Evolving Parametric Models using Genetic Programming with Artificial Selection

John Harding
^1 University of The West of England
^1 john3.harding@uwe.ac.uk

Evolutionary methods with artificial selection have been shown to be an effective human-computer technique for exploring design spaces with unknown goals. This paper investigates an interactive evolution of visual programs currently used in popular parametric modelling software. Although parametric models provide a useful cognitive artifact for designers to interact with, they are often bound by their topological structure with the designer left to adjusting (or optimising) metric variables as part of a design search. By allowing the topological structure of the graph to be evolved as well as the parameters, artificial selection can be employed to explore a wider design space more suited to the early design stage.

**Keywords:** genetic programming, parametric design, artificial selection, evolutionary design, design exploration

INTRODUCTION

Parametric modelling software for constructing visual programs is now commonplace in architectural design. Based on dataflow programming, visual programs define the development of form through a series of associated explicit functions, commonly taking the form of a Directed Acyclic Graph (DAG). The structure of the DAG describes a mapping of numbers to geometry, setting out a possible design space to be explored when parameters are adjusted (Aish and Woodbury 2005).

A combination of parametric modelling and performance analysis tools allow designs to be evaluated both quantitatively and qualitatively in real-time when adjusting parameters. This process is now well-known, with multiple 3rd party analysis plug-ins (environmental, structural, etc.) being developed for visual programming tools such as Rhino Grasshopper and Autodesk Dynamo. The recent increase in such applications shows a desire to allow modelling and analysis to take place in the same integrated environment.

**Cognitive Artifact**

A 'parametric schema' defines the development of form explicitly as well as providing a cognitive artifact for the design team to interact with. Parametric modelling therefore shifts focus on final form to the development of form. As Oxman states (2006, 243): "In digital design significant processes that have frequently been represented as non-explicit in traditional design models must now be considered explicit".

This differentiates explicit programs that have a defined hierarchy with implicit ones, for example those found in emergent systems. Whilst the latter can offer novel design approaches and wide design exploration (Bentley and Kumar 1999), the explicit developmental structure between 'seed' and 'design'
is often lost. Parametric models therefore provide a valuable cognitive artifact for design teams working at the level of process.

**Parametric lock-in**

Although a DAG-based parametric model keeps a record of how building geometry is created, displaying this explicitly comes at a price. As Aish and Woodbury (2005, 11) state: "nothing can be created in a parametric system for which a designer has not explicitly externalised... this runs counter to the often-deliberate cultivation of ambiguity that appears to be part of the healthy design process." As the DAG becomes increasingly complicated, so its flexibility reduces. The graph can quickly resemble a tangle of spaghetti, making it increasingly inflexible and unsuitable for the early stages of design where flexibility is most desired (Davis 2013).

Although theoretically an infinite amount of designs can be generated in a parametric system (Oxman and Gu 2015), they are bounded by the topological structure of the DAG. To date, evolutionary methods in parametric design have focused on adjusting numeric parameters only, for example when parametric models are combined with metaheuristics such as Genetic Algorithms (GA) or Simulated Annealing (SA) (Rutten 2013).

This paper therefore investigates whether a wider parametric design exploration using evolutionary methods can take place by allowing the topological structure of the DAG to evolve in addition to the metric parameters? This would help counter parametric lock-in, opening up design possibilities at the conceptual stage by working at a higher level of abstraction (Harding et al. 2012).

**GENETIC PROGRAMMING FOR DAGS**

DeLanda (2002, 11) states that if architectural designs are to enjoy the same degree of combinatorial productivity as biological ones, we must begin to "think topologically" and avoid too narrow a design search. By applying Genetic Programming (GP) techniques on DAG-based parametric models, this can potentially be achieved - essentially turning parametric design software into a combinatorial shape grammar with a vast selection of common geometric rules already at hand.

Shape grammars in architectural design have seen a resurgence in recent years (Grasl and Economou 2014), both in terms of alternative methods of representation using graphs (Grasl and Economou 2011) and in supporting designers who are still in the process of designing and may not yet have a clear shape grammar in mind (Strobbe et al. 2015). Whereas these examples tend to use bespoke software applications, the motivation for this work lies in working with existing visual programming methods such as Rhino Grasshopper.

