Size Does Not Matter: Evolving Parameters for a Cayley Graph Visualiser using 64 bits

Miguel Nicolau and Dan Costelloe

Natural Computing Research & Applications Group
University College Dublin
Dublin, Ireland
miguel.nicolau@ucd.ie, dan.costelloe@gmail.com

Abstract. In this paper, an Interactive Evolutionary system is described, which generates visually appealing 3D projections of mathematical constructs. This system uses a combination of the Grammatical Evolution paradigm and Jenn3d, a visualiser of Cayley graphs of finite Coxeter groups. A very compact representation is used for the genotype strings, using only 64 bits. The resulting visualisations, albeit somewhat restricted, still exhibit a large degree of complexity and evolvability, and are well representative of the domain.

1 Introduction

One of the central aspects of evolutionary computation is the representation used. Search operators are directly related to it, and the choice of representation directly affects the effectiveness of the search (how fast can optimal solutions be found), and also the navigation of the search space, such as the degree of changes to potential solutions. In systems using mapping processes, the representation is even more crucial, as the transformations of genotype to phenotype and then to fitness can easily lose causality, leading to an essentially exploratory role of the search operators.

In an interactive evolutionary system, with an essentially user-driven evolutionary process, strong causality is of vital importance. When a user selects a given solution, and through a small change this solution is changed into a completely different one, is likely to lead to user frustration, strengthen the sense of randomness in the generative process, and ultimately cause user fatigue.

At the other extreme, a certain degree of exploration is always required, particularly in creative systems: the existence of large neutral landscapes would result in the same phenotypic solution being shown to the user multiple times, and thus also negatively affect the interactive evolutionary process.

This paper explores representational issues for a Grammatical Evolution [10] approach to the visualisation of mathematical projections. It is based on an entry to the "World in a Word" 64-Bit Design Challenge [3] (WIAW), a competition ran in conjunction with the 2013 IEEE Congress on Evolutionary Computation. The objective was to evolve 64-bit strings, which were used to explore creative

domains. Entries were evaluated based on the originality of the domain, the diversity of the structures evolved, the appeal of the domain and the evolved individuals, and the extent to which the 64 bits were used.

There are many potential choices for the domain to explore. The competition suggested a wide range of potential areas, including visual artwork, graphic design, architectural motifs, 3D sculptures, musical styles, poetry, NLP tasks, simulations, cellular automata, games, puzzles, entire game systems, etc. Some of the examples provided include Fourier-based harmonic curves and versions of Conway's Game of Life.

Most of those domains suggested either a direct mapping, or a generative (or developmental) approach to creativity [2], such as L-Systems [6], to create as diverse structures as possible, from the (somewhat restricted) range of 64-bit strings.

The entry described here took a different approach. It used the 64 bits as a basis to create an integer string, which in turn was used with the Grammatical Evolution system [10], to evolve parameters for Jenn3d [9], a visualiser of Cayley graphs of finite Coxeter groups. The mapping process was regulated through a context-free grammar, defining the parameter space to navigate. This entry was the winner of the WIAW competition.

This is an extension of previous work [7], which created some award-winning visualisations. One won the Evolutionary Art competition at the EvoStar 2010 conference [4], and was subsequently chosen to illustrate all the proceedings associated with EvoStar 2011 [13]. Another was submitted to the prestigious *UCD Research Images* competition [1], held annually by UCD, Ireland, and finished in third place, amongst over 100 images; it is now used by the University to showcase its research programme. The use of 64 bits in this work limits the range of visualisations achievable, but still allows for an extensive exploration of the domain. It also highlights the compression potential of mapping systems such as Grammatical Evolution.

This document describes the approach taken. Section 2 gives an overview of the domain explored, and Section 3 describes the methodology used, including an overview of the evolutionary approach, the encoding used, and the fitness evaluation. Section 4 exhibits some examples generated, analyses the genotype strings evolved, and suggests further improvements.

2 Jenn3d

Jenn3d [9] is a visualiser of finite Coxeter groups on 4 elements. These groups can be represented as reflections of Euclidean 4-space, or as reflections of the 3-sphere. It builds Cayley graphs using the Todd-Coxeter algorithm, and visualizes those graphs by embedding them in the 3-sphere, then stereographically projecting the graph from the 3-sphere to Euclidean 3-space, and finally rendering the 3D structure as a 2D picture. Jenn3d renders using OpenGL, and is fast enough to allow the user to rotate and navigate in curved 3D space, and gain an intuition for the geometry of the 3-sphere.

