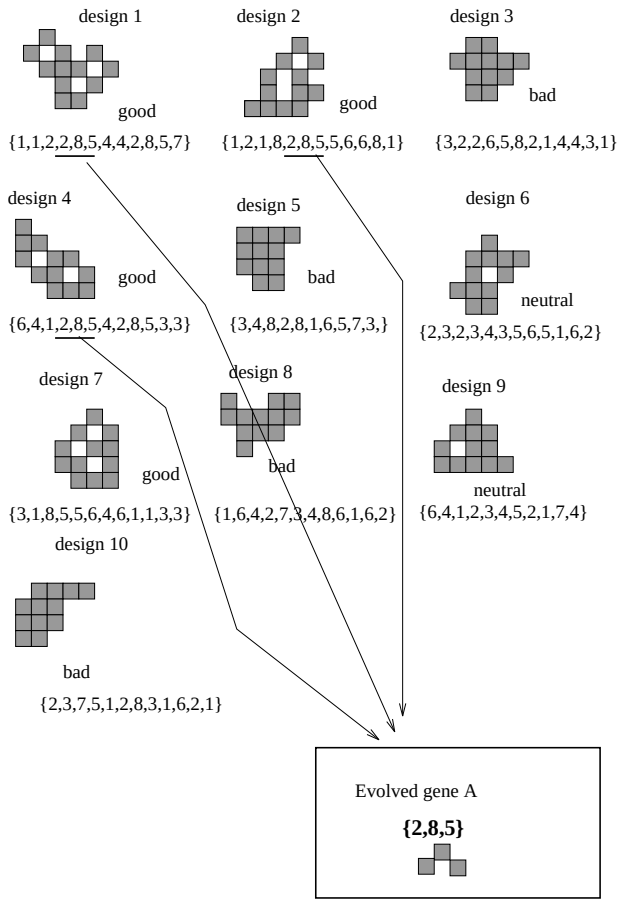

Figure 2. The identification of the gene structure {2,8,5} as the contributor to the high level of fitness allows a new gene, A, to evolve (after [9]).

If such gene groups can be found it is taken that they contribute to the high level of performance in the fitness since they only exist in the genetic material of those high performing phenotypes and not in the low performing ones. The genetic structures represented by these gene groups are learned and called *evolved genes*. These evolved genes are now added to the gene pool from which the genotype is constructed. The next time the evolutionary system is used to solve a similar problem it already has learned some of the characteristics which can be used to produce good solutions and as a consequence it will produce those solutions more effectively.

2.3 An example

Consider the trivial situation where we are trying to produce some simple designs in the form of plans generated by drawing vectors. We have only four basic drawing vectors available to us: up, down left and right. We could try a variety of combinations of these vectors to produce interesting plans, as an evolutionary system might. However, as we proceed we use the concepts of genetic engineering to locate and evolve genes which make the production of interesting plans easier.

Consider Figures 3(a) and 3(b). Figure 3(a) shows a series of concentric circles. Each circle shows the plans which can be constructed with the number of genes indicated. The larger the number of genes required, the more costly is the process and the less likely is it that interesting solutions will be adequately found. The innermost circle contains the original genes as the four drawing vectors. Note that circle 14 shows a closed plan composed of four squares – it takes 14 genes to produce this plan, which is one of a very large number of 14 gene plans, not all of which are closed. If the tool learns that square plans are interesting and uses

genetic engineering to locate the gene structure and evolves a gene with that structure we get the situation described in Figure 3(b) which occurs when the tool is reused to produce similar plans.

