ELSEVIER

PII: S0954-1810(96)00051-9

*Artificial Intelligence in Engineering* **11** (1997) 329–334

Printed in Great Britain
0954-1810/97/$17.00

# Learning and re-using information in space layout planning problems using genetic engineering

**John S. Gero & Vladimir A. Kazakov**

*Key Centre of Design Computing, Department of Architectural and Design Science, The University of Sydney, NSW 2006 Australia*

(Received 5 December 1996)

We describe the use of a genetic engineering version of genetic algorithms as a natural tool for gathering and re-using information about some classes of design problems. This information is stored in the form of sets of 'evolved genes' which are linked to beneficial qualities of the 'good' designs within this class of problems. The approach is illustrated on a space layout planning problem.
© 1997 Elsevier Science Limited.

*Key words:* genetic algorithms, genetic engineering, evolved genes, space layout planning.

## 1 INTRODUCTION

In design practice it is often required to solve the same problem with minor changes many times. Many design problems are solved numerically and the only information which is re-used regularly when the next similar problem is to be solved are the optimal points in the parameter space of the problems already solved. This seems to be a very limited use of the vast bulk of data which is produced during the search for the solution to the problems. Since no regular attempts are made to find and re-use similarities between the structures of the optima found there are no computational savings to be made when moving from one problem to another. In other words if one is solving many slightly different design problems that belong to the same general class then the designer usually has to do approximately the same amount of computation to solve the 10th, the 100th or 1000th problem as to solve the first one. Such inefficiency is caused by the nature of an approach which considers each new problem instance as not being related to the ones already solved. Such an approach does not try to identify generic features of the class of problems being solved and to develop a method for solving the 'standard' generalized problem which represents this class. An efficient computational approach should do just that—identify common generic features of the solutions of the problems from this class using the information about problem instances solved and to construct an optimal solution of the next problem instance by searching among designs which possess these features. It is clear that this method should perform better after a sufficient number of problem instances from this class has been solved and some essential generic features of the class have been found.

In order to see what is the other information that is produced by a search method and which can be re-used to solve similar problems one can think about the standard search process in the following terms: when the problem is being solved some kind of model is being implicitly developed for it in parallel with the search itself (that is, with the generation of better and better points). Although this model is not usually shown or even produced explicitly it actually governs the search. For example, if one uses the Newton method then this model is a quadratic approximation of the objective function. The search is a cycle of two steps. During the first step this model is built or corrected and then used to compute the current direction for the search. The second step is a linear search along this direction. Here the search process produces not only the optimal point but a localized quadratic approximation of the objective over some part of the search space (localized quadratic model of the fitness landscape of the problem in terms of genetic algorithms).

Assume one has solved a few similar problems using the Newton method. When the next such problem is considered the designer can first try to determine what are the generic features of the fitness landscape models

