N. Gu, S. Watanabe, H. Erhan, M. Hank Haeusler, W. Huang, R. Sosa (eds.), *Rethinking Comprehensive Design: Speculative Counterculture, Proceedings of the 19th International Conference on Computer-Aided Architectural Design Research in Asia CAADRIA 2014,* 523–532.

EVOLVING LEGO

Exploring the impact of alternative encodings on the performance of evolutionary algorithms

PATRICK JANSSEN[1] and VIGNESH KAUSHIK[2]
[1,2] *National University of Singapore, Singapore*
[1] *patrick@janssen.name*
[2] *vigneshkaushik@gmail.com*

Abstract. In evolutionary design algorithms, the evolutionary procedures have a major impact on the quality of the genotype-fitness mapping, which in turn impacts the performance of the algorithm. Two key issues affecting the quality of a mapping are the size of the genotype space and the locality of the mapping. In order to systematically investigate the role that genotype space and locality have on evolutionary performance, a set of experiments are conducted using benchmark test cases consisting of simple LEGO structures. Three different developmental procedures are implemented and tested. The results confirm that locality is critical in achieving good performance and in some cases may have a greater impact than genotype length.

Keywords. Evolutionary design; Evolutionary performance; Locality; Genotype-Fitness mapping.

1. Introduction

Evolutionary design is an approach that evolves populations of design variants in order to optimise certain performance measures. Designs are manipulated by a set of computational procedures, including a development procedure for generating design variants, one or more evaluation procedures for calculating evaluation scores, and a feedback procedure for closing the loop by assigning fitness scores to groups of individuals and then performing selection and reproduction.

The way that the evolutionary procedures are defined has a major impact on the performance of the evolutionary algorithm. In this case, performance refers to how effectively the evolutionary algorithm is able to converge on

certain optimal design variants. The performance of an evolutionary algorithm is usually measured in terms of the best fitness in the population, averaged over multiple runs.

The evolutionary procedures are often described as a mapping from genotype to phenotype to fitness. High quality mappings result in better performance, while low quality mappings result in worse performance. Two key issues affecting the quality of a mapping are the size of the genotype space and the locality of the mapping.

The size of the genotype space is defined by the number of genes and the range of values that each of those genes can have. The smaller the genotype space, the easier it will be for the evolutionary algorithm to search that space. For each gene added to the genotype, an extra dimension is added to the genotype space, thereby resulting in an exponential increase in the number of possible design variants.

The genotype-fitness mapping can be described using the concept of *locality* (Rothlauf and Goldberg 2003, Rothlauf and Oetzel 2006, Galvan-Lopez et al. 2011). High locality means that small changes to the genotype should result in small changes to the fitness. In terms of the genotype and fitness spaces, this means that points that are close together in the genotype space should also tend to be close together in the fitness space.

In theory, the role of the phenotype in this mapping is secondary. In practice however, the genotype to fitness mapping must pass via a phenotype stage, which means that with local mappings, small changes to the genotype should result in small changes to the phenotype, which in turn should result in small changes in fitness.

For complex design problems, developmental procedures have a big impact on the quality of the mapping. When defining a developmental procedure for design problems with complex constraints, these two issues are typically related – with the size of the genotype space being inversely linked to the locality of the mapping. For example, for a particular design problem, it may be possible to implement a developmental procedure in two different ways. The first way may use very few genes combined with a very complex set of developmental rules, resulting in low locality. The second way may use a lot of genes combined with a relatively simple developmental rules, resulting in high locality. In such a case, it is important to understand which has the greater impact on the performance of the evolutionary algorithm – the size of the genotype space or the locality of the mapping.

In order to systematically investigate this question, a set of experiments were conducted using benchmark test cases consisting of simple LEGO structures. Three different developmental procedures were implemented and

tested. Section two describes the benchmark, section three presents the experiment, and section four draws conclusions.

2. Proposed Benchmark

For the benchmark, a spatial problem was required that incorporated various spatial and geometric constraints. The construction of LEGO structures was chosen for the benchmark as it is a well-known spatial construction system for which tacit knowledge can be assumed.

For the benchmark, a set of target LEGO brick structures are defined, and the aim of the evolutionary algorithm is then to evolve structures that match these target structures. The key parameters for the structures are as follows:

- The structure consists of 24 standard LEGO bricks, each with 2 x 4 pegs.
- The structure is built on base plate that is 12 x 16 pegs in size, with a maximum height of 5 bricks.