The rich and complex domain of visualisations representable through Jenn3d is defined by the following parameters:

1. the 4×4 Coxeter matrix, as specified by the 6 integers in the upper triangle matrix;
2. a subset of up to 3 of the 4 generators, by which vertices should be fixed (the larger the subset, the fewer vertices in the quotient);
3. a list of group elements to define edges, where each element is written as a string in the generating set; and
4. a list of faces, where each face is written as a pair of generating elements.

Note that this parameter set is partial and redundant in that *(i)* most Coxeter matrices result in an infinite group and the Todd-Coxeter algorithm does not converge; *(ii)* permuting the generators results in the same structure, but with a different initial visualization; *(iii)* the lists of group elements defining edges are really sets, and so are invariant under permutation and duplication; and *(iv)* when defining edges, each group element definition can be written in infinitely many ways modulo group equality, e.g., $r_4 r_4 r_2 r_1 r_3 = r_2 r_1 r_3$, since $r_4 r_4 = 1$ is the identity.

This large and complex parameter space of drawings is very difficult to navigate, particularly for users interested in the visualisations, but unfamiliar with Coxeter groups theory; as such, it is an excellent candidate for exploration using Grammatical Evolution.

3 Methodology

3.1 Evolutionary Approach

To explore the Jenn3d parameter space, Grammatical Evolution (GE) [10] was used. GE typically uses a variable-length Genetic Algorithm (GA) [5] to create binary or decimal strings, which choose productions from a given grammar, and thus generate syntactically correct solutions of the search space.

There are several reasons for using GE, both in the original work [7] and in the current approach. The use of grammars really simplifies the representation of the search space, allowing for an easy to use *typed* version of GP. Also, the separation of genotype and phenotype (seen in the modular view of GE, in Fig. 1) allows the usage of any search engine capable of representing numerical strings. This is crucial for the work presented here, as it means that GE could be used with a standard, fixed-length GA (with its length set to 64 bits).

Fig. 2 shows the grammar used. In it both the required and optional parameters for Jenn3d are specified. The Coxeter matrix parameter is drawn from a fixed set, as overly complex matrices are too computationally demanding, resulting in a non-convergence of the Todd-Coxeter algorithm (and hence an invalid phenotype). However, all other parameters are not only optional, but also variable in size, resulting in a large number of possible phenotype structures arising from the 64-bit encoding.

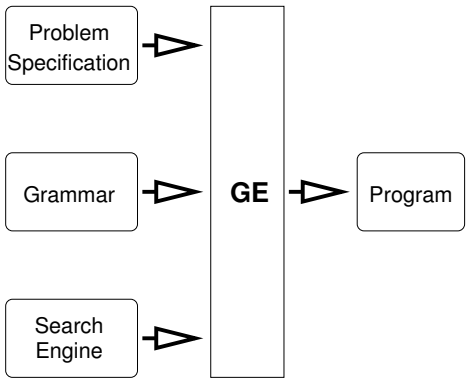


Fig. 1. Modular view of Grammatical Evolution. It combines three independent components: a problem specification (e.g., a fitness function), a grammar (syntactic representation) and a search engine (typically a variable-length genetic algorithm). The evolutionary search is performed at the level of the genotypic strings, while the fitness is evaluated on the resulting phenotype programs.

3.2 Encoding

The first step of the mapping process was to transform the 64-bit strings into integer strings. As each of these integers is used to choose productions associated to a *non-terminal* symbol defined in the grammar, the number of bits required to represent each integer is dependent on that grammar. An analysis of the grammar used (Fig. 2) shows that the symbol with the highest number of associated productions is `<FreePolyhedra>`, with 15 productions; therefore a minimum of 4 bits are required to encode each integer. The 64 bit strings are thus transformed into 16 integer strings.

Note that this introduces biases in the exploration of the phenotype space. For the `<FreePolyhedra>` symbol, for example, all productions have a $1/15$ chance of being chosen, apart from the first production (`<FreePolyhedra>` $\rightarrow$ `3 3 2 2 2 2`), which will have a $2/15$ choice of being selected; this is due to the fact that each integer has a range of $2^4 = [0, 15]$, and GE's usage of the modulus operator to map integers to the choice of productions increases the likelihood of choosing whichever productions are first declared (unless the range of the integers used is an exact multiple of the number of production choices). GE usually deals with these biases by introducing large amounts of redundancy [11], through the use of many bits per integer (typically 32 or even 64, in systems directly manipulating integers).

