Design Knowledge Acquisition and Re-Use Using Genetic Engineering-Based Genetic Algorithms

John S. Gero and Vladimir Kazakov

Key Centre of Design Computing and Cognition, Department of Architectural and Design Science
The University of Sydney, NSW 2006 Australia. e-mail:{john,kaz}@arch.usyd.edu.au

Abstract: This chapter describes an application of genetic engineering-based genetic algorithms as a tool for knowledge acquisition and re-use. This version of genetic algorithms is based on a model of neo-Darwinian evolution enhanced by an analysis of genetic changes, which occur during evolution, and by application of various operations that genetically engineer new organisms using the results of this analysis. The genetic analysis is carried out using various machine learning methods. This analysis yields domain-specific knowledge in a form of two hierarchies of beneficial and detrimental genetic features. These features can then be re-used when similar problems are solved using genetic algorithms. Layout planning problem is used to demonstrate the process and the results obtainable.

1. Introduction

The use of machine learning techniques in design processes has been hampered by a number of problems. The first problem arises because the problem solving process and the learning process are usually considered as two different activities [1], which are conducted separately, analyzed separately, and which employ different processes and tools.

The second problem is that learning is considered a more general task than problem solving and thus a more difficult one. Therefore computational and other costs associated with the conduct of machine learning is normally significantly higher than the corresponding costs for problem solving only. This limits the scope of applications of learning methods in the industry. Also the more efficient types of learning require considerable domain-specific knowledge and thus additional cost/time for their development. The effect of this is that it has to be done separately for each domain.

The third problem relates to how much knowledge a problem-solving tool developer is willing to incorporate into the tool. Clearly, the higher is the knowledge-content of the tool the more effort its development will require and less general the domain where this tool can be used. The ease of development and wide applicability of such knowledge-lean problem-solving tools such as genetic

algorithms [2], [3], simulated annealing [4], and neural networks have made them extremely popular.

The fourth problem is that the representations used by the learning tools/methods and the problem-solving tools/methods are usually incompatible. Therefore, an additional transformation of the knowledge that has been acquired during learning, has to be performed before it can be used in problem solving. Attempts to use this knowledge for further advanced processing (combining knowledge from different domains/problems, generalization, etc.) also leads to need for the development of some homogeneous knowledge representation.

In this chapter we present an approach that attempts addresses all these problems. Firstly, it is a problem-solving tool and also a learning tool. It includes a problem-solving component and a learning component that operate simultaneously and provide feedback to each other. Secondly, instead of reducing the efficiency of problem-solving process, the learning here enhances it. This makes the overall process more efficient than the problem-solving component of this algorithm alone. The learning approach that is employed by the algorithm is general and requires very little or no domain knowledge. Therefore the development of this method for new domains requires very little effort. Thirdly, the algorithm results in a knowledge-lean tool that can be initiated without any knowledge, but it acquires knowledge when it is being used and the longer it is used the richer in knowledge it becomes. Fourthly, because this algorithm acquires and utilizes the knowledge using a single homogeneous representation, no knowledge transformation is needed when it is re-used, re-distributed, generalized, etc

The problem-solving module of the proposed algorithm uses the standard genetic algorithm machinery (selection/crossover/mutation) [2] followed by an additional stage of genetic engineering. In this additional stage the population's genetic material is subjected to direct manipulation that simulates various genetic engineering techniques. The learning module can employ any one of the standard machine learning algorithms. At the end of each evolutionary cycle two training sets to be used by the learning module are formed from the best and the worst performing solutions within the current population. The learning module identifies which attribute (genetic) features are beneficial and which are detrimental for the current population. The genetic features that have been identified before from the previous populations are re-tested against the current population. The ones that fail are deleted from the list of beneficial/detrimental features. Next the standard GA processes produce an intermediate population from the current population. This population is subjected to genetic engineering processing, which promotes the presence of beneficial features and discourages the presence of detrimental features. This produces a new population and concludes the current evolution cycle and the next evolutionary cycle is initiated.

2. Genetic engineering genetic algorithms

Genetic algorithms (GAs) [2], [3] are search algorithms that simulate Darwinian evolutionary theory. GAs attempt to solve problem (e.g., finding the maximum of an objective function) by randomly generating a population of potential solutions to the problem and then manipulating those solutions using genetic operations. The solutions are typically represented as finite sequences (genotypes) drawn from a finite alphabet of characters. Through selection, crossover and mutation operations, better solutions are generated out of current population of potential solutions. This process continues until an acceptable solution is found, Figure 1. GAs have many advantages over other problem-solving methods in complex domains. They are very knowledge-lean tools, which operate with great success even without virtually any domain knowledge.

