NAVIGATING BETWEEN GENERATIVE SYSTEMS

An educational framework and an experiment with the qGrowth grammar

PEDRO VELOSO[1] and RAMESH KRISHNAMURTI[2]
[1,2]*Carnegie Mellon University*
[1,2]*{pveloso|ramesh}@andrew.cmu.edu*

Abstract. In this paper, we delineate an educational framework for the development of new generative systems. It comprehends eight schemas: unstructured constructive, structured constructive, variational, improvement, discrete simulation, continuous simulation, generative learning and behavioral learning. To demonstrate the application of the framework, we introduce the qGrowth grammar, an educational generative system that mimics biological growth using simple quadrilateral shapes and three rules: expansion, division and elimination. We describe the original formulation of qGrowth in the unstructured constructive schema, then we customize it with techniques from structured constructive schema. Particularly, we combine rule sampling and ordering of the frontier of target quadrilaterals by different data structures, such as a queue, stack or heap. It results in a structured qGrowth with easy control of the growth pattern, limiting its design space to meaningful regions. In the end, we discuss other potential extensions of the qGrowth using the different schemas of the framework.

Keywords. Generative systems; Classification; Constructive schema; Rule-based modeling; Shape Grammar.

1. Introduction

Generative systems (GS) is one of the main topics of intelligent and informed exploitation in computational design and have been studied for more than five decades. Pioneering research in GS incorporated problem-solving techniques from Artificial Intelligence to automate the synthesis and allocation of architectural spaces - this is known as space planning. The repertoire of generative procedures comprises a diverse set of computational techniques such as optimization, agent-based models, physics-based simulation, parametric modeling and neural networks.

From a theoretical perspective, the field of computational design would benefit from a systematic and historical understanding of the different computational paradigms available for GS. The historical component emphasizes the existing knowledge base in computational design research. In the same way that an algorithm course in computer science might refer to canonical algorithms as a

base for new developments in the field, canonical generative data-structures and algorithms provide a common background to support research on GS.

In this paper we introduce a general framework for GS based on a variety of computational schemas. This framework was developed for a course we teach at our institution and has resulted in interesting work and insightful feedback from the students. The framework is intended to support development of custom GS with an expressive design space and meaningful navigation. We introduce an exemplar GS used in the course, which combines multiple paradigms depicted in our framework. The objective is to show how the framework can be used to investigate the appropriate computational methods for the problems at hand and, potentially, become a base for novel GS for design.

2. Generative Systems

There is a vast published literature on GS in conferences and courses; see, for example (Celani and Veloso 2015; Mitchell 1977; Mitchell, Liggett, and Kvan 1987; Coates and Thum 1995; Coates 2010; Terzidis 2006). However, only in few publications are they defined and categorized. A brief review is given below.

Alexander (1968) is one of the earliest references for GS. He described two categories of systems: generating system and system as a whole. The former is a kit of parts, with rules that define how these parts may be combined, producing the latter, which consists of interacting objects, such as buildings and people. Alexander's description supported his later work on Pattern Language (1977), where each pattern is a semi-autonomous set of generative design rules to deal with specific problems in the different scales of the environment. Those rules associate a context, the interaction of its subsystem of forces and the spatial configuration that ensures the balance between those forces.

Mitchell (1977) used Aristotle's description of a theoretical system that produces a variety of animals by the combination of different kinds of organs as the foundation of GS. Then, he proposed three categories: analogue GS are composed of analogue elements that enable mechanical operations to change the state of the system; iconic GS are systems that use movies, models, drawings, and geometric operations to generate solutions, symbolic GS use symbols and computational data-structures to represent a solution and rely on arithmetic and logical operations to change it.

Each symbolic GS operates with discrete steps: the space of related designs that are visited in the computational process are codified in a directed graph called state-action graph, which structures the space of all possible designs and available operations for navigation (Mitchell 1977, pp.46-48). The goal of GS is to navigate the set of solutions in this state-action graph − which we refer to as the design space − looking for a subset of solutions that satisfy the design goals. Therefore, GS are divided in three technical components: (1) a fixed representation of spaces and locations to host the activities of the building, (2) an explicit criterion to evaluate the candidates in the design space, (3) a solution procedure that generates the candidates. Overall, Mitchell's definition of GS comprehends computational techniques that can produce a variety of potential space planning solutions that

meets certain specified criteria automatically.