Three different target structures are defined, as shown in Figure 1.

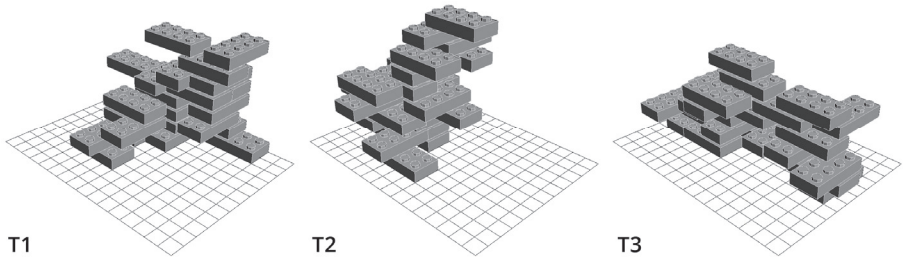

Figure 1: Three randomly generated LEGO structures to be used as target structures for the benchmark experiments.

2.1. EVOLUTIONARY PROCEDURES

The aim is to define developmental procedures capable of generating all three LEGO structures. Together with an evaluation procedure and a feedback procedure, these procedures can then be used within an evolutionary algorithm in order to evolve LEGO structures.

For the evaluation procedure, the fitness of an evolved LEGO structure is calculated by comparing the positions of pegs in the evolved structure to the positions of the pegs in the target structure. The fitness is defined as the mismatch between these peg positions, so that if the mismatch score is zero, then the evolved structure will exactly match the target structure. The mismatch is calculated as follows:

- For each peg in the target structure with no corresponding peg in the evolved structure, the mismatch score is incremented by one.
- If the evolved structure has a total number of pegs that is either lower than or higher than target structure, then the difference in the number of pegs is added to the mismatch score.

In this case, due to the simplicity of the fitness calculation, the mapping from the phenotype space to the fitness space can be assumed to have very high locality. This research therefore focuses on the developmental procedure that defines the mapping from genotype to phenotype space.

For the feedback procedure, a steady state algorithm was used. This algorithm, and the associated settings, will be discussed in more detail at the end of section three.

2.2. MEASURING PERFORMANCE

In order to be able to investigate the impact that the developmental procedures have on the performance of the evolutionary algorithm, we must first define an unambiguous way of measuring this performance.

For a particular run of the evolutionary algorithm, the performance is defined as the best structure in the population (i.e. the structure with the highest fitness) after a certain predefined number of structures have been generated. In this case, this number is set to 3000, as this has been found to be sufficient in order to be able to see the impact of the developmental procedure. In addition, in order to see how the performance changes over time, the best structure in each generation is recorded.

Since evolutionary algorithms are stochastic in nature, the algorithm will need to be run multiple times so that performance can be calculated as an average of all runs. In general, 10 runs is deemed to be sufficient in order to get a good measure of average performance. In this research, for each developmental procedure, the evolutionary algorithm was run 10 times per target structure, resulting in a total of 30 runs. The final performance was then calculated as the average of all these runs.

3. Developmental Procedures

In order to explore the impact of the developmental procedure on the performance of the evolutionary algorithm, three different procedures were defined. For these procedures, the size of the genotype space and the locality of the genotype-fitness mapping differed, so the performance of the evolutionary algorithm was also expected to differ depending on which of these factors had a greater impact.

Developmental procedures that do not disallow invalid structures are sometimes used, but they need to be combined with penalty functions and repair functions (Eiben and Smith 2003, pp 210-211). Previous experiments with such procedures and functions found that they performed very badly due to the fact that a very large proportion of structures being generated actually end up being invalid, thereby significantly hindering the overall evolutionary search process.

When defining the developmental procedures, it was therefore decided at the outset that the procedure should be guaranteed to only generate valid LEGO structures. Such structures have three main constraints:

- Bricks that are floating in space are disallowed.
- Bricks that are intersecting with one another are disallowed.
- Bricks that stray beyond the perimeter of the base plate are disallowed.
- Brick structures consisting of multiple disconnected parts are disallowed.

In total, three such procedures were defined. In all cases, these procedures use an encoding technique called *decision chain encoding* (Janssen 2004, Janssen and Kaushik 2013). The key feature of this encoding technique is its ability to control variability by handling complex sets of constraints.