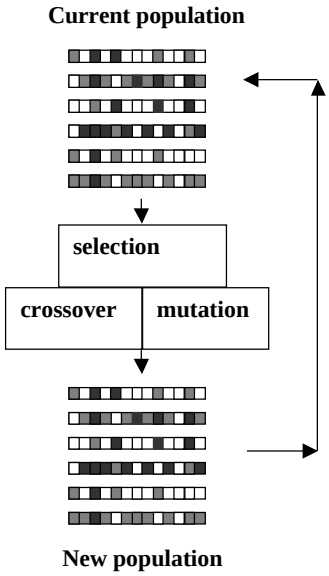


Figure 1. The structure of genetic algorithms evolutionary cycle.

Genetic-engineering based GA is an enhanced version of GAs that simulates Darwinian theory of the current stage of the natural evolutionary process into which genetic engineering technology has been introduced. This technology is based on the assumption that the genetic changes that occur during an evolution can be studied and that the genetic features that lead to wanted and unwanted consequences can be identified and then used to genetically engineer an improved organism. Thus, the model assumes that this technology includes two components:

(a) methods for genetic analysis, which yield the knowledge about the beneficial and the detrimental genetic features; and

(b) methods for genetically engineering an improved population from the new population generated by normal selection/crossover/mutation process by using the knowledge derived from genetic analysis.

The genetic-engineering GA uses a stronger assumption than (a), that standard machine learning methods can identify the dynamic changes in the genotypic structure of the population that take place during the evolution. The type of machine learning depends on the type of genetic encoding employed in the problem. For the order-based genetic encoding sequence-based methods [5] give the best results and for attribute-value based genetic encoding decision trees [6], [7] or neural networks [8] should be tried first. The two training sets for machine learning are formed by singling out the subpopulations of the best and worst performing solutions from the current population. The structure of the learning module is shown in Figure 2. Note that since new features are built by the system as derivatives of the already identified features (which are the elementary genes/attributes when the system is initiated), the approach readily leads to a hierarchy of genetic features. For example, if genetic features that occur are position-independent genetic "words" (as it is the case in the system in Figure 2) then the features that evolve later would by genetic "words" that include previously found "words" as additional letters of genetic alphabet.

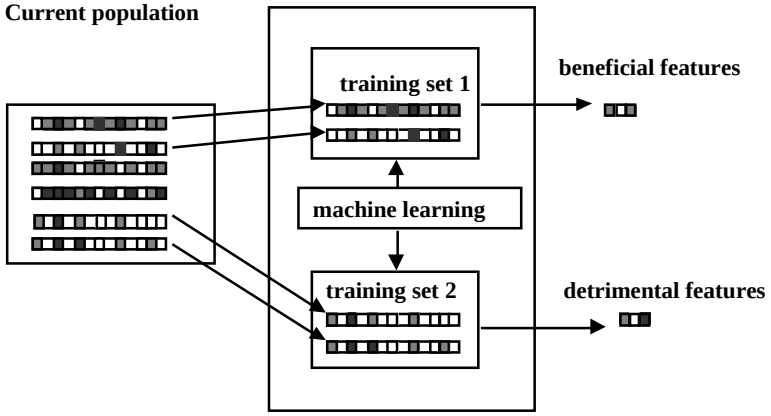


Figure 2. The structure of the learning module of genetic engineering GA. The genotypes in the current population are vertically ordered by their performances, so the higher the genotype is in the box, the higher is its performance.

The genetic engineering of the new population applies the following four operations to each genotype:

(a) screening for the presence of the detrimental genetic features;
(b) screening for the presence of incomplete beneficial genetic features;
(c) replacing the minimal number of genes to eliminate the detrimental features found during screening (a);
(d) replacing the minimal number of genes to complete the incomplete beneficial genetic features found during (b).

If some beneficial/detrimental genetic features are known when the genetic engineering GA is initiated, then even the initial population (which normally is generated randomly) is subjected to the genetic engineering operations (a)-(d). These are the principles, the implementation details of genetic engineering GA can be found in [9].