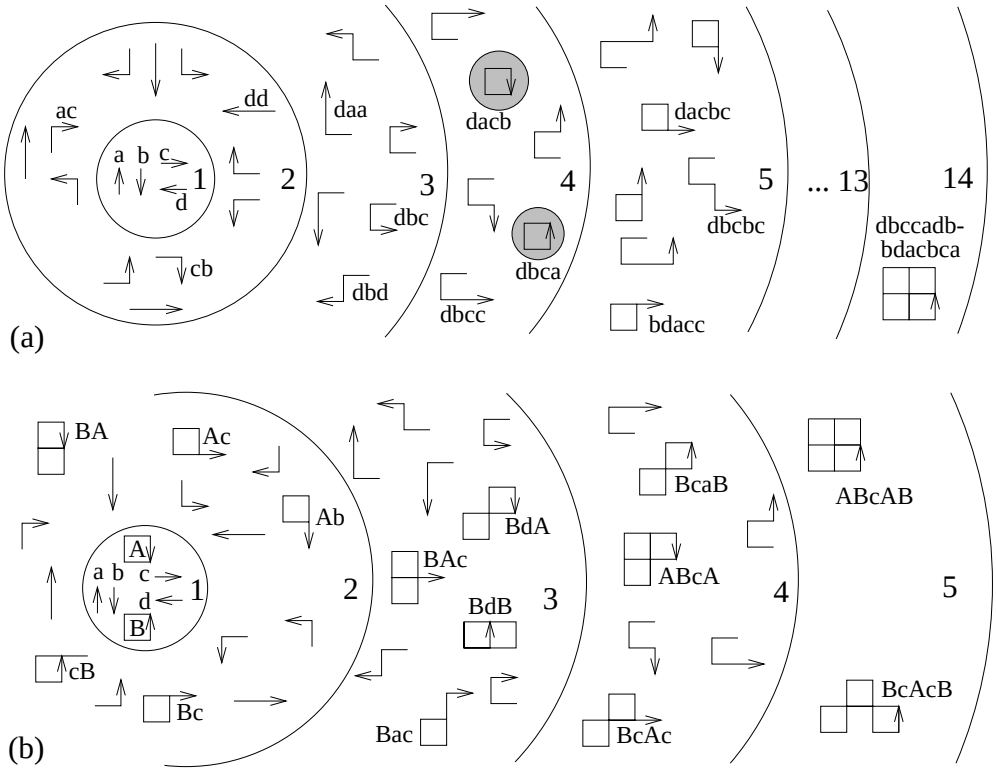

Figure 3. Example of an evolving representation: (a) original representation and (b) representation with evolved genes. Some of the corresponding genotypes are given, capital letters denote evolved genes. The transformation from phenotype to genotype is not always unique, e.g. the genotypes 'Abc' and 'Bac' produce the same phenotype. Arc segments indicate that only part of the space is shown [10].

As can be seen in Figure 3(b) the four-square plan which previously took 14 genes to describe with its consequential computational load now only takes 5 genes to describe this four-square plan. This is accompanied by a reduction in computation.

3 A tool that learns

These particular concepts using genetic engineering techniques have been implemented and are being applied in a number of domains. Let us look at its application to a space layout planning problem in architecture [11]. We take a spatial layout planning problem from the literature [12]. It consists of the problem of allocating a set of offices into a set of spaces predefined by the building envelope, Figure 4. The details of the spaces and their interactions can be found in the references listed. During the process of producing a solution to this problem the tools learns that a particular grouping of genes {0,1,2,3,4,8}, in no particular order, is a highly beneficial structure in that as long as these genes are next to each other a very high level of the fitness which measures layout performance is obtained, and evolves a gene to represent that.

Now the tool is used with a similar problem where an alternate building envelope is used as shown in Figure 5. The tool has already learned that a particular grouping of genes was useful in the former related problem and has that available as an evolved gene. As a consequence the system finds solutions in much less time as indicated in Figure 6, where the bold line shows that the tool's performance without any learning and the dotted line shows its performance after reusing the knowledge that it learnt in the previous

problem. The difference in performance can be measured approximately by the number of generations needed to locate similar solutions. The standard genetic algorithm took, on average, 48 generations while with learning and reuse of the learned knowledge the modified genetic algorithm took, on average, only 9 generations. This represents a saving of 80 percent.