3.1. DECISION CHAIN ENCODING

The decision chain encoding technique defines the LEGO structure generation process as a sequential chain of decision points, where each decision points involves placing the next brick. Each time a brick is placed, a choice must be made that selects the position of the brick from a list of possible options. The choices are made based on the gene values, so that a different set of genes will lead to a different set of choices.

At each decision point, the list of possible brick options depends on the decisions made up to that point. Thus, neither the options nor the number of options can be known in advance. The list of brick options is created dynamically at each decision point in a two-step process. In the first step, bricks are placed on all currently exposed pegs in all possible positions and orientations. In the second step, all brick options that would result in an illegal structure (for example, due to intersecting bricks) are filtered out, leaving only the valid options.

Once the list of brick options has been generated, the genes then need to be used to somehow specify which option to choose. It is in this aspect that the three developmental procedures differ. In all cases, the genotypes consist sequences of real valued genes in the range $\{0,1\}$. However, the number of

genes and the ways the genes are used in order to select options is different in each case.

All three procedures use the same general approach, using an intermediary spatial representation, referred to as the *decision space*. At each decision point, a set of genes are used to specify a point in the decision space, referred to as *selector*. All valid brick options are then also mapped into points in the same decision space and the brick option that is closest to the selector is the selected. The three procedures differ in the types of decision spaces they define and the ways they map brick options into points in the decision space.

3.2. DEVELOPMENT PROCEDURE 1

The first development procedure uses a 1D decision space, and therefore only requires one gene for each decision point, resulting in a total of 24 genes. At each decision point, the decision gene, labelled *t*, specifies a selector on a 1D line segment, one unit in length. Valid brick options at that decision point are mapped to unique points on the line segment, and the option that is closest to the selector is selected.

The process of selecting a valid brick option can be conceptualized as a spatial transformation of brick options from the actual 3D space to the 1D decision space. The brick options are sorted into a sequence and then evenly distributed along this one dimensional line segment.

The key part of this procedure is the way that the brick options are sorted. The problem is that there is no natural way of sorting points in space in a way that ensures that brick options that are close together in the actual space are also close together in the sorted sequence. This will reduce locality, therefore resulting in a lower quality mapping.

3.3. DEVELOPMENT PROCEDURE 2

The second developmental procedure uses a 3D decision space, and therefore requires three genes for each decision point, resulting in a total of 72 genes. At each decision point, the three genes, labelled *u*, *v*, and *w*, specify a selector within a 3D cubic volume, 1 x 1 x 1 units in size. Valid brick options at that decision point are mapped to unique points in the cube, and the option that is closest to the selector is selected.

This can be conceptualized as a spatial transformation of brick options from the actual 3D space to a 3D decision space. However, unlike procedure 1, in this case the spatial transformation is very direct, thereby ensuring that reasonable locality is achieved.

For this procedure, a different issue arises related to the spatial transformation. In the case of procedure 1, the spatial transformation ensures that

each option gets equal chance of being selected, since all options are distributed equally along the 1D line segment. However, in this case, options are clustered in space in an uneven manner, and as a result, their chances of being selected will vary. This is clearly visible when the decision space is represented as a 3D Voronoi partitioning as shown in Figure 2. Each Voronoi cell represents one option, and if the decision selector is inside the cell, then that is the option that is selected. Cells with few neighbouring options will tend to have a higher chance of being selected since they tend to be much bigger. This bias towards certain brick options may mislead the evolutionary search process.

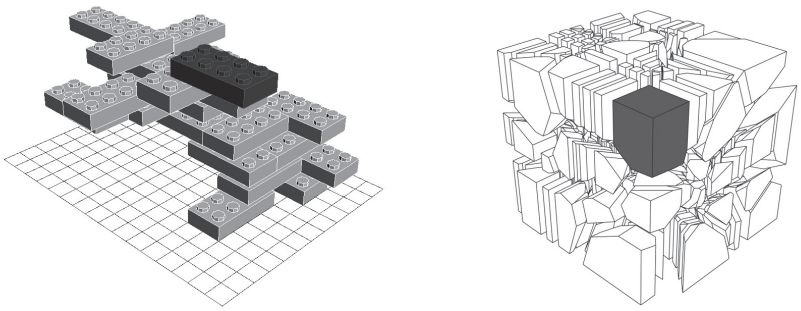

Figure 2: Spatial transformation between the actual space (left) and the decision space (right). The brick options in the actual space are transformed into points in the decision space from which a Voronoi partitioning is then generated.