Table 1 shows the number of productions associated with each grammar non-terminal symbol, and the list of probabilities of these being selected. There are 7 out of 17 symbols that exhibit some form of transformation bias, with various degrees of probability.

```
<cmdline>               ::= ./jenn <GEXOMarker> -c <CoxeterMatrix>
                            <GEXOMarker> <StabilizingGenerators>
                            <GEXOMarker> <Edges>
                            <GEXOMarker> <Faces>
                            <GEXOMarker> <VertexWeights> <GEXOMarker>
<CoxeterMatrix>         ::= <Torus> | <FreePolyhedra> | <FreePolytope>
<Torus>                 ::= <Int_2_12> 2 2 2 2 <Int_2_12>
<FreePolyhedra>         ::= 3 3 2 2 2 2 | 3 4 2 2 2 2 | 3 5 2 2 2 2
                          | 4 3 2 2 2 2 | 5 3 2 2 2 2 | 3 2 3 2 2 2
                          | 3 2 4 2 2 2 | 3 2 5 2 2 2 | 4 2 3 2 2 2
                          | 5 2 3 2 2 2 | 2 2 2 2 3 3 | 2 2 2 2 3 4
                          | 2 2 2 2 3 5 | 2 2 2 2 4 3 | 2 2 2 2 5 3
<FreePolytope>          ::= 3 3 3 2 2 2 | 3 3 2 2 3 2 | 3 3 2 2 4 2
                          | 3 3 2 2 5 2 | 3 4 2 2 3 2 | 3 5 2 2 3 2
                          | 4 3 2 2 3 2 | 3 2 3 2 2 3 | 3 2 3 2 2 4
                          | 3 2 3 2 2 5 | 3 2 4 2 2 3 | 4 2 3 2 2 3
                          | 5 2 3 2 2 3 | 2 2 3 2 3 3
<StabilizingGenerators> ::= | -v <Comb0123>
<Edges>                 ::= | -e <EdgeSet>
<EdgeSet>               ::= <Comb0123> | <EdgeSet> <Comb0123>
<Faces>                 ::= | -f <FaceSet>
<FaceSet>               ::= <FourInt0_3> | <FaceSet> <FourInt0_3>
<FourInt0_3>            ::= <Int0_3>
                          | <Int0_3><Int0_3>
                          | <Int0_3><Int0_3><Int0_3>
                          | <Int0_3><Int0_3><Int0_3><Int0_3>
<VertexWeights>         ::= | -w <Int1_12> <Int1_12> <Int1_12> <Int1_12>
<Int0_3>                ::= 0 | 1 | 2 | 3
<Int1_12>               ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
                          | 10 | 11 | 12
<Int2_12>               ::= 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
                          | 10 | 11 | 12
<Int3_12>               ::= 3 | 4 | 5 | 6 | 7 | 8 | 9
                          | 10 | 11 | 12
<Comb0123>              ::= 0 | 1 | 2 | 3
                          | 01 | 02 | 03 | 12 | 13 | 23
                          | 012 | 013 | 023 | 123 | 0123
```

Fig. 2. Grammar used to navigate the Jenn3d parameter space.

Also note that not all 2^{64} possible binary strings generate unique individuals. Some do not generate valid phenotypes at all, as the mapping process might not terminate; others generate the same choices in the grammar, and hence the same phenotypes; and finally some combinations of the leftmost bits will generate shorter mappings, which will not use some of the rightmost bits.