3. Learning in genetic engineering genetic algorithms

Let us discuss the major characteristics of learning process is genetic engineering GA. Firstly, note that if one employs any problem-solving method that is population-based (like GA), then the solution process will generate a large amount of information (the optimal solutions, the search trajectories, etc.) that can be used to carry out some form of learning. The possible recipes for the formation of the training sets for genetic learning here can vary between two extremes of:

(a) using only the optimal solutions [10]; here only the final population is used for learning and thus the cost of learning is minimal and the possibilities for knowledge acquisition are also minimal; and
(b) using the complete set of search trajectories.

Genetic engineering GA corresponds to some middle choice by using only the current population for learning. Nevertheless, it can be easily generalized to include a more comprehensive analysis of larger parts of the evolution history that lead to the current population. Obviously some tradeoff between the comprehensiveness of genetic analysis and its cost should be achieved to balance the advantages of more comprehensive genetic learning against its computational and developmental cost.

Secondly, note that the space of genetic sequences (genotypes) and its derivatives form the only domain that is used for learning in genetic engineering GA. Thus, whatever knowledge can be learned by the algorithm, it is always expressed in terms of some characteristics of these genetic sequences, that is, in terms of genetic representation. This is important since the learned knowledge is in the same representation as the source from which it was learned and hence it can be used directly. We shall call these learned characteristics genetic features. Many features types can emerge in a genetic sequence and a genetic engineering GA can be tailored to handle all of them [9] by including the machine learning

methods that are capable of learning these regularities and by creating the specific versions of genetic engineering operators to target these genetic regularities. Nevertheless, the two most frequent and important types of features are:

(a) the set of fixed substrings in fixed positions (called building blocks in GA theory [2]); and

(b) the fixed substring that is position-independent (like words in natural languages).

The specific versions of genetic engineering GA that efficiently handle these feature types can be found in [9]. They should be used when it is known a priori that it is likely that the problem under consideration has these types of genetic regularities. When this is not known then a general learning method that is capable of identifying a wide range of features and the corresponding non-specialized genetic engineering operators should be used.

Thus, the learning process in genetic engineering GA leads to a step-wise recursive enlargement of its feature space, layer by layer. This recursion builds a hierarchical structure of this feature space, where each new layer is a derivative of all the previous layers of features. For the problem where the characteristic genetic features are position-independent genetic substrings (genetic "words", Figure 2), this hierarchy can be represented as a tree, Figure 3. Here the new features are derivatives of the already evolved features because they use them as building components (sub-substrings).

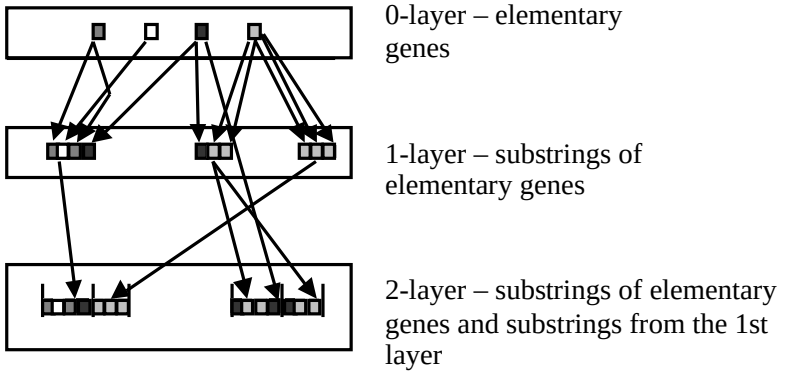


Figure 3. The hierarchical knowledge structure (feature space) is built by genetic engineering GA for the problem where evolved genes are genetic "words". The arrows denote the component from the lower lever that is used to assemble new feature on the next level.

The problem's knowledge in the genetic engineering GA is its feature space, which is derived from its genetic representation. Therefore, if two problems have identical or even partially overlapping genetic encodings then the knowledge that is acquired when one problem is solved can be translated into complete or partial initial knowledge (feature space) for the second problem. If a number of instances of the problems from the same class are solved then the generalization of their knowledge into a class of knowledge (class feature space) is the straightforward procedure of finding the overlapping part of their feature space. This concept is important as it allows the problem-solving tool to improve its performance as it acquires more problem class-specific knowledge.

4. Layout planning problem

The space layout planning problem is fundamentally important in many domains from architectural design to VLSI floor-planning, process layouts and facilities layout problems. It can be formalized as a particular case of a combinatorial optimization problem – the quadratic assignment problem. As such it is NP-complete and presents all the difficulties associated with this class of problems.