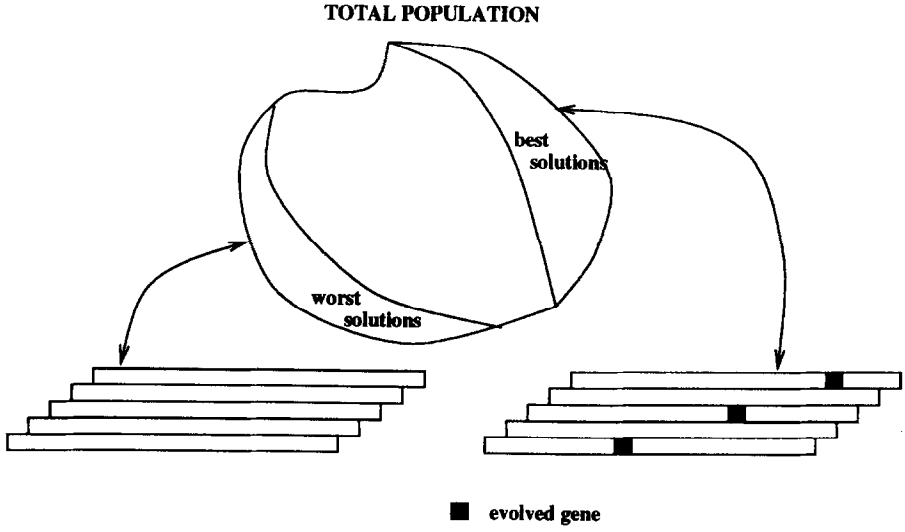


**Fig. 1.** Identification of evolved genes.

already developed and to construct a generalized model of the fitness landscape model of the 'standard' problem instance from the class of the problems considered. Probably, this model will describe only selected areas of actual state space where all (or many) problems are similar. If this model fits the fitness landscape of the next instance well then it is clear that the solution will be found faster than by constructing this model from scratch.

Hence the seemingly inefficient standard approach to the solution of a number of slightly different design optimization problems by using only optima found is caused not by a lack of available data about the class of problems under consideration but by a lack of processing of the data and the lack of learning from the data to build a generalized model of the class of problem from the available instances. That is the consequence of the fact that a majority of computational algorithms are designed as tools for just solving optimization problems and not as tools for knowledge acquisition about the class of optimization problems to which the solved instances belong.

The remainder of this paper is divided into three parts. The first of these briefly describes a genetic engineering extension of genetic algorithms which has the capacity to learn problem-specific information that can be re-used in other instances of the same class of problem. The problem-specific information can be viewed as a partial model of the solution characteristics of the problem being solved. The second part describes an application of the approach being suggested in the area of space layout planning. Here it is demonstrated that two classes of information may be learned by the genetic engineering extension to genetic algorithms. The first class is concerned with the character of the activity interaction matrix whilst the second class is concerned with geometrical characteristics of the solution. The final part discusses the implications of the availability of such learning processes.

## 2 GENETIC ENGINEERING BASED GENETIC ALGORITHMS

Genetic algorithms (GAs)[1,2] are now amongst the most popular search methods. They were designed as computational models of Darwinian evolution theory. Let us show how they work. Assume that any solution of the problems is determined by a vector of parameters. First, a random population of such vectors is generated. Then this population is recursively reproduced by choosing pairs of vectors with probability proportional to the values of their objective functions, cutting these vectors in (usually) two parts and concatenating these parts; this process is called crossover. The components of the resulting vector of parameters are changed randomly (mutated) with some low probability. This is the standard adaptation process of a simple genetic algorithm which generates better and better (increasingly adapted) populations of vectors of parameters.

We have recently proposed an extension to GAs which is based on the model of genetic engineering of natural organisms.[3] Here genetic engineering is concerned with identifying genes, gene sequences or gene structures in the species genome (i.e. the genetic material which is used to represent the organism as the reproduction level) which are directly related to beneficial or dysfunctional performances in a fitness. The goal is then to modify the genome in order to improve it. We will use both the concepts and methods of the genetic engineering of natural organisms. The resulting genetic engineering GA automatically identifies some features of the problem being solved that can be used naturally to build a generalized model of class of problems considered.[4] The automatically produced features are called 'evolved genes'. They are patterns in the parameter space of the problem which were found to be directly linked to the beneficial features of the

solution. Some of these patterns are beneficial for the whole class of problems considered.

In practice, the genetic engineering based GA includes one extra processing step during the standard reproduction cycle of GA and some modification of the standard reproduction operators. This extra step is executed each time when some given degree of adaptation is reached during a standard GA evolution. Then vectors of parameters of the best (with respect to objective function or fitness) solution in the population are compared with the vectors of parameters of the worst solutions (Fig. 1). The goal of this analysis is to find what are the distinctive features of the former group. These could be any arbitrary feature—substring, periodic pattern, etc. The apparatus of string analysis developed in genetic engineering and speech recognition can be used for this purpose.[5] The identified features are declared as the current set of evolved genes. Then the vectors of parameters of the current population are changes in such a way that more of such evolved genes (features) are present. Various realizations of operators which produce these changes have been proposed.[4,7] These operators should be designed in such a way that in addition to the ability to produce evolved genes they would change vectors of parameters as little as possible. After this processing is completed a standard GA resumes using slightly modified reproduction operators.[4,7] These modifications of crossover and mutation operators ensure that the evolved genes are not wiped out during the evolutionary process. After the next given degree of adaptation is achieved the genetic engineering analysis is performed again. The goal now is not only to identify the newly evolved beneficial patterns but also to check the beneficial qualities of the previously identified evolved genes against the current population. Those that do not pass this test (not present exclusively in the most adapted part of current population) are discarded and the newly found ones are added to the current set of evolved genes. This cycle is repeated until either a stationary point is reached or some predefined degree of adaptation is achieved.