In architectural design, GP has been explored in the generation of form by Frazer (1995) and later Coates et al. (1999) using GP with Lindenmayer systems. More recently evolving networks through augmenting topologies (NEAT) (Stanley and Miikkulainen 2002) have been employed for the exploration of novel forms (Clune and Lipson 2011).

A recent form of GP known as Cartesian Genetic Programming (CGP) has focussed specifically on evolving DAGs (Miller and Harding 2008). The attraction of CGP and its applicability in a wide range of fields is partly due to its relatively simple developmental encoding from an integer string to combinatorial structures. This form of encoding has inspired the method presented here that reduces a Grasshopper parametric model to a part-integer, part-floating point genotype, with the associated phenotype a parametric model.

**Embryo**

Embryo is the name of a Rhino Grasshopper plug-in developed by the author that maps a number string genotype to a complete parametric model phenotype. The choice of development within Grasshopper was partly due to its current popularity. As Vierlinger and Bollinger state (2014, 609): "Grasshopper unveils algorithmic design to non-programmers with intuitive interfaces". Its organisation of common (and
bespoke) explicit functions into 'components' make them similar to shape grammar rules, and the potential for combining both human and computer generated DAGS and analysis tools under one platform make it an attractive choice.

Constructing a graph in Grasshopper can be split into three categories: External parameters (for example numeric sliders, external geometry, etc.), the pool of components in the graph (nodes) and the topological structure that forms associations between components (edges). These three categories form the basis of the genotype used by Embryo when constructing a parametric model:

1. Metric genes: control the parameter values for generated sliders and have a direct numerical mapping. These can be either integer or floating point values. These metric parameters are the first things generated by Embryo.
2. Function genes: when a component is added to the graph, the function genes controls the type of component is selected from the pool.
3. The topological genes are integer based and map the output location for each component input when forming the graph. Altering these genes changes the topology of the graph.

If enough genes are not provided then genes are repeated. Furthermore, if the value of genes are too high for the current state of the system then modular arithmetic is used. Using a large gene pool negates this but requires more memory.

Figure 1 gives an outline of the process with a simple example. A component pool is specified (a), which in addition to the genotype is used by the Embryo component (b) to generate a parametric model (c). The metric genes are mapped directly to the numeric sliders. The functional (shown blue) and topological (shown red) aspects of the genotype indicate how each component is chosen and its inputs are then connected back to an output respectfully. A form of parameter datatype matching ensures wires are connected to relevant inputs and outputs.

Further details on the method used can be found in Harding (2014) which includes the current limitations with the approach. This include aspects such as handling multiple associations per component input which is yet to be included. Aspects such as dealing with parameter datatype matching and more settings such as removing unsuccessful components is also discussed but will not be elaborated on here.

Clearly the combinatorial possibilities with such a process is huge, hence a brute force approach for exploring the design space meaningfully is difficult and other metaheuristics should be considered that help counter bloat, a known problem when using GP (Miller 2001). The explicit embryogeny of the DAG encoding makes it suitable for evolutionary methods such as genetic algorithms (Kumar and Bentley 2000), where evolvability is also a key concern.

The CGP encoding method means that during mutation and crossover, aspects of the original definition are maintained. As the process is effectively evolving a visual program itself, this essentially translates to a form of genetic programming.
EVOLVING PARAMETRIC MODELS

Evolutionary algorithms commonly use an objective function to evaluate designs at each generation. Artificial selection however replaces this function with selection by human participant(s) who need not make their motivations explicit and may change during the evolutionary process (Dawkins 1986). Such Interactive Evolutionary Computation (IEC) methods are capable of exploring complexity without requiring human understanding of the specific process involved (Sims 1991, 328).

Leaving the objective function open to the design team allows for qualitative or intangible drivers to be included in the design search. That is not to say that quantitative performance criteria cannot be included, only that the human participant(s) becomes an active part of the design search, guiding the evolutionary method through the paradox of choice at the early design stage (Piasecki and Hanna 2011).

