The conception of parameters and constraint-based models in architectural projects: The case of Lelé's pivot domes

Fernando Ferraz Ribeiro[a], Davidson Martins Moreira[a], Arivaldo Leão de Amorim[b]

[a]*Programa de Modelagem Computacional, SENAI Cimatec, Av. Orlando Gomes 1845, 41.650-010 Salvador, Bahia, Brazil*

[b]*Laboratório de Computação Gráfica Aplicada à Arquitetura e ao Desenho LCAD, Faculdade de Arquitetura, Universidade Federal da Bahia (UFBA), Salvador, BA, Brasil*

Abstract

Following increasing interest among architects in the application of generative algorithms in the conception of projects, many questions have been raised concerning teaching this particular manner of thinking about forms. One of the main goals of learning the subject is to understand how the idea of the construction of an element can be translated into the rules of an algorithm. Other aims are to introduce the requisite programming-related concepts to implement the algorithms, and to locate this method within the scope of disciplines involved in the construction activity. This paper proposes an example-based strategy to discuss these three crucial issues by using the pivot domes designed by Brazilian architect Lelé as the object of study.

Keywords: Parametric Models, Geometric Constraints, Generative Algorithms.

1. Introduction

The concepts, techniques, and applications of generative algorithms have offered significant gains in the last few decades, and have thus garnered considerable interest from architects, researchers, and teachers [1]. Many leading global *bureaus* in the field of architecture have invested in implementing and maintaining programming departments to collaborate in the creative process of building design [2]. The range of influence of computer-aided design (CAD) extrapolates the aspect of an applied tool and reaches aesthetic theories [3] [4] and, through integration with computer-aided manufacturing (CAM) systems, construction activities [5]. Generative algorithms have contributed to this process.

In the academic world, the assimilation of the methods, tools, and knowledge involved in the understanding of generative design paradigms presents a range of possibilities and opportunities to rethink relations among disciplines involved in the design activities [5] [6].

Email addresses: ffribeiro@gmail.com (Fernando Ferraz Ribeiro), davidson.moreira@gmail.com (Davidson Martins Moreira), alamorim@ufba.br (Arivaldo Leão de Amorim)

One of the main issues in teaching generative design to novice users is to make them understand how an initial idea can be translated into rules that generate forms. Understanding how these forms can be generated is the first step in this different approach to project activities encouraged by such algorithms. It is important to remember that this generation of forms in not an objective in itself, but merely a tool to assist in answering questions raised by architecture as an interdisciplinary domain. In that sense, the building of effective generative algorithms also relies on understanding the characteristics of the building or construction element that may need to be explored and tested by the process, and the manner in which different parameters act together in a way that the variety of forms created by them establish a set of possible solutions, each one with particular characteristics, to be disposed of or adopted by the user.

This paper proposes an example-based strategy to introduce generative algorithms to students through the definition of the rules that governs a conceptual parametric model of an architectural element based on parameters and constraints. The geometrical constructions used, the concepts concerning list manipulation-based programming, and aesthetics aspects of the generated forms are presented and discussed from an interdisciplinary perspective.

Although aimed at architecture and construction, the contents explored in this article can be easily adopted by oder fields, such as mechanical engineering, robotics, product design, among others. Whenever a demand exists for the conception of a form to be built, and the variations of this form could generate gains or losses in some performances or characteristics of the final product, the parametric design strategies could collaborate in obtaining better results within the same period.

2. Generative design and generative algorithms

Algorithms are defined as a finite set of rules that describe steps to solve a specific problem [7]. The technical drawings applied in the design of architectural plans are, indeed, algorithms [8, chap. 3], and the sequential operations with compass, rule, and straight edges that are used to make geometrical constructions describe the relevant rules. Words such as "perpendicular," "parallel," and "tangent" summarize entire algorithms in the drafter's vocabulary. The development of CAD technology, began by providing the ability to reproduce drawing algorithms through software, and have since evolved to a "wider graphic vocabulary available to designers, together with a more elaborate syntax—in all, a richer and potentially more expressive graphic and spatial language" [9, chap. 15].

Generative design can broadly be defined as the application of algorithms or a "rule-based process through which various potential design solutions can be created" [10]. Generative algorithms are the core engine propelling this design strategy in the computational context. As stated above, all technical drawings in constructions are algorithms of a sort. The variety of buildings designed using these methods is innumerable. Generative design techniques differ from traditional methods for transforming the methodology of work. Generative systems lend a degree of automaticity to the process, hence enabling the user to evaluate a greater number of possible solutions to the architectural problem in a shorter amount of time than would be possible through traditional methods.