Mitchell and other researchers, such as Henrion (1978), produced classifications of GS of the time, focusing on space-planning. Their classification comprehends shape grammars and other methods such as search, optimization, hybrid and analytical procedures. Ligget (2000) updated their categories with the use of metaheuristics, such as Genetic Algorithms and Simulated Annealing.

Fischer and Herr define GS as a "(...) set-up based on abstract definitions of possible design variations capable of displaying or producing design products (or elements of design products)" (2001, p.3). With the adoption of generative techniques from a variety of sources in design, they moved away from the classical AI techniques of space planning and suggested four broader pedagogical categories: (1) Emergent systems, self-organization (e.g. cellular automata and swarm modelling); (2) Generative grammars (e.g. L-systems and shape-grammars); (3) Algorithmic generation and growth (e.g. fractals, re-writing rules, parametric design, data mapping); (4) Algorithmic (re-) production (e.g. genetic algorithms, selective procedures).

Recently, with the dissemination of parametric editors integrated with geometric modelers, GS have been widely used in the development of complex forms. In this context, Oxman and Oxman (2014) proposed high-level categories that, instead of emphasizing the computational logic of GS, focused on the inspiration or the source domain of the generative ideas used in computational design. More specifically, they provide six categories for form generation procedures: (1) mathematical: which exploits mathematical formulae for generative procedures; (2) tectonic, which employs tectonic patterns for form generation; (3) material: which uses tectonic and assembly patterns, such as folding, braiding, knitting and weaving, to generate form; (4) natural or neo-biological, which employs biological principles to generate form; (5) fabricational, which uses existing patterns of fabrication for design generation; and (6) performative, which models physical data of the context as the input for a generative process that satisfy certain objectives.

3. A Framework for Generative Systems

Design space and navigation are crucial features both for the traditional and computational approaches to design. In the former, designers use heuristics to reformulate the problem, building expressive design spaces and navigational strategies to explore solution candidates (Rowe 1987). In a computational setting, design space and navigation are a consequence of the choice of representation, analysis and solution procedures embedded in algorithms and models for GS. Given its potential to mediate traditional and computational design, our framework for GS uses design space and navigation to define eight schemas (see table 1).

The first six schemas comprehend techniques traditionally used for the development of GS. The 7th and 8th schemas employ Machine Learning – a multidisciplinary field concerned with programs that automatically improve with experience (Mitchell 1997) – to learn a design space or navigation from data, behavior or knowledge. For example, generative models in Machine Learning

model the distribution of input and output data of a task, so they can synthesize new data by sampling (Bishop 2006, p.43). In reinforcement learning, agents can learn what actions to take to maximize a numerical reward in an environment by trial and error (Sutton and Barto 2018). Additionally, all the schemas in the framework can be combined to form hybrid GS.

Table 1. Computational schemas for generative systems .

Schema	Design space	Design navigation	Examples
1) Unstructured constructive	Possible transformations of the initial state by set of rules	Application of rules or procedures	Fractals, L-systems, mesh subdivision, tessellations and shape grammars
2) Structured constructive	Possible transformations of the initial state by set of rules structured by graph or tree	Search, traversal or sampling by application of rules or procedures	partition trees, uninformed search, informed search, random search, colorings and dissections
3) Variational	Parameter space with properties and relations between entities	Navigation in parameter space	Early algorithmic art (8-corner, Turbulence centered and Schotter) and parametric models (grid attractor, population of surfaces, etc)
4) Improvement	Parameter space with properties and relations between entities with corresponding fitness value.	Navigation in parameter space, guided by fitness landscape	generate-and-test, hill climbing, gradient descent, simulated annealing, particle swarm optimization and evolutionary algorithms
5) Discrete simulation	Possible states of a composite representation, starting from an initial configuration	Local rules applied on the components of the representation over time	cellular automata, reaction-diffusion, diffusion-limited aggregation, urban simulations, slime mold and ant foraging
6) Continuous simulation	States resulting from interaction between agents and continuous environment	Local rules followed by each agent in the environment	rigid/soft body simulation, differential growth and swarms
7) Generative learning	Parameter space and generative functions learned by data distribution	Navigation in the learned parameter space	Autoencoder, GAN and PCA
8) Behavioral learning	Policy or value of states learned by experience	Set of possible actions to be learned associated with the states	dynamic programming, Monte-Carlo, Temporal-difference, Q-Learning and DQN
Hybrid	combine previous design spaces	combine previous design navigations	Shape Annealing (1 + 4), Shape GAN (1 + 7), optimization of a partition tree (2 + 4) and learning steering behaviors (6 + 8)