Judgement during evolution therefore takes place on two forms of phenotypic representation, both the parametric schema (development model) and the building model (generated design). The design team can (if they choose) steer the process to evolve parametric models that are legible, or capable of being understood. Maintaining this engagement gives the evolved models a chance to be further developed manually following an evolutionary process.

Grasshopper Setup

Grasshopper proved to be a suitable environment for combining Embryo, a metaheuristic algorithm and an artificial selection method. By explicitly formulating the search algorithm in the same environment as the problem is defined, a more flexible reaction to special requirements is possible (Vierlinger and Bollinger 2014, 611). This meant that several third party components could be used to construct an interactive evolutionary approach.

Octopus Explicit (Vierlinger and Bollinger 2014) is a set of tools for forming customised evolutionary algorithms. In this implementation the genotype format (Octopus solution), components for crossover (Simulated Binary Crossover) and mutation (point) are all utilised. Hoopsnake by Yannis Chatzikonstantinou [1] facilitates feedback loops within Grasshopper. This is activated following selection, allowing the population to evolve and pause for user input at each generation.

Figure 2 shows the overall process with associated special components highlighted in red. An initial population of genotypes and associated parametric model phenotypes are randomly generated. These can be viewed by the designer (both in terms of the DAG and generated geometry) and two elite parents selected for the next generation. Crossover and mutation is then applied to form the next generation.

The associated implementation in Grasshopper is given in Figure 3. Below the horizontal threshold line is the metaheuristic process as per the previous diagram. This includes the manipulation of the population of genotypes and their use by the Embryo component (shown bottom right). The parametric models generated by the machine are located above the threshold line. The components used in the generated graph are selected from a pool, shown top-left. These could be any Grasshopper components, either...
Figure 3
The setup in Grasshopper. The model above the threshold line is machine-generated.

Figure 4
Three bespoke Grasshopper components for the Tower Hamlets project.

standard ones or those created by third parties. Note that in this example, some human placed components have also been incorporated during graph generation, something that Embryo can handle with outputs being tagged (*). This is one of the main benefits of using a single environment, as designers can begin to work at different levels of abstraction (i.e. crossing the threshold).

Project Application
In collaboration with 3DReid Architects (London), Embryo was originally used to generate parametric models for a residential project in Tower Hamlets (Harding 2014). Shape grammar rules were embedded as Grasshopper components (Figure 4) with heuristics both specific for this project and based on the experience of the architect.
This included two types of residential massing block of set width and variable length: one with a set minimum height (a) and one with an additional variable controlled by the 'add storeys' input (b). Planes for connecting the massing blocks were set as input and output parameters, allowing different but constrained permutations to be explored. An additional third component formed a link bridge between residential blocks to improve connectivity and contrast the orthogonality of the block layout (c).

The original process involved mostly using a random seed to generate designs, but only small alterations could then be made manually. Essentially, the process whilst useful became a task of selecting a particular generated model as opposed to developing one iteratively using cumulative selection.

Figure 5
Six generations of development. Gross internal area is shown for each design.
In light of this, the original project was therefore revisited, adapted into the evolutionary method as discussed in the previous section and a new trial conducted. The generated parametric models therefore lie somewhere in-between a human and machine approach, with the process becoming more of a conversation with the machine able to suggest alternatives but with neither party dictating development.

**Results**

In the trial, an initial random population of 8 designs was presented to the design team. The parametric models generated began with 6 numeric parameters and 12 components. A crossover and mutation rate of 0.7 and 0.1 respectively was used. During the search, quantitative analysis could be conducted again within Grasshopper. In this simple example, usable floor area (as a function of volume) and a measure of heat loss (surface area / volume) is offered to the design team that can potentially influence the selection process.

Designs with around 40,000m$^2$ GIA based on the number of residential units were deemed to be desirable. The massing context could be easily referenced in during the design search due again to the single Rhino/Grasshopper environment.

The results of a typical run is shown in Figure 5 with each design having an associated parametric model phenotype (Figure 6). Each design is labelled A-H for each generation. In this example, only six generations are shown but the process could in theory continue indefinitely. Two parent designs are selected from each generation as shown highlighted.