Even so, the number of possible combinations is still very large, and the functionality of each bit can vary, depending on the production choices of the

Table 1. Probabilities of non-terminal symbol mapping transformations, using 4 bits to encode each choice

Nonterminal Symbol	Choices	Mapping Transformation Probabilities
`<cmdline>`	1	100%
`<CoxeterMatrix>`	3	37.5%, 31.25%, 31.25%
`<Torus>`	1	100%
`<FreePolyhedra>`	15	12.5%, 6.25% .. 6.25%
`<FreePolytope>`	14	12.5%, 12.5%, 6.25% .. 6.25%
`<StabilizingGenerators>`	2	50%, 50%
`<Edges>`	2	50%, 50%
`<EdgeSet>`	2	50%, 50%
`<Faces>`	2	50%, 50%
`<FaceSet>`	2	50%, 50%
`<FourInt0_3>`	4	25%, 25%, 25%, 25%
`<VertexWeights>`	2	50%, 50%
`<Int0_3>`	4	25%, 25%, 25%, 25%
`<Int1_12>`	12	12.5%, 12.5%, 12.5%, 12.5%, 6.25% .. 6.25%
`<Int2_12>`	11	12.5%, 12.5%, 12.5%, 12.5%, 12.5%, 6.25% .. 6.25%
`<Int3_12>`	10	12.5% .. 12.5%, 6.25%, 6.25%, 6.25%, 6.25%
`<Comb0123>`	15	12.5%, 6.25% .. 6.25%

bits preceding it. Furthermore, from a visualisation point of view, the search space is potentially infinite, as the user can select from a multitude of rotation and zoom combinations for each visualisation.

Efficient Exploitation A custom GA is used, along with extensions in GE, to introduce crossover markers through the grammar [8]. This technique is especially useful when domain knowledge allows the apriori identification of blocks of information: special non-terminal symbols are introduced in the grammar, and the GA is then limited to these crossover points. It has been used in the past to separate well-defined behaviours for video controllers [12], and also in the previous application of GE to Jenn3d [7]; previous work has even allowed these crossover markers to evolve their position [8].

The crossover point markers used in this study can be seen in the grammar (Fig. 2), in the `<cmdline>` non-terminal symbol declaration. A special `<GEXOMarker>` symbol was inserted between each parameter set; these allowed those sets to be exchanged between individuals, thus increasing the visual relationship between parents and offspring.

Note that the Java framework provided for the competition [3] also uses a Monte-Carlo (MC) algorithm [14] as a search procedure. This however is incompatible with the representation used, because deep explorations typical of MC search (i.e. explorations of the rightmost bits in the bit string) can potentially always generate the same visualisation; this is particularly true for bit strings

where the mapping process does not use the whole genotype string. As such, the MC algorithm was not used.

3.3 Fitness Evaluation

As the objective of the system is to evolve attractive and personalised visualisations, it runs in an interactive manner. Each correctly generated individual is exposed to the user, to receive a fitness score. This allows the individual to directly interact with the 3D-visualisation, have a better understanding of the generated structure, and achieve his/her preferred projection, before assigning a fitness score.

Fig. 3 shows the Jenn3D interface, extended so that a scoring process is present. Ideally, the evolutionary process proceeds in an endless manner; every time a fitness score is attributed to a structure, a new one is presented immediately after. If the user instead chooses to *exit* the application, the evolutionary process terminates. The full range of exploration tools in Jenn3d is available for each presented structure; this includes options to save the evolved parameters, and/or export a high-resolution image of the current visualisation.

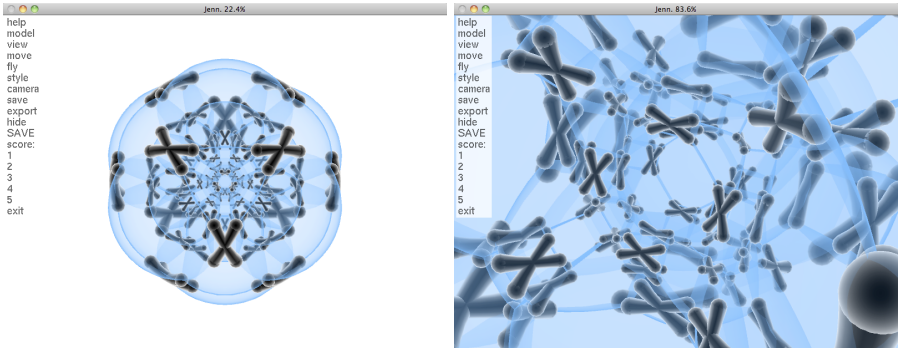


Fig. 3. The Jenn3d interface, along with the extensions encoded. An example structure is shown in the left; the same structure, when zoomed in and rotated (right), can produce a dramatically different visualisation.

