Learning Representations for Evolutionary Computation

Thorsten Schnier and **John S. Gero**

Key Centre of Design Computing
Department of Architectural and Design Science
University of Sydney, NSW, 2006, Australia
fax: +61-2-351-3031
email : {thorsten,john}@arch.su.edu.au

Abstract

Evolutionary systems have been used in a variety of applications, from turbine design to scheduling problems. The basic algorithms are similar in all these applications, but the representation is always problem specific. Unfortunately, the search time for evolutionary systems very much depends on efficient codings, using problem specific domain knowledge to reduce the size of the search space.

This paper describes an approach, where the user only specifies a very general, basic coding that can be used in a larger variety of problems. The system then learns a more efficient, problem specific coding. To do this, an evolutionary system with variable length coding is used. While the system optimizes an example problem, a meta process identifies successful combinations of genes in the population and combines them into higher level evolved genes. The extraction is repeated iteratively, allowing genes to evolve that have a high level complexity and encode a high number of the original, basic genes. This results in a continuous restructuring of the search space, allowing potentially successful solutions to be found in much shorter search time. The evolved coding can then be used to solve other, related problems. While not excluding any potentially desirable solutions, the evolved coding makes knowledge from the example problem available for the new problem.

The paper shows an example from the domain of two-dimensional shape designs. In this example, a coding is evolved that uses 366 evolved gene, encoding on average 17 basic genes. The evolved coding is successfully used to optimize a room layout, where the original coding converges to a local maximum in fitness and therefore does not find any feasible solutions.

Keywords: Evolving Representation, Evolutionary Learning, Evolutionary Computation, Evolutionary Systems, Genetic Programming.

Learning Representations for Evolutionary Computation

Abstract

Evolutionary systems have been used in a variety of applications, from turbine design to scheduling problems. The basic algorithms are similar in all these applications, but the representation is always problem specific. Unfortunately, the search time for evolutionary systems very much depends on efficient codings, using problem specific domain knowledge to reduce the size of the search space.

This paper describes an approach, where the user only specifies a very general, basic coding that can be used in a larger variety of problems. The system then learns a more efficient, problem specific coding. To do this, an evolutionary system with variable length coding is used. While the system optimizes an example problem, a meta process identifies successful combinations of genes in the population and combines them into higher level evolved genes. The extraction is repeated iteratively, allowing genes to evolve that have a high level complexity and encode a high number of the original, basic genes. This results in a continuous restructuring of the search space, allowing potentially successful solutions to be found in much shorter search time. The evolved coding can then be used to solve other, related problems. While not excluding any potentially desirable solutions, the evolved coding makes knowledge from the example problem available for the new problem.

The paper shows an example from the domain of two-dimensional shape designs. In this example, a coding is evolved that uses 366 evolved gene, encoding on average 17 basic genes. The evolved coding is successfully used to optimize a room layout, where the original coding converges to a local maximum in fitness and therefore does not find any feasible solutions.

1 Introduction

Computational methods based on evolutionary principles have been around for about 25 years (see (Spears, Jong, Baeck, Vogel & de Garis 1993) for an overview), and the last five years have seen a strong increase in interest in them. However, they are still far from being used as a routine tool in research and development. One main reason is that, while the general algorithms are usable for a fairly large variety of applications, the coding of the problem usually has to be designed and optimized for every application. This is not a simple task, and coding problems are often the reason for unsuccessful applications of evolutionary methods.

In cases where large spaces have to be searched, evolutionary systems are often superior to other search algorithms, at least as long as no heuristic is known that can be used to guide the search. But still, the search time of evolutionary systems usually grows very fast with the number of variables. For example in standard genetic algorithms, the convergence time grows exponentially with the length of the genetic code.

In addition to the number of variables, at least two other factors can influence the performance of an application of evolutionary systems:

- in applications where not all possible genetic codes translate into valid individuals, either large portions of the offspring are invalid, or special problem specific genetic operations have to be introduced;

- any reduction in the number of variables or special genetic operations may exclude solutions from the search space.

The goal, therefore, is to find a representation with a minimal number of variables, while at the same time not producing a high number of illegal offspring and not excluding any possible solutions. This is difficult, and at the same time tends to be very specific for the applications it is developed for.

One possible way out of this is to use an approach where only a very basic, general representation is specified, and the systems successively learns an improved representation while it is looking for solutions. Moreover, the resulting coding can then be used to solve other, similar problems. The basic coding can be general enough to be applicable to a large variety of related problems. This paper will show how such an approach can be realized.