Thus, we formulated the layout-planning problem as a problem where such one-to-one mapping ρ

$$\rho: \{M \rightarrow N\}, j = \rho(I), i \quad M, j \quad N$$

of the discrete set M with m elements (set of activities, for example office facilities) onto another discrete set N of n elements (set of locations, for example floors of the building where these facilities should be placed), $m \leq n$ is sought that the overall cost of the layout I is minimal, i.e.

$$I = \sum_i f_{i\rho(i)} + \sum_i \sum_j q_{ij} c_{\rho(i)\rho(j)} \rightarrow \min$$

where f_{ij} is the given cost of assigning element $i \quad M$ to the element $j \quad N$, q_{ij} is the measure of interaction of elements $i,j \quad M$, and c_{ij} is the measure of distance between elements $i,j \quad N$.

Usually the space layout-planning problem contains some additional constraints, which prohibit some placements and/or impose some extra requirements on feasible placements.

5. Example

As a test example we use Liggett's problem of the placement of a set of office departments into a four-storey building [11]. The areas of the 19 activities to be placed (office departments, numbered 0,..,18) are defined in Table 1 in terms of elementary square modules. There is one further activity (number 19) whose location is fixed. The objective of the problem does not have a non-interactive cost term, i.e. $f_{ij} = 0$. The interaction matrix q_{ij}, $i, j = 0, \ldots, 19$ is given in Table 2. The set of feasible placements is divided into 18 zones numbered from 0 to 17, Figure 4 (a). The areas of these zones are defined in Table 3. The activity number 19 is an access area which has a fixed location - zones numbers 16 and 17, Figure 5.

Since the layout cost does not depend on the areas of the zones numbered 16 and 17 or on the area of the activity number 19 they are not shown in Table 3. The matrix showing travel distances between all zones is defined in Table 4. We use the same genetic representation as was used in [12] and [13] – the genotype of the problem is a sequence that determines the order in which zones are filled with activity modules. Each zone is filled line by line starting from its highest one and from the leftmost position within it. For example, the genetic sequence

x = {13, 14, 0, 2, 6, 9, 16, 11, 3, 4, 18, 15, 17, 1, 12, 10, 8, 7, 5}

generates a layout plan in the following manner. All 18 elementary modules of activity 13, i.e. department 0800 are placed, followed by the 31 modules of activity 14, i.e. department 0900, followed by the 2 modules of activity 0, i.e. department 0210, etc.

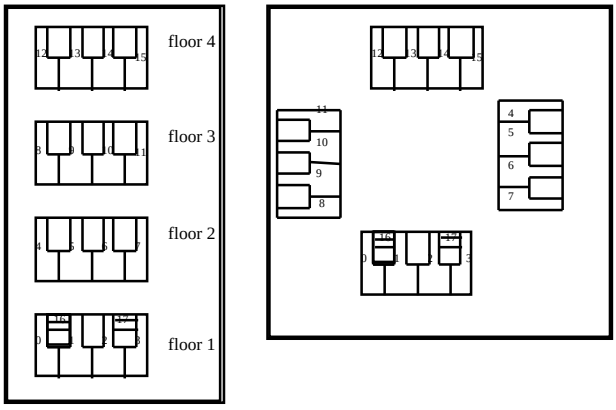
Table 1: The definition of activities [11].

Activity	0	1	2	3	4	5	6	7	8
Dept	0210	0211	0220	0230	0240	6815	0300	0400	0500
Num. Of modules	2	2	8	15	15	13	15	7	6

Act.	9	10	11	12	13	14	15	16	17	18
Dept	0600	0700	6300	6881	0800	0900	1000	Extra	extra	Extra
Mod.	12	53	10	16	18	31	61	1	1	1

Table 2: Activity interactions matrix [11].

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



(a) (b)

Figure 4. Zone definitions for Liggett's example [11] (a) and its modification (b).

Table 3: Zone definitions, measured in number of modules [11].

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

The genetic engineering GA will be first used to solve this layout planning problem. During the process of solving this problem knowledge about the problem will be acquired. The acquired knowledge will be then re-used during the solution of a similar problem where the same set of activities is to be placed into a modified zone location, Figure 4(b). This is a common situation where alternate possible receptor locations are tried. In building layouts this can be the result of

alternate existing buildings. The same applies to the layout of processes and department stores. In this case the change is from a four-storey building to a courtyard-type single storey building. This new zone structure yields the modified distance matrix shown in Table 5.