4. qGrowth

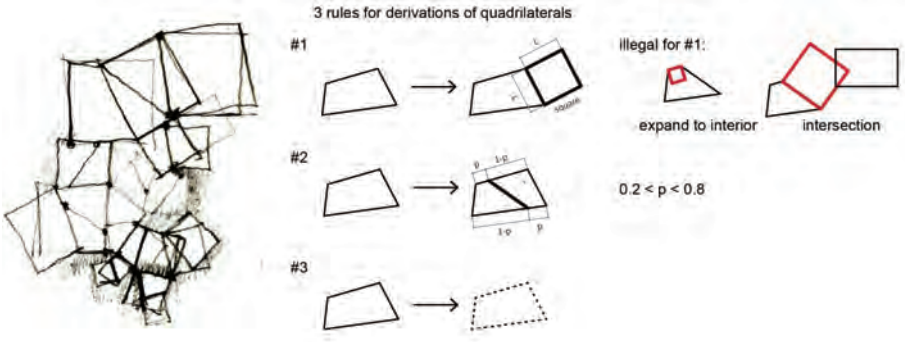

Figure 1. Left: sketch with the biological motivation of the qGrowth. Right: three rules for derivation: (1) expansion, (2) parametric division and (3) elimination.

The qGrowth is a dimensionless shape grammar to mimic a biological growth process using simple quadrilateral shapes (see figure 1). The growth starts with a single square of any size and grows by applying successive rules to the current shapes, which will potentially expand it in multiple directions or even separate it into multiple clusters of quadrilaterals. The canonical version of the grammar consists of three base rules: (1) expansion of a new square, (2) parametric division, and (3) elimination. In rule 1, it is not allowed to generate a square that is inside or intersects with an existing quadrilateral.

qGrowth is both customizable within its original schema or by transitioning to different schemas. For customization, qGrowth can be generalized to a three-dimensional setting with polyhedra. New rules can generate new forms of divisions, expansions or even re-connections, while new constraints can ensure the satisfaction of minimum and maximum angles, areas and edge lengths in the new quadrilaterals. In the sequel we present an extension of the qGrowth grammar in the structured constructive schema.

4.1. A FULLY STRUCTURED QGROWTH

To translate the qGrowth grammar from an unstructured to a structured constructive schema, we build its search tree and use algorithms to look for good paths and states (Russel and Norvig 2010, pp.64-102). Each node of its search tree is a set of quadrilaterals, resulting in a combinatorial state space that is potentially infinite. The outgoing edges are instances of its rules (expansion, division and elimination) applied to a target quadrilateral in a given state. Following the resulting structure of the tree, a search algorithm will organize a frontier and explored set, to look for good paths and states (adapted from Russel and Norvig 2010, p.77):

```
function GRAPH—SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal then return the solution
      add the node to the explored set
      expand the chosen node,
          add resulting nodes to the frontier
          only if not in the frontier or explored set
```

In one extreme, it is possible to use deterministic transitions to fully structure qGrowth. In this case, the transitions should combine the three rules, the potential target quadrilaterals in a state, a discretization of the parameters of rule 2, sides of the quadrilateral selected for division, and consider the constraints of rule 1 (see figure 1). As the number of quadrilaterals in the state increases, there are more potential collisions pruning some of the branches, but, still, the potential derivations grow with the number of quadrilaterals in a state. This combination results in a variable and large branching factor (see figure 2). This is a proper formulation for a systematic search over the state space of the qGrowth. However, as the frontier is too large, it is computationally expensive for standard search

algorithms. The discretization of the division parameters also eliminates large portions of the state space, which might contain interesting solutions. Even more critical is the joint formulation of the actions leading to a large and variable branching factor, which makes it difficult to parameterize and control the expression of the resulting organism.