In addition to the optimal solution the genetic engineering GA produces a model of the problem solved in the form of an output set of evolved genes. Note that it is possible to view genetic engineering based GA as a formal tool for building a phenomenological (experimental) model of the current problem. It uses abstract system-independent string manipulation techniques which do not require any problem-specific knowledge. Of course, once evolved genes are found (that is, the hypothesis of the casual relationship between the presence of the corresponding pattern in the vector of parameters and the high quality of the corresponding solution is verified experimentally) one can try to find out why it happens. This analysis is not a part of a genetic engineering GA. It requires a knowledge about the system. Let us also note that since usually multiple (and not necessarily unique) parameterizations of the problem can be used, the same feature in a state space of the problem can be mapped onto different features in the parameter space. Therefore it is important to try to identify the parameterization-independent features in a state space which correspond to the evolved genes found. Since it is always clear from the context to which case we refer we will use the term evolved gene with respect to both cases—parameterization-independent and parameterization-dependent.

After solving a number of similar problems one ends up with a number of (partially overlapped) sets of evolved genes. The goal now is to create a generalized set of evolved genes for the whole class of problems based on these sets of evolved genes. We can do that either simply by using their union (as we will do in the following examples) or by applying some processing (for example, by including in it only those evolved genes which were found in a given percentage of the problems solved). When a new problem from the same class needs to be solved we can start its solution not by generating an initial population completely at random, but by making the initial random population biased to the generalized set of evolved genes; that is, by generating random sets of the parameters' vectors which have random numbers of evolved genes each of which is chosen with equal probability from the generalized set of evolved genes. Then an ordinary GA evolution is executed until some degree of adaptation is achieved and the above-described genetic engineering analysis for evolved gene identification is used to find the set of evolved genes which has developed from the generalized ones. The process can be viewed as an adaptation of the models evolved for already solved instances of the problem to the current problem.

## 3 SPACE LAYOUT PLANNING PROBLEM

### 3.1 Introduction

In Ref. 6, genetic engineering GA was applied to a space layout problem (SLP). It was concerned with finding the minimal cost placement (spatial layout) of a set of $m$ 2-d spaces with given areas and shapes (activities) into a given 2-d placement site. The 'forces' of interaction between different activities $f_{i,j}$, $i,j = 0,\ldots,m-1$ are also given. The objective is to find the layout which minimizes the layout cost defined as

$$I = -\sum_{i,j} f_{i,j} d_{i,j}.$$

Here $d_{i,j}$, $i,j = 0,\ldots,m-1$ are distances between activities $i$ and $j$.

A large number of heuristic methods have been developed for SLP problems[8]. One can interpret evolved genes that are produced by genetic engineering GA as primitive forms of heuristic methods for generating a good layout. They are primitive in a sense that they are