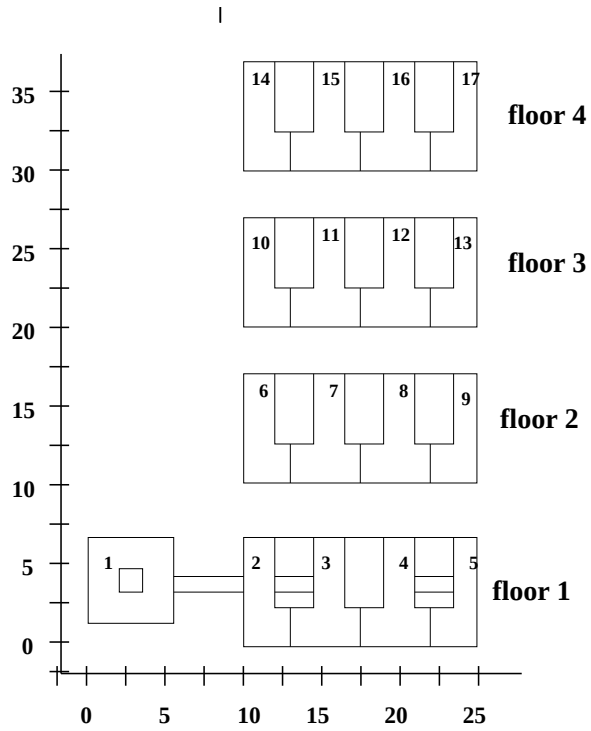

Figure 4: The building into which the spatial activities must be allocated, the numbers are zone labels (after [12]).

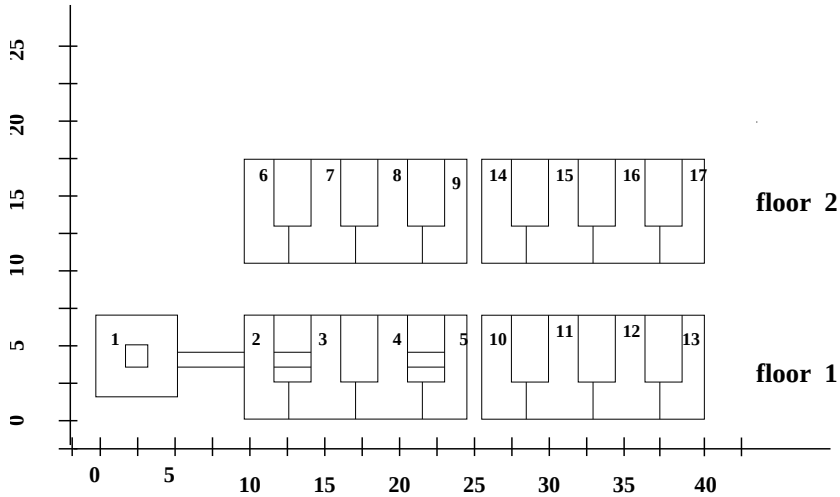

Figure 5: Alternate building envelope for the layout problem [11].