Table 4: Distance matrix [11].

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0		4	9	21	27	9	10	23	28	10	11	24	29	11	12	25	30	6	24
1			5	17	23	5	6	19	24	6	7	20	25	7	8	21	26	2	20
2				13	18	6	5	18	23	7	6	19	24	8	7	20	25	2	15
3					5	23	18	5	6	24	19	6	7	25	20	7	8	15	2
4						24	19	6	5	25	20	7	6	26	21	8	7	20	2
5							5	18	23	5	6	19	24	6	7	20	25	3	21
6								13	18	6	5	18	23	7	6	19	24	3	16
7									9	23	18	5	6	24	19	6	7	16	3
8										24	19	6	5	25	20	7	6	21	3
9											5	18	23	5	6	19	24	4	22
10												13	18	6	5	18	23	4	17
11													5	23	18	5	6	22	4
12														24	19	6	5	17	4
13															5	18	23	5	23
14																13	18	5	18
15																	5	18	5
16																		23	5
17																			18
18																			

Table 5: The modified distance matrix for new zone location, Figure 4(b).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0		4	9	21	27	9	10	23	28	10	11	24	29	11	12	25	30	6	24
1			5	17	22	29	34	46	51	58	53	41	36	29	24	12	7	2	20
24				12	17	24	29	41	46	51	56	49	44	37	30	25	13	2	15
3					5	12	17	29	34	41	46	58	53	46	41	29	24	15	2
4						7	12	24	29	36	41	53	58	51	46	34	29	23	10
5							5	17	22	29	34	46	51	58	53	41	36	28	15
6								12	17	24	29	41	46	53	58	46	41	40	27
7									5	17	22	34	39	46	51	58	48	45	32
8										7	12	24	29	36	39	51	56	52	39
9											5	17	22	29	34	46	51	57	44
10												12	17	24	29	41	46	59	56
11													5	12	17	29	34	45	65
12														7	12	24	29	35	55
13															5	17	23	30	51
14																12	17	18	38
15																	5	15	33
16																		6	25
17																			18
18																			

6. Results of simulations

In the initial problem the genetic engineering GA was not able to find any detrimental genetic regularities but found four instances of beneficial genetic regularities. They were of a non-standard type of genetic regularity that is characteristic not only for this particular example but also for many other layout planning problems. This feature is a gene cluster – a compact group of genes. The actual order of genes within each group is less significant for the layout

performance than the presence of such groups in compact forms in genotypes. The genetic engineering GA was able to find first two clusters {0, 1, 2, 8, 3, 4} and {12, 14} after the 5-th generation. Then it found two more clusters {6, 13}and {15, 12, 14} after the 10-th generation. Note that the last cluster was derived from the previously identified cluster {12, 14}. This is an example of how the genetic engineering GA builds a multilayer hierarchy of evolved genetic structures, where each new layer is derived from the components below it in the hierarchy (from the features, which have been already identified).

The comparison of the standard GA evolution process and the genetic engineering GA evolution processes shown in Figure 5, illustrates the significant computational saving genetic engineering brings in terms of the number of generations that are needed for convergence. The overall computational saving here was 60%. In the general case, this saving typically varies between 10% and 70%, depending on the computational cost of the evaluation of the genotype relative to the computational cost of the machine learning employed.

If these clusters are given to genetic engineering GA when it is initiated, then the number of generations needed for convergence drops from 21 to 15.

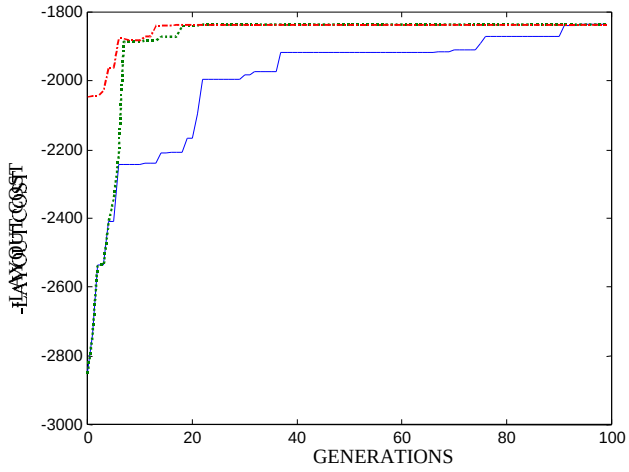


Figure 5. The best layout cost vs generation number for the standard GA (solid line), genetic engineering GA without prior knowledge (dotted line) and genetic engineering GA with prior knowledge (dashed line). The results are averaged over 10 runs with different initial random seeds.