Fitness and Typicality Alongside a fitness score (in the range $[0.0, 5.0]$), the competition also required a *typicality* value in the range $[0.0, 1.0]$, indicating how well an individual represents the domain (typically a functional measure that captures some key structural properties). In the case of Jenn3d, it is not intuitively obvious how to set this score, as structural analysis of the generated visualisations is not possible, and hence the range of typicality values was limited.

Table 2 shows the range of possible fitness and typicality scores. Due to the complexity of the Todd-Coxeter algorithm, some visualisations are impossible

to generate, causing the Jenn3d software to crash[1]; as these might be close to correct (and potentially attractive) visualisations, fitness and typicality scores of 0.5 are automatically attributed.

Table 2. Fitness and Typicality scores, and events generating those scores; fitness values 1-5 are user supplied

Event	Fitness	Typicality
Unsuccessful GE mapping	0.0	0
Non-convergence of Tedd-Coxeter algorithm	0.5	0.5
Rejected visualisation	1.0	1.0
Poor visualisation score	2.0	1.0
Average visualisation score	3.0	1.0
High visualisation score	4.0	1.0
Visualisation remains in population unchanged	5.0	1.0

The user can control the evolutionary process through the fitness score. A fitness score of 1.0, for example, guarantees that the current visualisation is replaced by a random one in the next generation, whereas a score of 5.0 guarantees that the visualisation is passed unchanged to the next generation, thus potentially being used as a seed for alternative, similar visualisations (as per the scoring process proposed previously [7]).

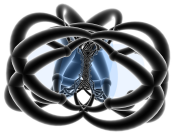
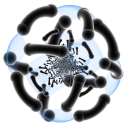
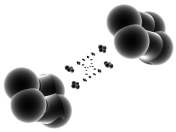
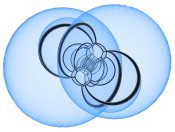
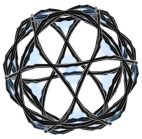
4 Results

Table 3 shows example projections, achieved through evolution, along with their hexadecimal, integer and mapped strings. The specific visualisations were chosen to the taste of the authors. Higher resolution images are shown in Fig. 4.

These examples also show the variety with which the bit strings are used. The 2^{nd}, 3^{rd} and 4^{th} solutions, for example, came from the same evolutionary run, and their similar parameter structure still exhibits traces of the genetic material exchanged through marked crossover (and subsequently altered by mutation); the resulting projections are however substantially different. The 7^{th} solution (`0xdb400f50dc50f13a`), on the other hand, illustrates how some solutions can be very small, by using only 40 bits to encode its solution (i.e. it has a tail of 24 unused bits).

[1] This can be frustrating for the user, and a workaround in the Jenn3d software is beyond the scope of this study (and the mathematical knowledge of the authors); a smart way to deal with this problem (in a Mac environment) is to leave the crash report window open, as only a single such window can be open at any given time.

Table 3. Parameters evolved for the examples shown in Fig. 4

	Hex: 0x9134db20eb2d6f07 Genotype: 9 1 3 4 13 11 2 0 14 11 2 13 6 15 0 7 Phenotype: jenn -c 3 2 2 2 2 5 -e 0 0123 -f 23
	Hex: 0x53090ffede18f1b8 Genotype: 5 3 0 9 0 15 15 14 13 14 1 8 15 1 11 8 Phenotype: jenn -c 3 3 2 2 5 2 -e 0 -f 21
	Hex: 0xc3298ff8d8003028 Genotype: 12 3 2 9 8 15 15 8 13 8 0 0 3 0 2 8 Phenotype: jenn -c 5 2 2 2 2 4 -v 13 -e 123 13
	Hex: 0xc3090fe89c10f098 Genotype: 12 3 0 9 0 15 14 8 9 12 1 0 15 0 9 8 Phenotype: jenn -c 5 2 2 2 2 2 -v 0 -e 13 -f 03
	Hex: 0xdb410f50dc59f1ba Genotype: 13 11 4 1 0 15 5 0 13 12 5 9 15 1 11 10 Phenotype: jenn -c 2 2 2 2 3 4 -e 0 -f 01 -w 4 2 12 11
	Hex: 0xd1410f70dc59f0ac Genotype: 13 1 4 1 0 15 7 0 13 12 5 9 15 0 10 12 Phenotype: jenn -c 3 4 2 2 2 2 -e 0 -f 01 -w 4 1 11 1
	Hex: 0xdb400f50dc50f13a Genotype: 13 11 4 0 0 15 5 0 13 12 5 0 15 1 3 10 Phenotype: jenn -c 2 2 2 2 3 4 -w 6 1 2 1
	Hex: 0xd2410ef0d859f4be Genotype: 13 2 4 1 0 14 15 0 13 8 5 9 15 4 11 14 Phenotype: jenn -c 3 5 2 2 2 2 -e 0123 -f 01 -w 4 5 12 3