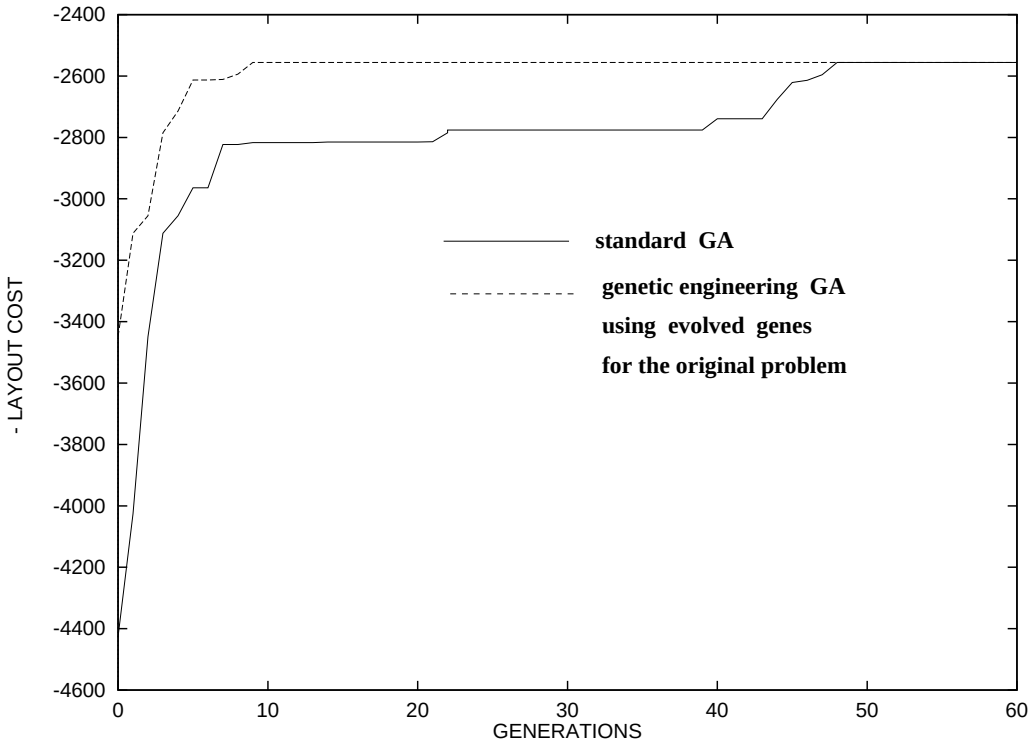

Figure 6: The effect of reusing knowledge learned for previous problem on the efficiency of the tool (after [11]).

4 Some basic precepts for the development of tools that learn

4.1 What can be learned?

The primary relationships which are most interesting to learn are those which connect some aspect of the performances of designs with the representation of the structure of designs which produce high levels of performance.

At some fundamental level a representation consists of features, ie items for which there is a representation. If there is no representation available for that feature it does not exist directly in the representation. Thus, a representation may be conceived of as a set of feature detectors which can be used to locate the feature which can then be represented through features. Figure 7(a) shows the image of a square. Figure 7(b) shows some ways in which this square could be represented. Thus, a square could be a single line, two reflected L-shapes, a U-shape with a line across its opening, two reflected U-shapes, and so on. Certainly then, one aspect that can be learned is which features best go to make up a representation. In some sense this is what was happening in the example above concerning the genetic engineering of the representation to produce evolved genes.

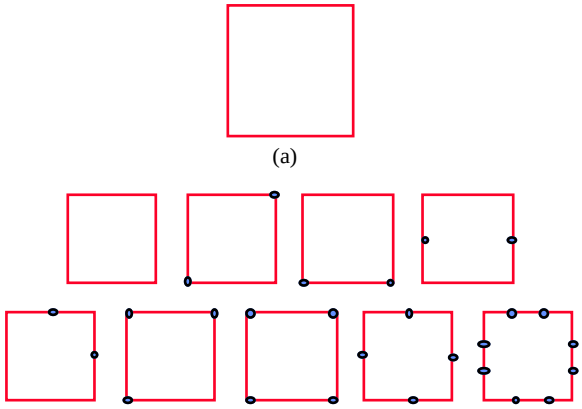

Figure 7: (a) Image of a square; (b) different representations of that square based on different feature detectors all of which produce the same image but potentially different 'readings' (the circular nodes indicate edges of the feature detectors and do not appear in the image).

4.2 Emergence

Emergence is the process of making explicit (ie represented) features which were previously only implicit (ie not represented). This is a form of learning also in that knowledge is acquired and the representation may be restructured. The most common forms of emergence relate to the visual/graphical domain but emergence is not restricted to such domains. Consider the two floor plans in Figure 8. We would like to emerge the features which could be used to represent them at some level.