Some of the motivations behind the selection at each generation were recorded during the process:

1. First choice simply based on potential for future development.
2. Option D chosen due to the introduction of a

![Figure 6](image-url)

Associated parametric model topology at each generation.
block in the x-axis. Option B chosen due to interesting array of link bridges
3. Option D again for the same reasons. Option (F3) with non-working components not selected. Parameter domains reduced and component number increased to 16. Option G selected due to possible courtyard space at centre.
4. Options D&E appear similar with slightly different parameters. D selected. GIA needs reducing to brief requirement influencing choice of H.
5. Heat losses getting quite high hence Option H selected. Elevated elements providing interest. Ramp on A a possible entrance to a centrally located building away from existing massing. Courtyard created between new buildings and existing.
6. Option F now has ramps from one location to two blocks. Heat loss performance has recovered. GIA is a little low but further adjustments can be made.

The motivations during the search were both qualitative and quantitative, and constantly subject to change. Once the final design was selected, a parametric definition exists (as opposed to a CAD model) could be taken forward for further development. Figure 7 shows how the model was developed in terms of parameter adjustment (a), alterations to the graph (b) and enhancing the massing model (c). The GIA could therefore be increased to the brief requirement parametrically.

Clearly these are simply massing models and some of the choices made are based on relatively simple criteria, but the potential for including graph development as part of the search has potential if the legibility of the graph can remain. The clean structure of machine generated parametric models can help with this. In reality there is a balance to be struck between having too few metric parameters to understand their influence and too many to keep track of. As Davis (2013, 76) states: "An ideal parametric model would encompass all the variations the designer wants to explore within the smallest dimensionality possible."

CONCLUSIONS
This research shows how a combination of parametric modelling tools with genetic programming can explore wider design spaces in architectural design whilst retaining a parametric definition. By opening up the topology of the graph as part of the search, parametric models are generated as opposed to adjustment of numeric parameters by the metaheurisic. Using an interactive evolutionary algorithm, human-computer interaction can be used to enhance the search even when the problem cannot be defined.

The example given has shown potential, but has revealed a number of limitations that need addressing in future work:

- At present only one parametric model can be viewed at a time (Grasshopper is not designed for this type of use). Further interface development is therefore required to display mul-
Multiple models in each generation simultaneously.

- All generated numeric parameters have the same range giving no hierarchy of scale appropriate to the components they are associated to.
- As discussed in Section 3, multiple edges per component input is yet to be implemented.
- It was found that crossover caused a larger than expected disruption to the parent graphs. This was likely due to the CGP method of encoding and using mutation alone may be more suitable approach in future in line with other CGP applications (Clegg et al. 2007).
- Bloat can still be an issue and generated parametric models can sometimes freeze. Incorporating a maximum calculation time and removing any model that does not compute from the gene pool is a possible solution.
- Facilitating groups of components in the pool (i.e. clusters).
- User fatigue is a known issue with interactive evolutionary computation. Kruse and Conor (2015) have shown how intelligent agents can be used in conjunction with an IGA.

As well as addressing these limitations, future related research includes using the technique to evolve parametric models for existing CAD geometry in order to suggest new design directions for models that are inflexible. Here a combination of target based evolution is more suitable, perhaps using other meta-heuristics such as simulated annealing.

ACKNOWLEDGEMENTS

The author would like to thank 3D Reid Architects for their support during this research. This work is part-sponsored by the 2016/17 UWE VC Early Career Researcher Development Award.

REFERENCES

Aish, R and Woodbury, R 2005 'Multi-level interaction in parametric design', *Smart Graphics*, pp. 151-162
Bentley, P and Kumar, S 1999 'Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem', *Proceedings of the genetic and evolutionary computation conference*, pp. 35-43
Grasl, T and Economou, A 2011 'GRAPE: Using graph grammars to implement shape grammars', *Proceedings of the 2011 symposium on simulation for architecture and urban design*, pp. 21-28

Kumar, S and Bentley, PJ 2000 'Implicit evolvability: an investigation into the evolvability of an embryogeny', *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, Las Vegas, Nevada, USA, pp. 198-204


Miller, J and Harding, S 2008 'Cartesian genetic programming', *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pp. 2701-2726


Vierlinger, R and Bollinger, K 2014 'Accommodating change in parametric design', *Proceedings ofACADIA*, pp. 609-618