2 Evolving Representation

Like most evolutionary systems, the starting point is a population of randomly created individuals. The coding of these individuals, the 'basic' genes, is chosen to be very low-level, putting as little domain knowledge into the coding as possible, and making sure not to exclude any interesting part of the search space. The only condition is that the coding has to allow for genetic codes of variable length.

The individuals are than subjected to the cycle of replication with errors and survival of the fittest. At the same time, an additional operation screens the population, identifying particularly successful combinations of genes. For every such gene combination, a new, 'evolved' gene is created that represents this combination, and is introduced into the population. In the first few cycles, the evolved genes will be composed from basic genes, but in later cycles most evolved genes will represent combinations of other, lower level evolved genes, or combinations of those with basic genes. This growing hierarchy of representations gives rise to a more and more complex and abstract coding, that is increasingly adapted to the application. In other words, the process gradually collects application specific knowledge and codes it into the representation, rather than being coded into it by the user in the first place.

What does an evolving representation mean in terms of evolutionary systems? Since an evolved gene is an "atom" for the cross-over operation, the low level genes that are represented by it are protected from being cut apart by this operation. The genes that are "glued together" by this operation are by definition successful, that is they are highly represented in successful individuals. The schema theorem (Holland 1992) explains the success of genetic algorithms with the fact that successful schemata will be much more often represented in the population, and therefore be less likely to be cut apart. The genes selected for evolved genes are special instances of these schemas, and the creation of high level genes has a similar effect. The difference is that the genes are totally protected from cross-over, and that not only the high number of occurences of the schema in the population is regarded in the selection pro-

cess, but also the success of the individuals it occurs in. Thus, the creation of genes amplifies and improves the propagation of successful schemas.

However, learning a representation for a one time application is not the only goal scheme presented here. Another advantage appears if the evolved representation is used for other, similar search problems. Here, the initial population is generated using both the original and the evolved genes. The presence of the original genes ensures that the whole original search space can still be searched. The evolved genes however restructure the search space in favour of structures that are already established as useful when the genes originally evolved. The following illustrates this.

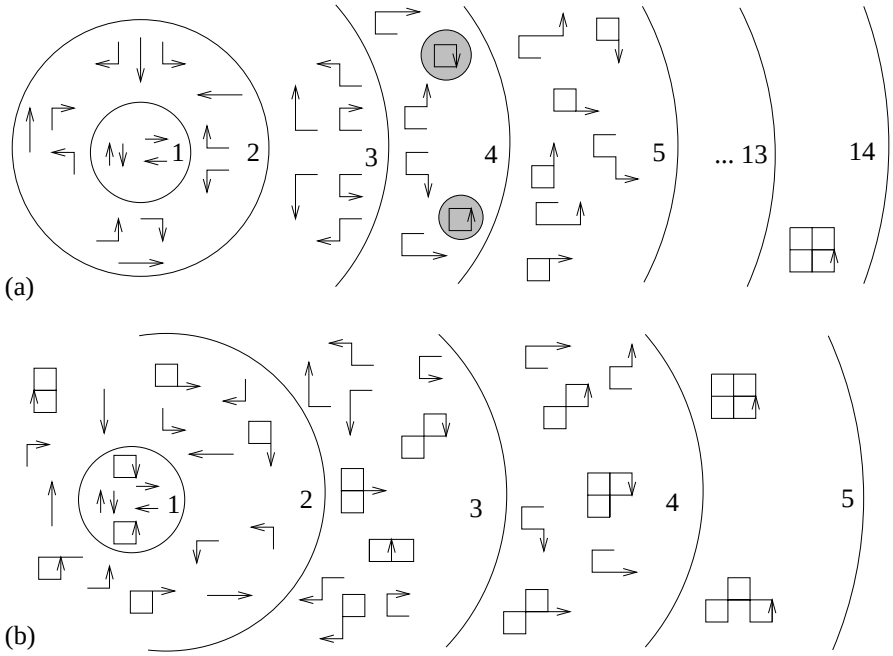

Figure 1: Example of an evolving representation: (a) original representation and (b) representation with evolved genes