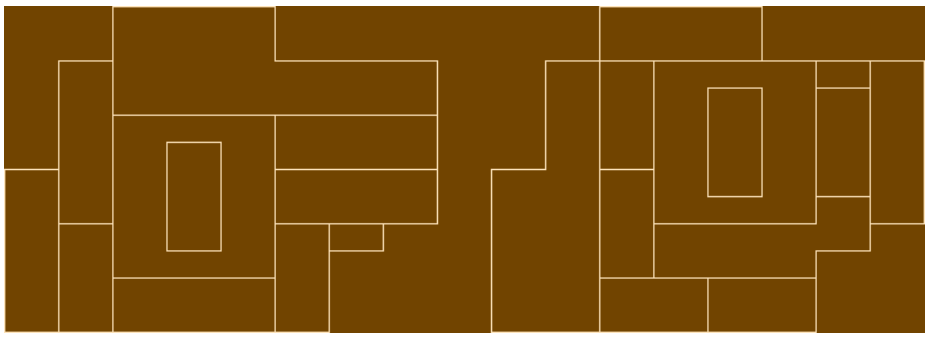

Figure 8: Two floor plans with related features (after [8]).

Figure 9 shows some of these emerged features in the form of segments of room shape boundaries. These emerged features restructure the representation and are now available to be reused in another related design problem.

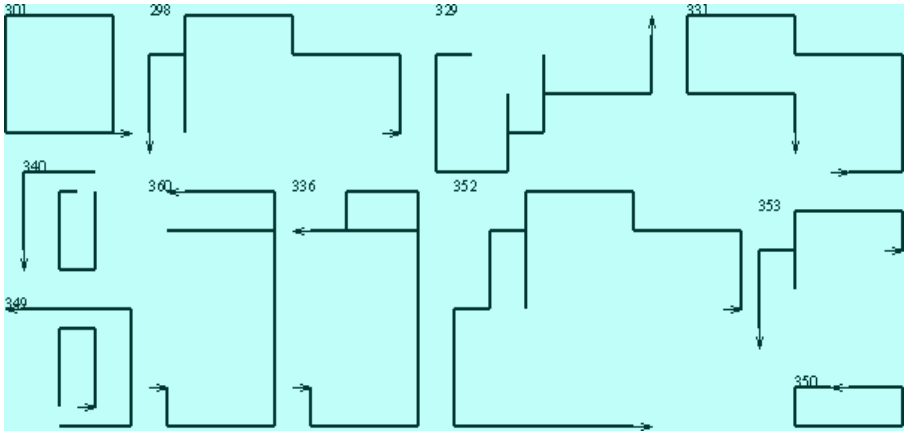

Figure 9: A set of emergent features which have been learned by the system through a change in representation (after [8]).

These emergent features can now be used to generate new designs which exhibit those features. Figure 10 shows five floor plan layouts produced using the emerged features but with different client requirement than those set for the original floor plans. Here the number of rooms and their size ranges are different. An examination of the floor layouts in Figure 10 shows the similarity of the characteristics of these designs with those of the previous designs.

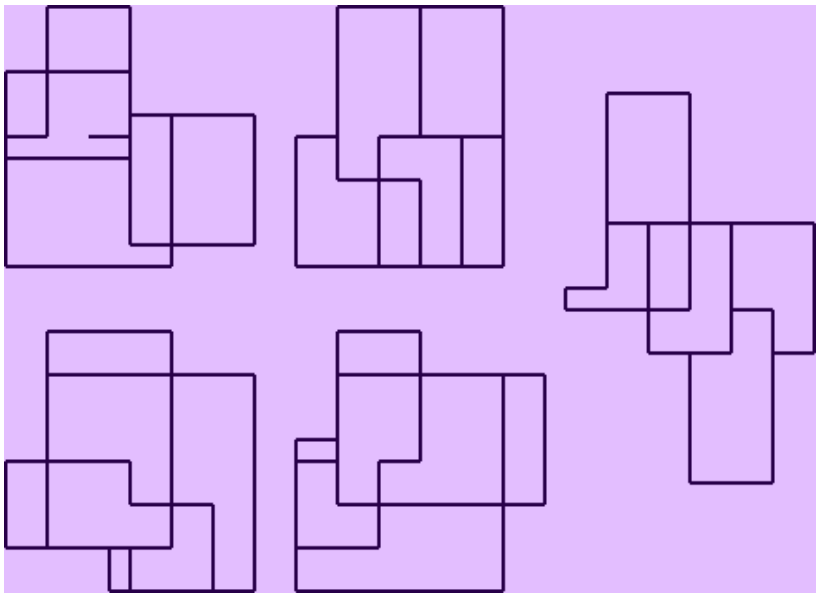

Figure 10:. Five floor plans generated using emergent features learned from the designs in Figure 8 (after [8].