3.4. DEVELOPMENT PROCEDURE 3

The third developmental procedure uses two different decision spaces, a two dimensional space and a one dimensional space. Three genes are required for each decision point, resulting in a total of 72 genes.

The reason for using two decision spaces is to reduce the biases introduced by procedure 2 (as explained in more detail below). These two decision spaces are used in a two-step process. First, the 2D decision space is used for selecting a group of brick options that are vertically stacked on top of each other. Second, the 1D decision space is used for selecting one option from the group of options.

In the first step, the two genes, labelled *u* and *v*, specify a selector on a 2D plane, 1 x 1 units in size. Groups of brick options are mapped to unique points on the plane, and the group of options that is closest to the selector is selected. The groups are defined as follows: for each peg on the base plate, one group is created consisting of all brick options that intersect the z axis of the peg. Empty groups are discarded, so pegs with no brick options above

them are ignored. Note also that this allows brick options to be in more than one group. (Since each brick has 8 pegs, it can be in up to 8 groups.) The mapping of groups to unique points on the decision plane is based on the position of the group's base peg.

In the second step, the third gene, labelled *t*, specifies a selector on a 1D line, one unit in length. Brick options in previously selected group are then mapped to unique points on the line segment, and the option that is closest to the selector is selected. This step works in a way that is similar to procedure 1. However, a key difference that the problem of sorting the brick options is now less problematic: brick options are sorted according to their height, as well as a few other factors such as position and orientation. .

In this case, two spatial transformations are involved: groups of brick options are mapped from the actual 3D space to a 2D decision space, and subsequently from the selected group of brick options in 3D space to a 1D decision space.

The two spatial transformations remain fairly direct, thereby ensuring that reasonable locality is achieved. In addition, the bias of procedure 2 towards certain options has also been reduced. In the second step, no bias exists, since all brick options within the selected group are evenly distributed along the decision line. In the first step, the groups of brick options are mapped to unique points in the decision plane. These groups of options are clustered on the plane in an uneven manner, again resulting in certain biases, as can be seen when a Voronoi partitioning of the plane is created. However, each brick can be in more than one group, thereby increasing the number of ways a brick can be selected and also reducing the bias to some extent.

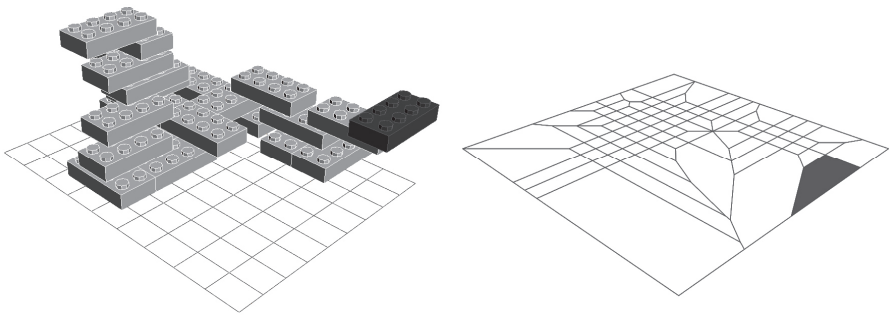

Figure 3: Spatial transformation between the actual space (left) and the decision space (right). The groups of brick options in the actual space are transformed into points in the decision space from which a 2D Voronoi partitioning is then generated.

3.5. RESULTS

A steady-state evolutionary algorithm was used (De Jong 1975, Whitley and Kauth 1988). After a number of initial experiments, the following settings were found to work well:

- A population size of 100, with 10 individuals being replaced each generation.
- For the feedback procedure, fitness proportionate selection was used for the selection strategy (i.e. for selecting parent individuals for reproduction) and elitist selection for the replacement strategy (i.e. for selecting the worst individuals to be deleted).
- For the reproduction operators, standard crossover and mutation operators were used. The crossover probability was set to 1, while the mutation probability was calculated as 1 / n, where n is the number of genes (i.e. 1/24 for development procedure 1, and 1/72 for development procedures 2 and 3).

For each development procedure, the evolutionary algorithm was run 30 times (10 times for each of the three targets). For each generation, the average of all the best individuals for all 30 runs was calculated and plotted on a graph, and shown in Figure 4.