The length of the genotype is not restricted, therefore the search space is infinitely large. It can be illustrated by an (infinite) number of concentric circles, each defining the space of solutions that can be defined by a genotype of a certain length. The inner circle contains the genotypes of length one, i.e the basic building blocks. The further away a solution is from the centre, the more difficult it is to find by means of search. Every time an evolved gene is created, the structure of the search space is changed. The locus of the new gene in the search space is moved into the centre, all solutions in the next circle that can be derived from that solution are moved into the second circle, and so on. Figure 1 illustrates this: the original search space is illustrated in Figure 1(a), with the four basic building blocks in the centre. The second circle shows all solutions that can be derived from genes of length two (i.e., using two building blocks). The other circles give some examples of solutions using genotypes of length 3, 4 and 5. If now the two closed shapes in the fourth circle are identified as particularly successful and an evolved gene is introduced for both of them, the search space changes as shown in Figure 1(b): the squares are now basic building blocks, and the shapes on the fifth circle that

are derived from the squares can now be found in the second circle. The more evolved genes a solution involves, the more it is moved towards the centre. For example, the shape with the four squares that is now on the fifth circle (i.e. can be constructed from genotypes of length five) would have been on the fourteenth circle before.

The introduction of evolved genes obviously changes the probability that a gene sequence maps onto a useful feature. While the number of different genes that can be used in a genotype expands, the length in the genotype that is necessary to describe a feature shrinks drastically. As an example, in Figure 2(a) a search space with the basic coding is drawn, useful solutions (black dots) may be far from the centre and from each other. After the representation has evolved, the solutions lie much closer together. The part of the search space that has to be searched to find these solutions is now much smaller, Figure 2(b).

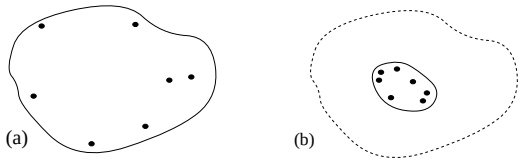

Figure 2: Effect of evolving representation: (a) original representation, black dots representing interesting solutions; and (b) evolved representation

Some numbers from the example discussed below may give a rough estimate of the effect: In that example, 366 evolved genes were created. Expressed in basic genes, these evolved genes have a median length of about 17. Imagine a feature that can be described by two of these high level genes. To find this sequence among all genotypes of the same length, the probability is $\frac{1}{366^2} = \frac{1}{133956}$. If this feature were to be expressed in terms of the four different basic genes only, it would need a genotype of length 34. The probability here is $\frac{1}{4^{34}} \approx \frac{1}{2.9exp20}$. Since evolutionary search is not a blind search the actual difference is less.

3 Example of evolving representation

As an example, we show how the idea can be used in the design of shapes. As basic coding, we use a turtle graphics-like coding with only four different basic genes, that either draw a line in the current direction, move the pen ahead or change the current direction (see Figure 3).

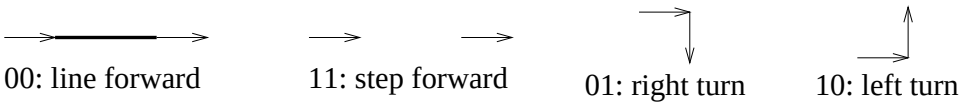

Figure 3: Basic coding, arrows show pen position and current direction before and after the gene is drawn.

This coding is able to represent any kind of two dimensional shape composed of horizontal and vertical straight lines, and has been used for building elevations and floor plans. An extension to three dimensions and different shapes is straight forward.

3.1 Evolving the representation

The example discussed here uses the coding above to create floor plans, the two floor plans shown in Figure 4 are used together as the designs to be represented. The fitness function compares individuals with this drawing, and rewards individuals depending on how much of the drawing they fit.

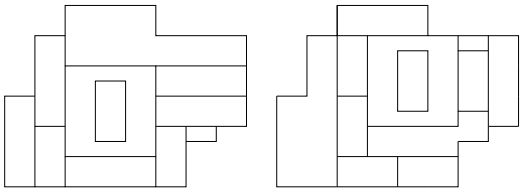

Figure 4: Room plans used as example cases

To create evolved genes, successful combinations of genes in the population have to be identified. Ideally, this has to be any number and combination of genes in the genotypes. However, any composition of more than two genes can be constructed from a number of compositions of two genes. For example to combine three genes, in a first step an evolved gene is created that combines two of the original genes, and in the next step this new evolved gene is combined with the third original gene. An additional feature of the basic representation used here allows further simplification: the further apart two genes are in the genotype, the less likely they are to depend on each other in the fitness. Therefore, we have only to consider pairs of successive genes in the creation of evolved genes, using the following.