The simulation was then carried out for the modified example, Figure 4(b). The results are shown in Figure 6 for genetic engineering GA's runs under the following conditions:

(a) without any prior knowledge;
(b) with partially incorrect prior knowledge (using the four clusters that have been acquired by solving the non-modified example); and
(c) with correct prior knowledge for the modified example.

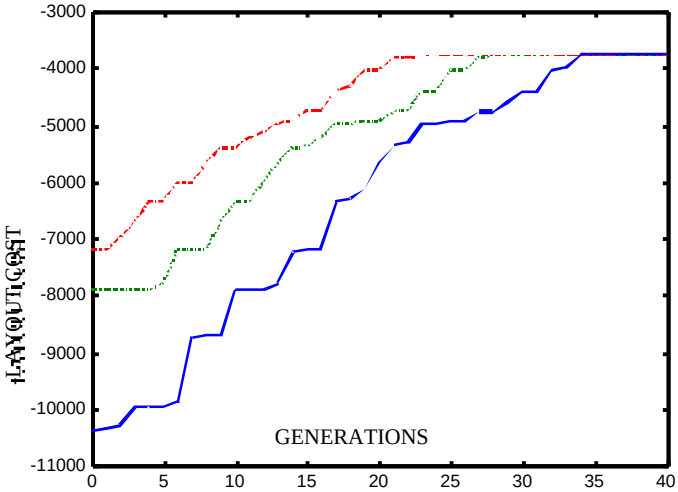


Figure 6. The best layout cost vs. generation number in modified example genetic engineering GA without prier knowledge (solid line), with non-perfect prior knowledge (dotted line) and with perfect prior knowledge (dashed line). The results are averaged over 10 runs with different initial random seeds, which converge to the best solution found.

Only one of the four beneficial clusters from the initial problem survived the problem's modifications and was retained by the genetic engineering GA in simulation (b). This is the first cluster {0, 1, 2, 8, 3, 4} of a group of very small but strongly attracted to each other activities. After modification it even grew into a larger cluster {0, 1, 2, 8, 3, 4, 5, 17}. The genetic engineering GA also found one new cluster {6, 7, 13, 15}.

To demonstrate the benefits of these clusters for the layout cost we calculated the average cost of layouts with and without these clusters for the modified test problem, Table 6. The average layout costs were calculated using the Monte-Carlo method with averaging over 500 trial points. For the first cluster, which is beneficial in both problems, we also present the decomposition of the average layout cost into two components: the cost of the inter-cluster activity interactions I_I and the $I_c = I - I_i$.

Table 6: The average layout cost in the modified test problem with and without evolved beneficial genetic clusters.

Gene clusters	No clusters	{0, 1, 2, 8, 3, 4, 5, 17}	{1, 5, 6, 7, 13}
Average layout cost	10880	5541	8924

This cost decomposition is presented in Table 7 for a sample of gene sequences within the cluster for both original and modified problems. For the original problem the reduction in I_i. was 55% (with standard deviation of 2%) and the reduction in I_c was 2%. For the modified problem the reduction in I_i. was 66% (with standard deviation of 5%) and I_c was actually increased by 2.5%. This cost decomposition provides some insight as to why this cluster is beneficial - the placement of its genetic components in a tight group leads to the large reduction of I_i. but very small variation of I_c. Thus, the genetic engineering GA in layout planning problem "discovers" the gravitational-type model that is hidden in the problem statement.