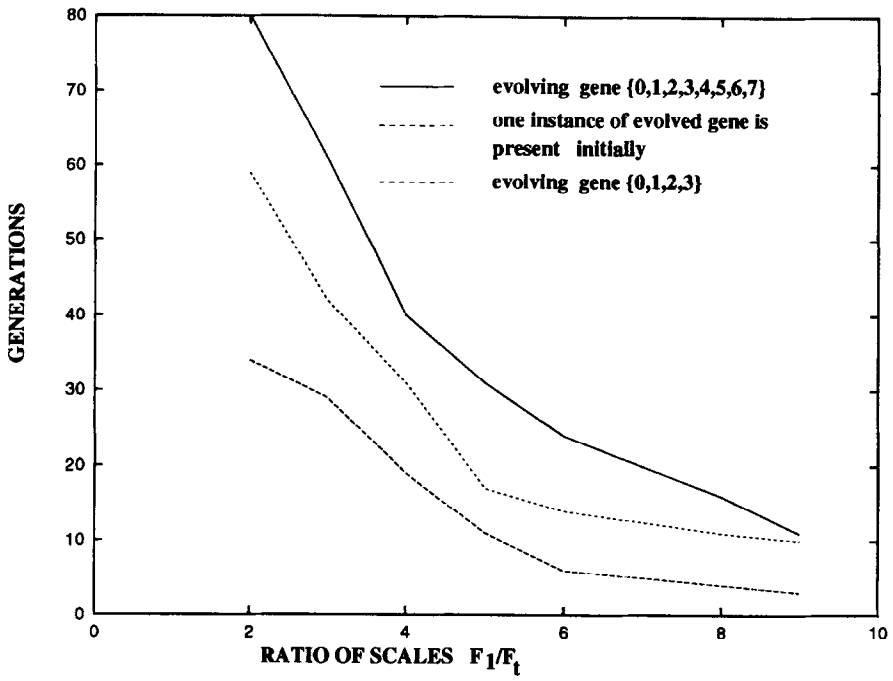


**Fig. 2.** The dependence of the number of generations necessary to run the GA before the evolved gene {0, 1, 2, 3, 4, 5, 6, 7} can be identified on the ratio of the scales of interaction within this group of activities $F_1$ and over the whole set of activities $F_t$. The dotted line shows the dependence of the number of generations which is necessary to run the GA before the evolved gene {0, 1, 2, 3} can be identified in a modified problem with a reduced number of strongly interacting activities on the ratio of the scales.

formulated in terms of a particular class of SLP problems which are similar to the ones already solved (have partly the same sets of activities, partly the same interaction matrices and partly overlapping placement sites). The genetic engineering GA can be viewed as a method for automatic development of specialized heuristics for a particular class of problem.

Since many different paramaterizations can be used to code SLP problems many different types of regularities in parameter space can be found. Nevertheless we have found only two types of generic regularities in the state space of the problem (features of layout structures). Here we will present only these features and not their expressions in the various genetic codings. In practice, these two types of evolved genes rarely exist in a pure form. Usually some mixture of these two types is realized.

### 3.2 Activity interaction-induced evolved genes

The first type of evolved gene is a subset of activities which should be treated as a single activity of undefined shape (soft cluster of activities) during a search for the optimal layout. The positions of the elementary activities within this aggregated activity are not fixed. The position of this aggregated activity within the layout is normally not fixed either (if the SLP problem does not have fixed activities strongly connected to any activity from this group otherwise its position is fixed). In order to see the nature of this type of evolved gene we consider SLP problems with unconstrained placement sites and sets of activities of approximately equal areas and arbitrary shapes. This is a gravity-type problem—the only driving force of the process is the difference in the force of interaction between activities. Let us assume that the forces of interaction within some groups of activities are much stronger than the force of interaction within the whole set of activities of the problem. It turns out that the evolved genes here are the 'super' activities—the groups of activities which should be placed as a compact spatial group in any 'good' layout. Compact means that the diameter of the smallest circle that encloses this group should be small (minimal, if it is feasible). If the layout contains this group in a dispersed form then it can be improved by placing them in a more compact group. The necessary condition for any layout to be non-improvable is the presence of the activities from this set as an adjacent cluster of spaces. The non-geometrical nature of this evolved gene leads to its position-independent character. That is, these 'super' activities can be placed anywhere on the placement site and still lead to overall reduction in layout cost. It is clear that if we introduce some changes to the geometry of the placement site or to the distribution of areas between the activities it is still likely that these 'super' activities will be beneficial in modified problems with the same interaction matrix. The existence of this type of evolved genes is primarily the consequence of the special structure of the activity interaction matrix.