1. Create a table of all different pairs of successive genes that occur in the population.
2. For all individuals in the population: divide the fitness by the length of the individual. In the table, add this value to all pairs occuring in the genotype of that individual.
3. Find the pair with the highest sum of fitnesses in the table.
4. Create a new evolved gene with a unique designator, and replace all occurrences of the pair in the population with the new evolved gene.

The number of evolved genes is kept to a certain percentage of the population (3% in the examples shown).

Figure 5(a) shows a branch of the hierarchical composition of one of the evolved genes (no. 363) from lower-level evolved genes and basic genes (numbers in brackets). Figure 5(b) shows how an individual is composed from six evolved genes.

3.2 Using the representation

After a representation based on the examples in Figure 4 has been developed, it is used for new, different fitness requirements. A standard evolutionary algorithm is used, where the fitness requirements are coded into the fitness function. The representation is not evolved further, instead the set of evolved l genes learned from the examples is used, together with the original basic genes.

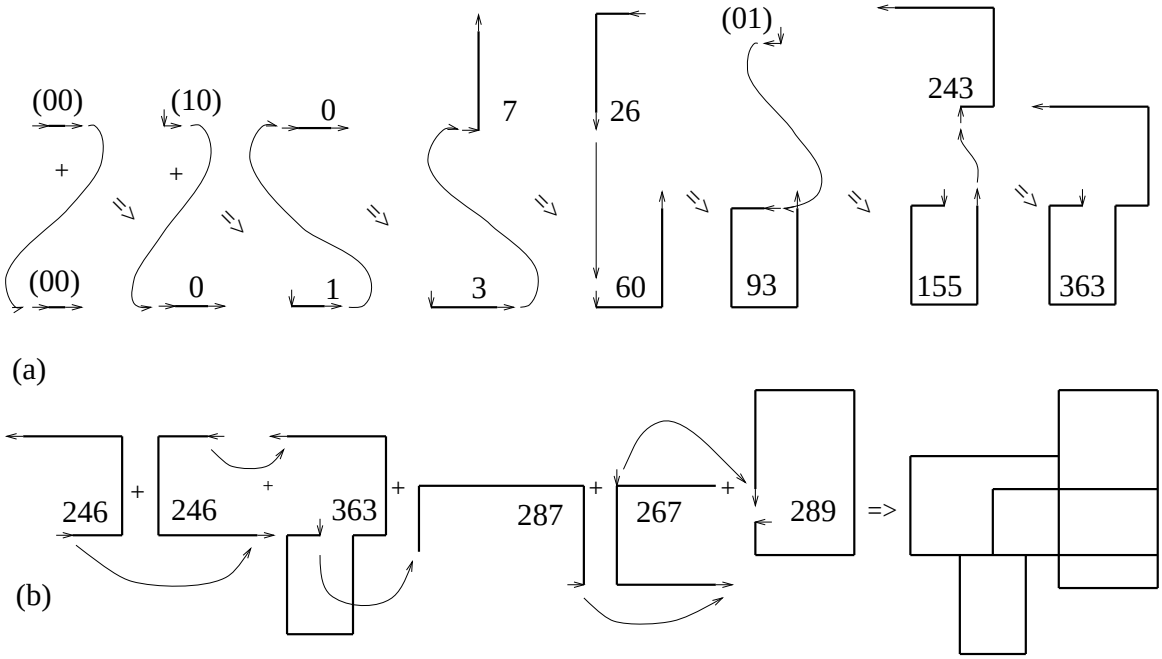

Figure 5: Evolved representation: (a) part of the hierarchical composition of the evolved gene 363 and (b) composition of the individual with the genotype (289 267 287 363 246 246)

As an example, the new requirement was to create a floor plan with minimal overall wall length, while at the same time fulfilling the following additional requirements:

- no walls with "open ends", that is walls that do not build a closed room
- 6 rooms, sizes 200, 150, 100, 75, and 50 units

The additional requirements were given higher priority than the minimization of the wall length.

Figure 6 shows the result of one run, after 100000 cross-overs were performed. The population size was 1000 individuals, and 366 evolved genes created from the examples in Figure 4 have been used. They were introduced into the population by using them with equal probability in the generation of the initial random population, and by the mutation operator. Shown are the best individuals, with the exception that individuals that are rotated copies of already drawn individuals have been omitted.