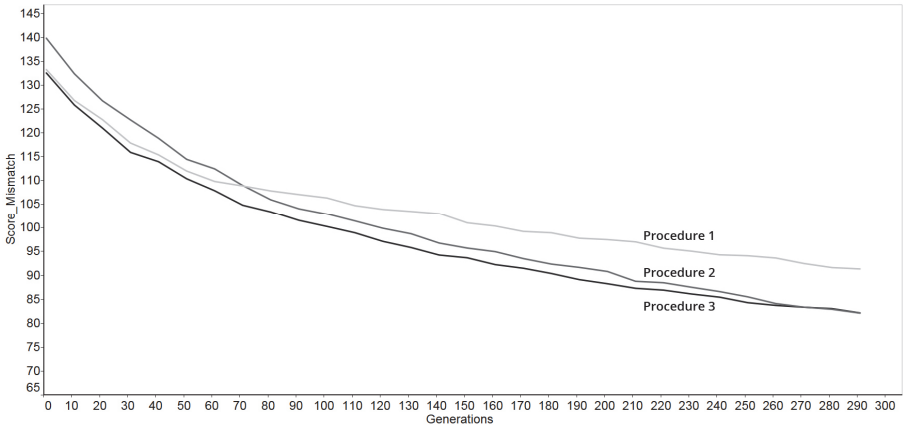

Figure 4: Averaged results for the three developmental procedures.

The graph shows that the evolutionary algorithms using development procedures 2 and 3 perform significantly better than those using development procedure 1. Procedures 2 and 3 have genotypes that are three times as long, and the search space therefore has three times as many dimensions. Considering only the size of the search space, procedures 2 and 3 would actually be expected to result in worse performance rather than better performance. The reason for the better performance is likely to be the higher local-

ity of the developmental procedures. The graph therefore confirms the importance of having high locality.

Regarding procedures 2 and 3, they result in similar performance. In procedure 3, an attempt was made to reduce to some degree the biases introduced into procedure 2 due to the uneven partitioning of the genotype space. Although procedure 3 does tend to perform a little better, the difference is not significant enough to be able to draw any general conclusions.

4. Conclusion

The research set out to investigate the role that genotype space and locality have on evolutionary performance. A set of experiments were conducted using benchmark test cases consisting of simple LEGO structures. Three different developmental procedures were implemented and tested. The results confirm that locality is critical in achieving good performance.

Future research will focus on developing techniques for measuring locality for complex design problems requiring non-binary genotypes and multiple objective optimisation. Such techniques will enable a better understanding of the relationship between locality and evolutionary performance.

References

De Jong, K. A.: 1975, *An analysis of the behaviour of a class of genetic adaptive systems*. Doctoral dissertation, University of Michigan, Ann Arbor.

Eiben, A.E. and Smith, J.E.: 2003, *Introduction to Evolutionary Computing*. Springer, Natural Computing Series, 1st edition.

Galván-López, E., McDermott, J., O'Neill, M., Brabazon, A.: 2011, Defining locality as a problem difficulty measure in genetic programming, *Genetic Programming and Evolvable Machines*, **12**(4), 365–401

Janssen, P.: 2004, *A design method and computational architecture for generating and evolving building designs.* Doctoral Dissertation, Hong Kong Polytechnic University.

Janssen, P. and Kaushik, V: 2013, Decision Chain Encoding: Evolutionary Design Optimization with Complex Constraints, *Proceedings of the 2nd EvoMUSART Conference*, 157–167.

Janssen, P., Chen, K.W. and Basol, C: 2011, Iterative Virtual Prototyping: Performance Based Design Exploration, *Proceedings of the eCAADe Conference*, Ljubljana, Slovenia, 253–260.

Janssen, P. and Chen, K.W.: 2011, Visual Dataflow Modelling: A Comparison of Three Systems, *Proceedings of the CAAD Futures Conference*, Liege, Belgium, 801–816.

Rothlauf, F. and Goldberg, D.: 2003, Redundant Representations in Evolutionary Algorithms, *Evolutionary Computation*, **11**(4), 381–415.

Rothlauf, F. and Oetzel, M.: 2006, On the locality of grammatical evolution, *Proceedings of the 9th European Conference on Genetic Programming*, Budapest, Hungary, 10 - 12 Apr. 2006, Volume 3905 of Lecture Notes in Computer Science, 320–330.

Whitley D. and Kauth K.: 1988, GENITOR: A different Genetic Algorithm, *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, Denver, 118–130.