In order to illustrate this evolved gene type we generated 50 random SLP problems with the following conditions: the number of activities $m = 19$, the

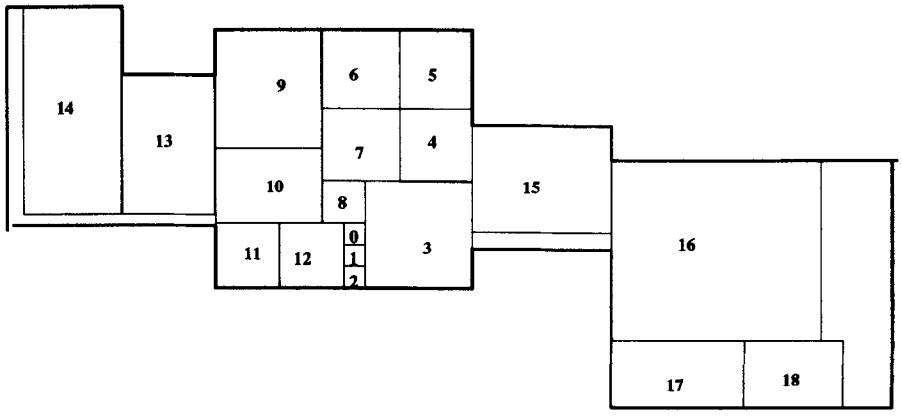


**Fig. 3.** Spaces 0–12 out of 19 spaces located contiguously in a single chamber without any empty spaces.

average area of activity was 30 (measured in elementary squares) with a standard deviation $\sigma = 5$. The elements of the left-upper submatrix of the activity interaction matrices $f_{i,j}$, $i, j = 0, \ldots, 7$ were random numbers with average $S_1$ and $\sigma_{s_1} = s_1/3$. The rest of the activity interaction matrix was filled with random numbers with the average $S_2$ and $\sigma_{s_2} = s_2/3$. The result is a set of problems with a two-scale interaction matrix. The average value of the elements of the interaction matrix over the first group of 8 activities $F_1$ and over the whole set of activities $F_t$ gives estimates of the ratio of these two scales. We measured the number of GA generations necessary to execute in order to identify the evolved gene which corresponds to the 'super' activity made from elementary activities. The dependencies of the numbers of generations on the ratio of these two scales of interaction are shown in Fig. 2. The dashed line here is the result of computations which use random populations which have one instance of this evolved gene (that is, with the initial population weakly biased towards a generalized set of evolved genes). The dotted line corresponds to the same settings except there is a reduction in size of the 'super' group from the first 8 activities to the first 4 (the part of activity interaction matrix $f_{i,j}$ $i, j = 4, 7$ was reset using random numbers with average $S_2$ and $\sigma_{s_2}$). This line shows how fast the development of the pruned 'super' activity {0, 1, 2, 3} proceeds compared to its aggregation from the elementary activities. The population size was 200, $p_{\text{mutation}} = 0{\cdot}01$, $p_{\text{crossover}} = 0{\cdot}6$. The results were averaged over 50 runs with different initial seeds. We consider an evolved gene as developed when it is present in more than 10 instances in the most fit part of population. The computation for placement sites of different shapes produced essentially the same results.

### 3.3 Geometry-induced evolved genes

The second type of evolved gene is a subassembled cluster of activities—a sublayout which is used as a rigid building block embedded into a complete layout. If the problem has a constrained placement site then the sublayout usually has to be placed in a particular location on this site because it fits the subarea of this location especially well.