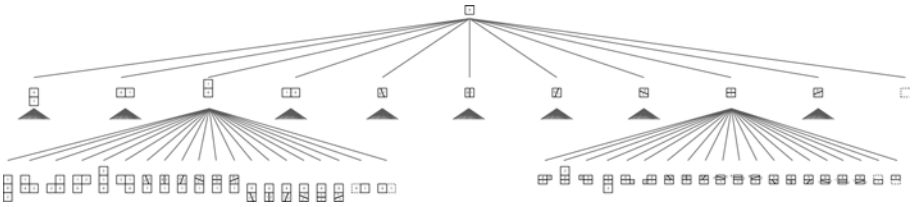

Figure 2. qGrowth with fixed branches and p in {1/4, 2/4, 3/4}. Despite the constraints and the discretization of p, the number of states grows fast. In our implementation, we counted the following numbers of states per level: 1 (root), 11, 200, 5380, 186552, ...

4.2. A PARTIALLY STRUCTURED QGROWTH

Considering that the original motivation of the qGrowth is to express certain biological patterns of growth, we can formulate a custom tree for a generative process, simplifying the transitions and avoiding simple discretizations.

We divided the transitions between states in two parts: the rules are selected by sampling, while the matching quadrilaterals are ordered and selected by deterministic datastructures and operations. This changes the formulation of our state, which is now composed of two distinct sets, one set with the quadrilaterals already expanded and another with the child quadrilaterals that can still be expanded, divided or eliminated. For simplicity, the quadrilaterals can only be expanded from the frontier to the explored set once but are never moved back to the frontier. We also customized the rules of qGrowth to ensure that their application to any given quadrilateral will result in a single branch, fixing the maximum branching factor to three (which can still be pruned by collision detection):

- In rule 1, the quadrilateral from the frontier is added to the explored set and all the four possible squares added to the frontier. The collision test is applied between each of the four squares and the explored set.
- In rule 2, the quadrilateral from the frontier is divided in two, which stay in the frontier. The parameter p is sampled from an uniform distribution in the interval $[0.2, 0.8]$.
- In rule 3, the quadrilateral is eliminated from the frontier.

In this partially structured implementation, each edge is sampled according to a probability distribution defined by the user (p_{rules}), while the target quadrilaterals are ordered in the frontier according to a strategy (to be specified in the next section). Therefore, it combines a random search for the rules with a deterministic organization of the frontier. Below is the resulting iterative algorithm for a single sampling.

```
function GROWTH(initial_quad, rules, p_rules, max_i)
  initialize the frontier using the initial_quad
  initialize the explored set as an empty set
  loop do
    if the frontier is {} or reached max_i then return explored
    choose current quadrilateral q from the frontier
    if q does not collide with the explored set then
      sample rule from distribution p_rules
      if rule is 1
        then add q to the explored set,
        derive the new quadrilaterals applying rule 1 to q,
        if the new quadrilaterals satisfy constraints
          then add them to the frontier
      if rule is 2
        then sample a parameter p in [.2, .8],
        sample a side s of q,
        derive two quadrilaterals with rule 2, q, s and p,
        if the new quadrilaterals satisfy constraints
          then add them to the frontier
```

With this two-step transition, the ordering strategy generally used for search can be adapted to control the frontier of the quadrilaterals. In this sense, this formulation conciliates the stochasticity of the biological growth with a control over the expression of the resulting organism.

4.3. DIFFERENT STRATEGIES AND EXPRESSIONS

In this section we present examples of strategies applied to the frontier and the resulting growth patterns. We adopted the following settings:

- an initial square of edge size 34
- p_{rules} is [rule 1: 60%, rule 2: 30%, rule 3: 10%]
- a length constraint that only accepts new quadrilaterals if their longest edge length is larger than 15
- the side for division is sampled by a biased coin parameterized by length - i.e. longer sides have more probability of being divided
- two static rectangles represent obstacles in the environment

Using the logic of breadth-first search (BFS), the frontier of quadrilaterals is organized as a queue, so the first element in the frontier is selected first for expansion (see figure 3). The resulting growth pattern is an irregular flood, susceptible to accidents of collisions, divisions and eliminations.