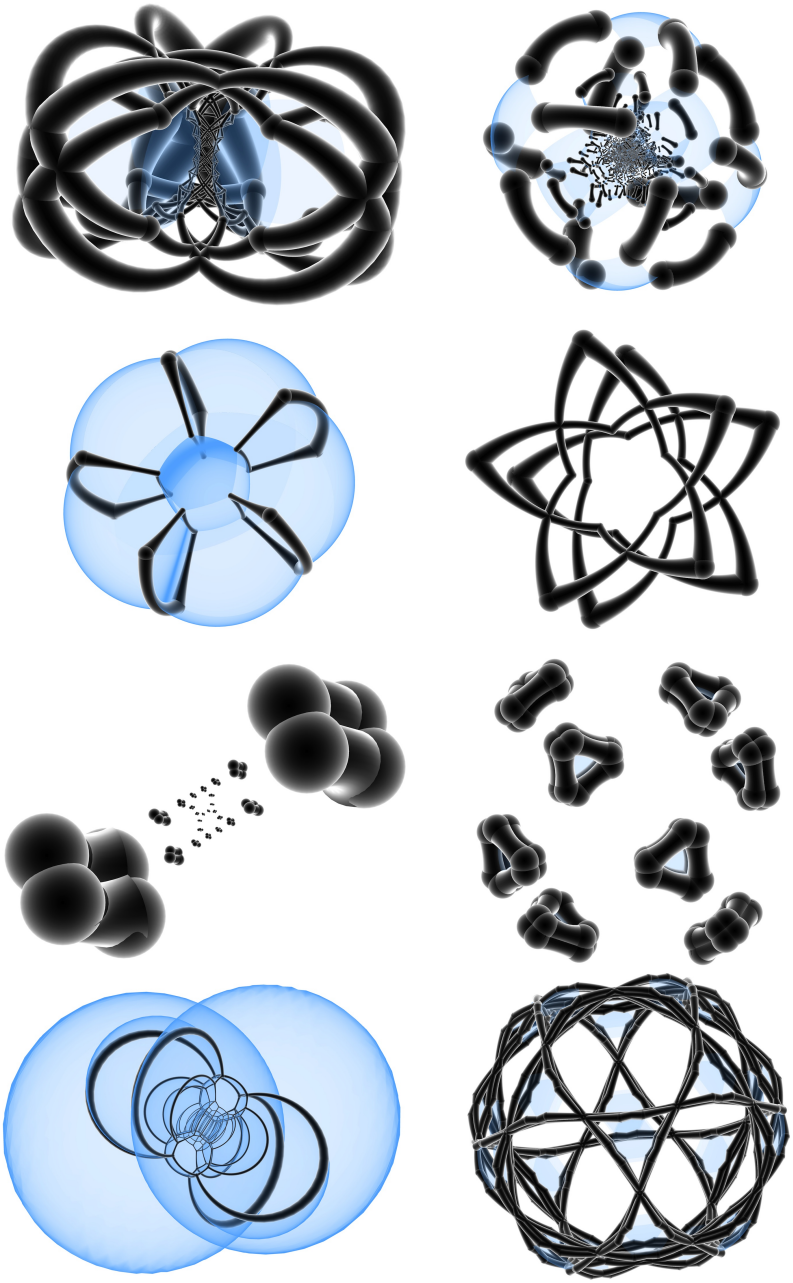


Fig. 4. Mapping examples.

4.1 More Efficient Encoding

Although the encoding used allows a vast exploration of the domain, there is still room for improvement. A variable number of bits per integer can be used, as the number of grammar choices for each non-terminal symbol is known during the mapping process: for example, the `<CoxeterMatrix>` symbol has only 3 choices, requiring only 2 bits, and symbols such as `<StabilizingGenerators>` require only one bit to encode a choice. This would decrease the average number of bits per integer, which in turn would increase the number of integers generated (and thus the size of the landscape explored); this is essentially a version of Arithmetic Encoding [15], adapted to the specific mapping process employed by GE.