4.3 Reusing the tool not the knowledge

A tool that learns about a particular problem may not find that what it has learnt is useful for another problem because there is insufficient similarity in either the problem structure or the problem's numerical description. It is important then that the efficacy of the tool in its other uses not be impaired by the knowledge it has acquired. The implication of this is that the basic tool be unchanged by the learning process. This implies that either the tool has the capacity to recognise appropriate problems to which to

apply some or all of its learned knowledge or the learned knowledge does not affect the operation of the tool.

5 Roles for tools that learn

Tools that learn could be treated in a number of ways. They could be personalised so that each designer has his own representations which encodes what the tool have learnt while being used by that designer. The effect of this is to personalise the tool in the same way that a handwriting recognition tool is personalised for each user.

An alternate or possibly conjoint view is that tools could be specialised to individual tasks. Thus, a genetic algorithm which is a general search process could have its representation specialised for layout tasks and would require a different representation for other tasks since the layout representation would contain domain knowledge.

Acknowledgments

The ideas in this paper are based on projects carried out at the Key Centre of Design Computing, University of Sydney in conjunction with Vladimir Kazakov, Thorsten Schnier and Phil Tomlinson. These projects are all funded by the Australian Research Council. Computational support is provided by the Key Centre of Design Computing.

References

1. M. L. Maher, M. Balachandran and D. M. Zhang, "Case-Based Reasoning in Design", Lawrence Erlbaum, Hillsdale, New Jersey, 1995.

2. D. Gunaratnam and J. S. Gero, "Neural network learning in structural engineering applications", in "Computing in Civil and Building Engineering", Cohen, L. F. (Editor), ASCE, New York, 1448-1455, 1993.

3. J. Carbonell, "Paradigms for machine learning", in "Machine Learning Paradigms and Methods", Carbonell, J. (Editor), MIT/Elsevier, Cambridge, MA, 1-10, 1990.

4. M. L. Maher and H. Li, "Automatically learning preliminary design knowledge from design examples", Microcomputers in Civil Engineering,Vol. 7, 73-80, 1992.

5. T. Arciszewski, Z. Mustafa and W. Ziarko, "A methodology of design knowledge acquisition for use in learning expert systems", Man-Machine Studies, Vol. 27, 23-32, 1987.

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

[7] W. H. Sofer, "Introduction to Genetic Engineering", Butterworth-Heinemann, Stoneham, 1991.

[8] J. S. Gero, V. Kazakov and T. Schnier, "The genetic engineering of evolutionary systems in design", AID'96 Workshop on Evolutionary Systems, (to appear), 1996.

[9] J. S. Gero and V. Kazakov, "Evolving building blocks for design using genetic engineering: a formal approach", in "Advances in Formal Design Methods for CAD", Gero, J. S. (Editor), Chapman and Hall, London, 31-50, 1996a.

[10] T. Schnier and J. S. Gero, "Learning genetic representations as alternative to hand-coded shape grammars", *in "Artificial Intelligence in Design'96*", Gero, J. S. and Sudweeks, F. (Editors), Kluwer, Dordrecht, 39-57, 1996.

[11] J. S. Gero and V. Kazakov, "Evolving design genes in space layout problems", Working Paper, Key Centre of Design Computing, University of Sydney, Sydney, 1996b.

[12] R. S. Liggett, "Optimal spatial arrangement as a quadratic assignment problem", in "Design Optimization", Gero, J. S. (Editor), Academic Press, New York, 1-40, 1995.