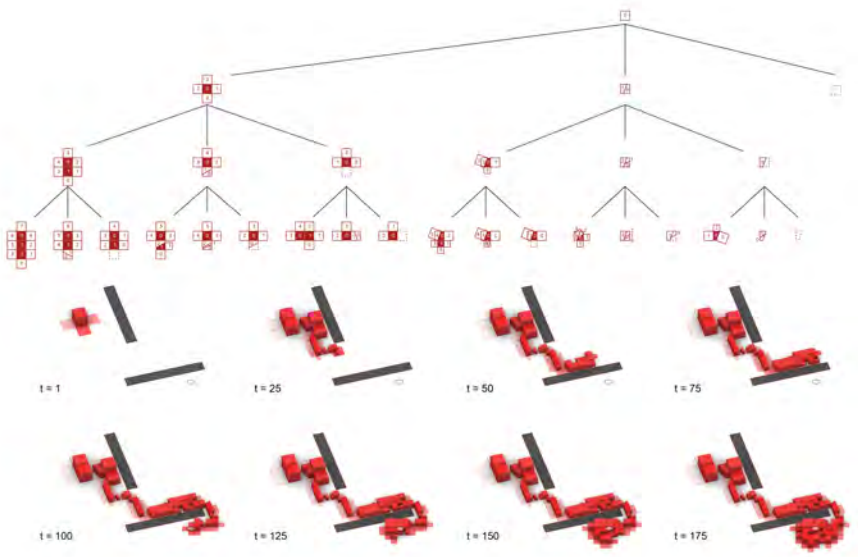

Figure 3. qGrowth using a queue to order the frontier. Top: tree. Bottom: growth.

Using the logic of depth-first search (DFS), the frontier is organized as a stack, so the last element in the frontier is selected first for expansion (see figure 4). The resulting growth pattern is a folding branch.

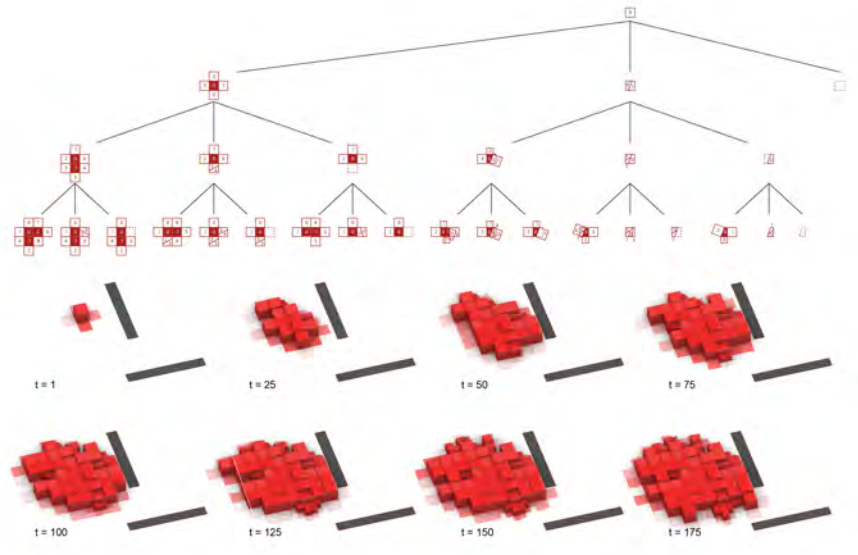

Figure 4. qGrowth using a stack to order the frontier. Top: tree. Bottom: growth.

The quadrilaterals of the frontier can also be stored in a heap and ordered according to a function, such as in Uniform-Cost, Greedy and A* search, which contains information from the growth or from the environment (see figure 5).

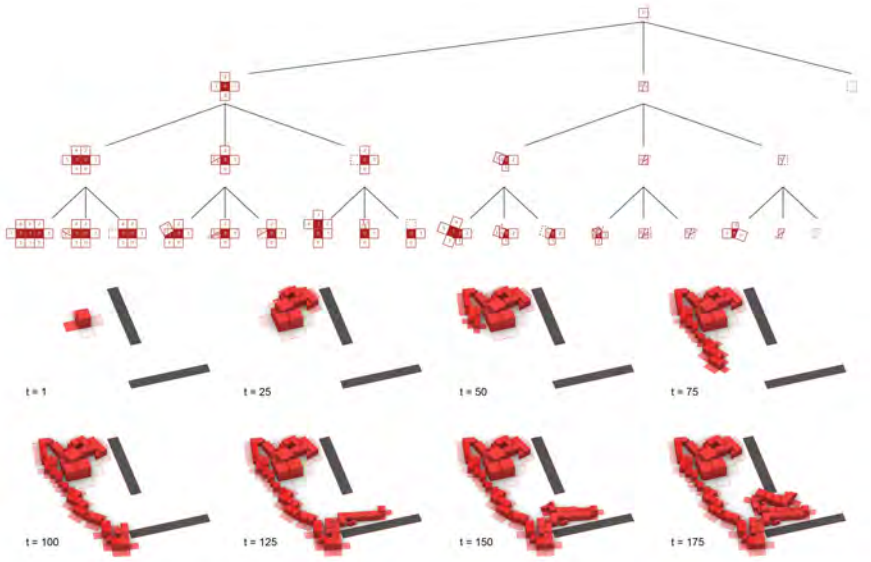

Figure 5. qGrowth using a heap to order the frontier. Top: tree. Bottom: growth. The target point for the function is indicated by an ellipse.