In order to see the nature of this type of evolved gene we consider an extreme case of the SLP problem where the activity interactions are approximately the same across the whole set of activities and the geometry of the problem is the only driving force which determines the optimal layout structure (this is the same as the bin-packing problem). Here the evolved genes correspond to some local partial placements (the solutions of the local problem of placement of some subset of activities anywhere in the placement set which gives the minimal average distance between the activities of this subgroup but does not change the average distance within the whole set of activities much). Here the very nature of the regularity led to its position dependent character. That is, the evolved gene here corresponds to the fixed sublayout embedded in a complete layout. For example this case takes place when the placement site has a chamber in its middle and there is only one subgroup among the activities which can be fit in it without empty spaces. Any attempt to fit any other activities into it leads to empty spaces in this chamber. This causes an overall spreading out of the resulting layout, an increase in average distances between activities and a rise in layout cost. Figure 3 shows one example result of this class of problems where the evolved gene contains activities 0–12 out of 19 activities, placed contiguously without empty spaces.

## 4 DISCUSSION

Evolved genes represent knowledge learned about the class of problem being solved, knowledge in a re-usable form. It has been demonstrated in Gero and Kazakov[6] that this knowledge results in a significant reduction in computation time to achieve the same quality of solution as the standard GA.

An important question is how similar must the next problem be to gain benefit from the evolved genes. This

is a question which can only be answered empirically, although estimates can be made for each class of problem. Of interest is that evolved genes are composed of lower complexity evolved genes where complexity is defined recursively as: 0-complexity for the initial set of genes, 1-complexity for those evolved genes made up only of 0-complexity genes, 2-complexity for those evolved genes made up of only of 0-complexity and 1-complexity genes, and $i$-complexity is for those evolved genes made up of only evolved (and possibly initial) genes of $(i-1)$-complexity or lower. The lower complexity evolved genes generally represent less problem-specific knowledge which has been learned during the evolutionary process. If the lower complexity evolved genes are also available then the range of applicability of the learned knowledge can be increased and the range of problems which can benefit from this learned knowledge can be increased.

The genetic engineering extension of genetic algorithms provides the foundation for a new class of design tools: design tools that learn.[9] Previous design support tools have been passive, i.e. they have been unchanged by their use. Every time they have been used they take the same computational effort. There are clear and obvious benefits in having tools unchanged by their use as this makes them independent of any problem. However, there are also significant disadvantages in having tools unchanged by their use. Each design that a designer works on adds to the experience of that designer; in this sense a designer learns from each design. For tools to be increasingly useful to designers they should also change with use. A tool that learns about a particular class of problem may not have an enhanced performance for another problem because there is insufficient similarity in either the problem structure or in the problem's numerical description. It is important that the efficacy of such a 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 results of the learning process. This implies that either the tool has the capacity to recognize 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 in unrelated situations. A tool based on the genetic engineering extension to genetic algorithms comes close to this ideal.

## ACKNOWLEDGEMENTS

This work is directly supported by a grant from Australian Research Council, computing resources are provided through the Key Centre of Design Computing.

## REFERENCES

1. Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
2. Holland, J., *Adaptation in Natural and Artificial Systems*. University of Michigan, Ann Arbor, 1975.
3. Sofer, W. H., *Introduction to Genetic Engineering*. Butterworth-Heinemann, Stoneham, 1991.
4. Gero, J. S. & Kazakov, V., Evolving building blocks for genetic algorithms using genetic engineering. *Proceedings of the IEEE Conference on Evolutionary Computing*. 1995, pp. 340–5.
5. Sankoff, D. & Kruskal, J. B. (eds), *Time Warps, String and Macromolecules: the Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.
6. Gero, J. S. & Kazakov, V., Evolving design genes in space layout planning problems. *Artificial Intelligence in Engineering* (in press).
7. Liggett, R. S., Optimal spatial arrangement as a quadratic assignment problem. In *Design Optimization*, Ed. J. S. Gero. Academic Press, New York, 1985, pp. 1–40.
8. Meller, R. & Gau, K. Y., The facility layout problem: recent and emerging trends and perspectives. *Journal of Manufacturing Systems* (in press).
9. Gero, J. S., Design tools that learn: A possible CAD future. In *Information Processing in Civil and Structural Design*, ed. B. Kumar. Civil-Comp Press, Edinburgh, 1996, pp. 17–22.