The downside of this very compact representation would be the impossibility of using an integer-based genetic algorithm, as mutations in early integers in the string would affect the functionality of integers further down, meaning that the boundaries of integer definitions (in the binary string) are only known during the mapping process (thus breaking somewhat the modular aspect of GE, as shown in Fig. 1).

5 Conclusions

This paper presented a study of a compact representation for the Grammatical Evolution system, used to evolve parameters for the Jenn3d visualiser. The resulting system was the winning entry of the 2013 "World in a Word" 64-Bit Design Challenge competition. The ability of GE to use very compact binary strings was used; this is achievable by reducing the number of bits required to encode each grammar choice (at the cost of some biases). The resulting Jenn3d projections are varied and well representative of the domain (check the Jenn3d website [9] for a gallery of projections).

There are many possible future work avenues from this point. One potential route concerns the use of crossover markers in the grammar. It would be interesting to analyse/quantify to what extent this technique aids the generation of visually pleasing, thematically consistent candidate solutions – without too much disruption of the user's evolutionary flow towards interesting areas of the search space. Since we are operating in an interactive context, this would be particularly useful as fighting user-fatigue remains an ongoing battle.

This quantification can even be explored without the need to rely on human input. For example, by examining the complexity of images undergoing Grammatical Evolution both with and without the use of crossover markers via non-subjective computational techniques.

References

1. UCD Research Images. http://www.ucd.ie/research/images/2011winners/, February 2012.
2. Margaret A. Boden and Ernest A. Edmonds. What is generative art? *Digital Creativity*, 20(1-2):21–46, 2009.

3. Cameron Browne. World in a word 64-bit design challenge. http://www.cameronius.com/research/cec/, June 2013.
4. Anna I. Esparcia-Alcázar, Anikó Ekárt, Sara Silva, Stephen Dignum, and A. Sima Uyar, editors. *Genetic Programming, 13th European Conference, EuroGP 2010, Istanbul, Turkey, April 7-9, 2010, Proceedings*, volume 6021 of *LNCS*. EvoStar, Springer, 2010.
5. John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
6. Aristid Lindenmayer. Mathematical models for cellular interaction in development, parts i and ii. *Journal of Theoretical Biology*, 18:280–315, 1968.
7. Miguel Nicolau and Dan Costelloe. Using grammatical evolution to parameterise interactive 3d image generation. In Cecilia Di Chio et al., editor, *Applications of Evolutionary Computation - EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part II*, volume 6625 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 2011.
8. Miguel Nicolau and Ian Dempsey. Introducing grammar based extensions for grammatical evolution. In *IEEE Congress on Evolutionary Computation, CEC 2006, Vancouver, BC, Canada, July 16-21, 2006, Proceedings*, pages 2663–2670. IEEE Press, 2006.
9. Fritz Obermeyer. Jenn3d for visualizing coxeter polytopes. http://jenn3d.org, June 2010.
10. Michael O'Neill and Conor Ryan. *Grammatical Evolution - Evolutionary Automatic Programming in an Arbitrary Language*, volume 4 of *Genetic Programming*. Kluwer Academic, 2003.
11. Michael O'Neill, Conor Ryan, and Miguel Nicolau. Grammar defined introns: An investigation into grammars, introns, and bias in grammatical evolution. In Lee Spector et al., editor, *Genetic and Evolutionary Computation - GECCO 2001, Genetic and Evolutionary Computation Conference, San Francisco, CA, USA, July 7-11, 2001, Proceedings*, pages 97–103. Morgan Kaufmann, 2001.
12. Diego Perez, Miguel Nicolau, Michael O'Neill, and Anthony Brabazon. Evolving behaviour trees for the mario ai competition using grammatical evolution. In Cecilia Di Chio et al., editor, *Applications of Evolutionary Computation - EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part I*, volume 6624 of *Lecture Notes in Computer Science*, pages 123–132. Springer, 2011.
13. Sara Silva, James A. Foster, Miguel Nicolau, Mario Giacobini, and Penousal Machado, editors. *Genetic Programming, 14th European Conference, EuroGP 2011, Torino, Italy, April 27-29, 2011, Proceedings*, volume 6621 of *LNCS*. EvoStar, Springer, 2011.
14. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
15. Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.