Table 7: The average layout cost in the original and modified test problem with and without cluster of genes {0, 1, 2, 8, 3, 4}

Gene cluster	Original problem		Modified problem	
	I_i	I_c	I_i	I_c
Scattered	1539	2539	4025	6854
{8, 0, 1, 2, 3, 4}	607	2540	4025	6832
{0, 1, 2, 8, 3, 4}	637	2536	972	6827
{0, 1, 2, 3, 8, 4}	671	2536	985	6847
{4, 1, 2, 8, 0, 3}	729	2510	1015	6790
{8, 2, 1, 0, 3, 4}	628	2535	1013	6801
{4, 2, 1, 8, 0, 3}	687	2509	984	6718

7. Discussion

In this chapter we have presented an approach to automated knowledge acquisition and re-use in design through the deployment of a genetic engineering-based genetic algorithm that operates in conjunction with the problem-solving process. This approach obviates a number of problems that occur when automated knowledge acquisition systems are used separately from the problem-solving process. Genetic engineering-based genetic algorithms make use of the notions of genetic engineering: that structural features of the genotype influence the fitness

or behavior of the resulting designs. If these features can be isolated they can be manipulated to the benefit of the resulting design process.

The genetic features in the genotypic representation of designs are problem specific knowledge that has been acquired using a knowledge lean process. The knowledge acquired in this manner is in the same representational form as the genotypic design representation and can therefore be used without the need for any additional interpretive knowledge. These genetic features can also be subjected to genetic engineering style operations. Operations such as gene surgery and gene therapy have the potential to improve the performance of the resulting system.

Since the genetic features are in the form of genes, they can be replaced by a single “evolved“ gene and thus extend the range of symbols used in the genetic representation. This extension allows for any genetic algorithm that uses it to search for design solutions in a more focussed way since it it using knowledge that was previously not available in the formulation of the problem. The genetic engineering approach to knowledge acquisition provides opportunities to acquire different kinds of knowledge. It can be used to acquire knowledge about solutions as was demonstrated in the example above, or it can be used to acquire knowledge about design processes [14]. In acquiring knowledge about solutions it is possible to caquire design knowledge from two different design problems that are linked only by a common representation and then to combine this knowledge in the production of a novel design that draws features from both sources [15]. This provides a computational basis for the concept of combination.

This form of knowledge acquisition has applications beyond the design domain. Any domain where there is a uniform representation in the form of genes in a genotype may benefit from this approach.

Acknowledgments

This work is directly supported by a grant from Australian Research Council.

References

[1] Weiss, S. and Kulikowski, C., 1991, *Computer Systems that Learn*, Morgan Kaufmann, Palo Alto, CA.

[2] Goldberg, D.,E., 1989, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, Mass.

[3] Holland, J., 1975, *Adaptation in Natural and Artificial Systems*, University of Michigan, Ann Arbor.

[4] Kirkpatrick, S., Gelatt, C. G. and Vecchi, M., 1983, Optimization by simulated annealing, *Science*, **220** (4598): 671-680.
[5] Crochemore, M., 1994, *Text Algorithms*, Oxford University Press, New York.
[6] Quinlan, J. R., 1986, Induction of decision trees, *Machine Learning*, **1**(1) 81-106.
[7] Salzberg, S, 1995, Locating protein coding regions in human DNA using decision tree algorithm, *Journal of Computational Biology*, **2** (3): 473-485.
[8] Rumelhart, D. E. and McClelland, J. L., 1986, *Parallel Distributed Processing*, Vol. 1, MIT Press, Cambridge, Mass.
[9] Gero. J. S. and Kazakov, V., 1995, Evolving building blocks for genetic algorithms using genetic engineering, *1995 IEEE International Conference on Evolutionary Computing*, Perth, 340-345.
10] McLaughlin, S. and Gero, J. S. 1987, Acquiring expert knowledge from characterized designs, *AI EDAM*, **1** (2): 73-87.
[11] Liggett, R.S., 1985, Optimal spatial arrangement as a quadratic assignment problem, *in*: Gero, J. S. (ed.), *Design Optimization*, Academic Press, New York: 1-40.
[12] Gero, J. S. and Kazakov, V.,1997, Learning and reusing information in space layout problems using genetic engineering, *Artificial Intelligence in Engineering*, **11** (3): 329-334.
[13] Jo, J.H. and Gero, J.S., 1995, Space layout planning using an evolutionary approach, *Architectural Science Review*, **36** (1): 37-46.
[14] Ding, L. and Gero, J. S., 1998, Emerging Chinese traditional architectural style using genetic engineering, *in* X. Huang, S. Yang and H. Wu (eds), *International Conference on Artificial Intelligence for Engineering*, HUST Press, Wuhan, China, 493-498.
[15] Schnier, T. and Gero, J. S., 1998, From Frank Lloyd Wright to Mondrian: transforming evolving representations, *in* I. Parmee (ed.), *Adaptive Computing in Design and Manufacture*, Springer, London, 207-219.