In our implementation, we defined the Euclidean distance from the center of the quadrilateral to a target point in the environment as our heuristic function $h(s)$. The organism will choose quadrilaterals in the frontier that are closer to the target, moving towards the target and, after reaching it, spreading as a flood fill.

5. Discussion

In the paper, we navigated between two traditional schemas – unstructured and structured constructive schemas – to create new versions of qGrowth with custom navigation and design space. This example reinforces the idea that knowledge accumulated in decades of research in computational design, AI and other related fields can become an operative knowledge base. Other schemas could also be used to extend the qGrowth grammar. For example:

1. An improvement procedure can be used to generate a state with n quadrilaterals. In this case, an optimization could be used to learn the best way to order the frontier and select actions that satisfies a certain pattern of growth or occupation.
2. Other alternative to learn this behavior is on-line reinforcement learning, where many trajectories or segments of the trajectories could be sampled to learn a policy (the probability of choosing each action), given a state. For a discrete approach, a Monte-Carlo search can be used. Alternatively, a function approximation such as

a neural network could be used to generalize the learning to unknown states.
3. A GAN can generalize the qGrowth rules to a generator able to produce images with a certain growth pattern.

More than being a template for new computational design courses, our framework provides a common language to access content spread across a diversity of sources. It is an initial resource for investigating the appropriate computational methods for the problems at hand and can become a base for future developments in design methods. Even in courses without a strong programming background, high-level investigation about design spaces and navigations can lead to the adoption of general computational strategies. We hope that this endeavor can expand the boundaries of design within the discipline of architecture and beyond.

Acknowledgements

We would like to express our gratitude to the Brazilian National Council for Scientific and Technological Development (CNPq) for granting Pedro Veloso a PhD scholarship (grant 201374/2014-5).

References

Alexander, C.: 1968, Systems Generating Systems, *Architectural Design*, **38**(12), 605-610.
Bishop, C.M.: 2006, *Pattern Recognition and Machine Learning*, Springer-Verlag, New York.
Celani, G. and Veloso, P.: 2015, CAAD Conferences: A Brief History, *Proceedings of CAAD Futures 2015*, Sao Paulo, 47-58.
Coates, P.: 2010, *Programming Architecture*, Routledge, London.
Coates, P. and Thum, R.: 1995, *Generative Modelling*, University of East London, London.
Fischer, T. and Herr, C.M.: 2001, Teaching Generative Design, *In Proceedings of the 4th Conference on Generative Art*, Milan.
Henrion, M.: 1978, Automatic Space-Planning: A Postmortem?, *Proceedings of Artificial Intelligence and Pattern Recognition in Computer Aided Design*, Amsterdam, 175-196.
Liggett, R.S.: 2000, Automated Facilities Layout: Past, Present and Future, *Automation in Construction*, **9**(2), 197-215.
Mitchell, W.J.: 1977, *Computer-Aided Architectural Design*, Mason Charter Pub, New York.
Mitchell, T.M.: 1997, *Machine Learning*, McGraw-Hill, Boston.
Mitchell, W.J., Liggett, R.S. and Kvan, T.: 1987, *The Art of Computer Graphics Programming: A Structured Introduction for Architects and Designers*, Van Nostrand Reinhold Company, New York.
Oxman, R. and Oxman, R. 2014, From Composition to Generation, *in* R. Oxman and R. Oxman (eds.), *Theories of the Digital in Architecture*, Routledge, New York, 55-61.
Rowe, P.: 1987, *Design Thinking*, MIT Press, Cambridge.
Russel, S.J. and Norvig, P.: 2010, *Artificial Intelligence: A Modern Approach*, Prentice Hall, New Jersey.
Sutton, R.S. and Barto, A.G.: 2018, *Reinforcement Learning: An Introduction*, The MIT Press, Cambridge.
Terzidis, K.: 2006, *Algorithmic Architecture*, The Architectural Press, Oxford.