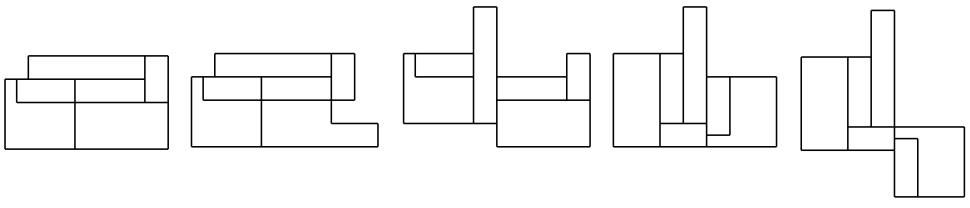

Figure 6: New floor plans, using coding knowledge from the example cases. See text for fitness requirements.

As described above, every evolved gene represents a sequence of basic genes, and the length of this sequence directly corresponds to the amount of information the evolved gene contains about the example case. An analysis of the lengths of genes used in the genotypes of the final population can therefore show how much information from the example case is used in the new results. Figure 7 shows a histogram for the example application. The peak at the left contains the four basic genes, which together are used about 900 times in the final population of 500 individuals. Many of these occurrences are left and right turns, due to the fact that while identical individuals are not permitted in the population, adding a turn at the front of the code creates a new individual with the same fitness as the original code. Most of the evolved genes in the population have lengths of around 12, but the final results also use a considerable number of evolved genes with lengths around 40. This shows that the new results indeed make use of the evolved representation to a very large degree.

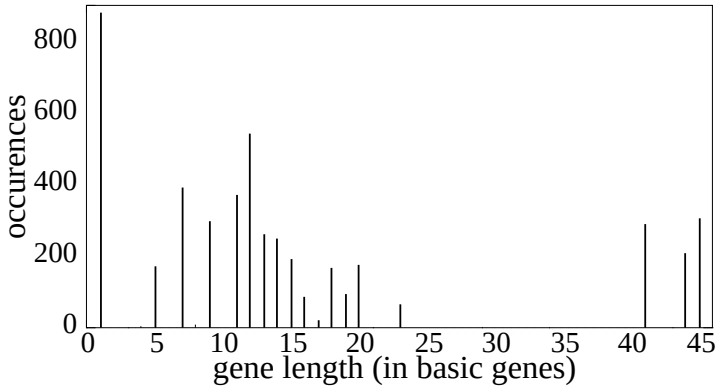

Figure 7: Histogram of the lengths of evolved genes (in basic genes) used by the new results

4 Related Work

Evolving coding has been introduced into genetic coding (Koza 1992), for example in (Angeline & Pollack 1992) and (Koza 1994). The "evolved genes" in those examples are usually functions composed from larger "chunks" of lisp code, and the extraction is not explicitly based on particularly high success. Also, the function generation usually does not exceed a few levels of complexity (compared with e.g. 7 in Figure 5).

5 Discussion

If the basic coding of Figure 3 is used without evolving coding to solve the example problem used in Section 3.2, it still finds solutions with 6 rooms, but all of them have only unit size. Since any lines that do not totally enclose a room are penalized by the fitness function, the evolutionary system has to create the next larger room size in "one step" from the small rooms. In other words, the first local maximum found in the fitness function is a layout with six unit-sized rooms, and the distance to the next local maximum is too far to be found easily by either mutation or cross-over. The "granularity of possible solutions" seems to be too fine compared

with the "granularity of solutions rewarded by the fitness function". It is important to see the difference between this fitness function and the one used to create the evolved coding. The latter is smooth, and any improvement in the code is rewarded appropriately. If the fitness of Section 3.2 had been used to evolve a coding, the system would have been no more able to find larger rooms than without evolving coding. This highlights the point that the gain is not the evolving coding, but the evolved coding when used in similar applications.

6 Acknowledgments

To come.

References

[1] Peter J. Angeline and Jordan B. Pollack. Coevolving high-level representations. In Christopher G. Langton, editor, *Artificial Life III*, pages 55–72. Santa Fe Institute, Addison Wesley, June 1992.

[2] John H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, Massachusetts, 2 edition, 1992.

[3] John R. Koza. *Genetic Programming.* MIT Press, 1992.

[4] John R. Koza. Architecture-altering operations for evolving the architecture of a mutilpart program in genetic programmin. Technical report, Computer Science Department, Stanford University, Margret Jacks Hall, Stanford, California 94305-2140 USA, October 1994.

[5] William M. Spears, Kenneth A. De Jong, Thomas Baeck, David B. Vogel, and Hugo de Garis. An overview of evoluationary computation. In *Machine Learning: ECML-93*, pages 442–459. Vienna, Austria, April 5-7, Springer Verlag, 1993.