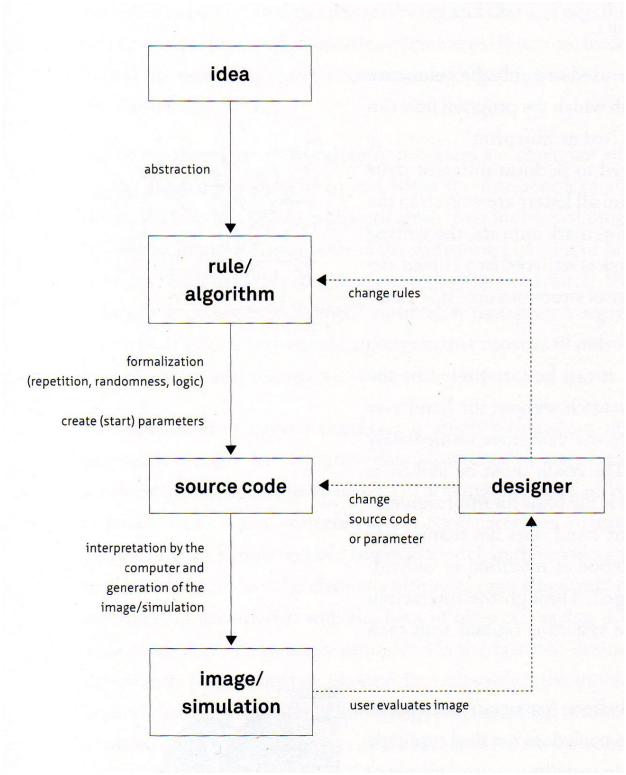


Figure 1: Flowchart of a generative algorithm

In Fig. 1 [11], a flowchart illustrates the proceedings for creating and operating a generative algorithm. The real shift promoted by generative algorithms is in the process of electing the form to be constructed. It starts with an idea of what the constructed object should be, as an abstraction that is translated into a set of rules. An algorithm is implemented in a programming language, code, or a visual programmable ambiance. A simulation of the model is generated by setting parameters values and executing the algorithm. The designer evaluates the output and, by changing the parameters values, the rules, and/or the code, generates different outputs and chooses the final solution among these. The construable artifact appears only when the designer elects the proposal to be built [12].

Many computational methods have been applied for the creation of forms driven by generative algorithms: shape grammars [13] [14] [15] [16], genetic algorithms [17] [10] [18] [19], cellular automata [20], simulated annealing [21] [22] [23], tabu search [24], swarm intelligence [25] [26], and parametric models [27] [28] [29] [30], among others.

3. Parameter- and constraint-based models (PCM)

All generative algorithms, as well as all computer processing, are based on parameters, although their parametric design differs from those of others due to its emphasis on the manipulation of parameters [12]. To draw a right prism in a CAD system, the user needs

to enter values for width, height, and depth. In a parametric design system, each of these inputs can be interactively modified, and the dimensions of the prism are automatically updated in the drawing.

Among the many simulation methods to implement a generative design system, the Constraint Solver, since its introduction by Pro/Engineer in the 1980s, was adopted by most CAD systems [31]. It has garnered considerable research interest due to its intuitive and fluid manipulation of forms. The method relies on the satisfaction of constraint problems that, given a finite set of variables, consist of the assignment of these to a finite domain such that the values taken together are restricted by certain constraints, and finding a combination of these values that satisfies all constraints [32]. The definition of parameter- and constraint-based models proposed here is a way of emphasizing both the parameters and the constraints that define a generative design process.

The creation and manipulation of geometric forms in an ambiance that enables the use of a constraint solver is closely related to proof theory for geometric theorems [31]. Geometry has been a cornerstone of architecture and construction throughout history [2], and the teaching of generative algorithms should explore geometrical knowledge from an alternative point of view.

4. Object of study

The Brazilian architect João Filgueiras Lima, better known by his nickname Lelé, is one of the most important and celebrated architects of the country [33]. He is considered by Oscar Niemeyer to be a "great master of architecture," and by Lucio Costa as "the architect in whom art and technology meet and merge" [34]. His works reveal a strong interest in rational use of materials, an outstanding development of prefabrication and environmental integration, and adopting innovative daylight and air circulation solutions [35] [36].

Figure 2: Sarah Kubitschek Hospital auditorium, Rio de Janeiro - RJ - www.sarah.br

As an object of study, we used the pivoting domes in the auditorium of the Sarah Kubitschek Hospital in Rio de Janeiro (Fig. 2), and in two buildings of the Labor Justice

Figure 3: Labor Justice Courthouse Complex, Salvador - Ba. Source: Instituto Habitat archives

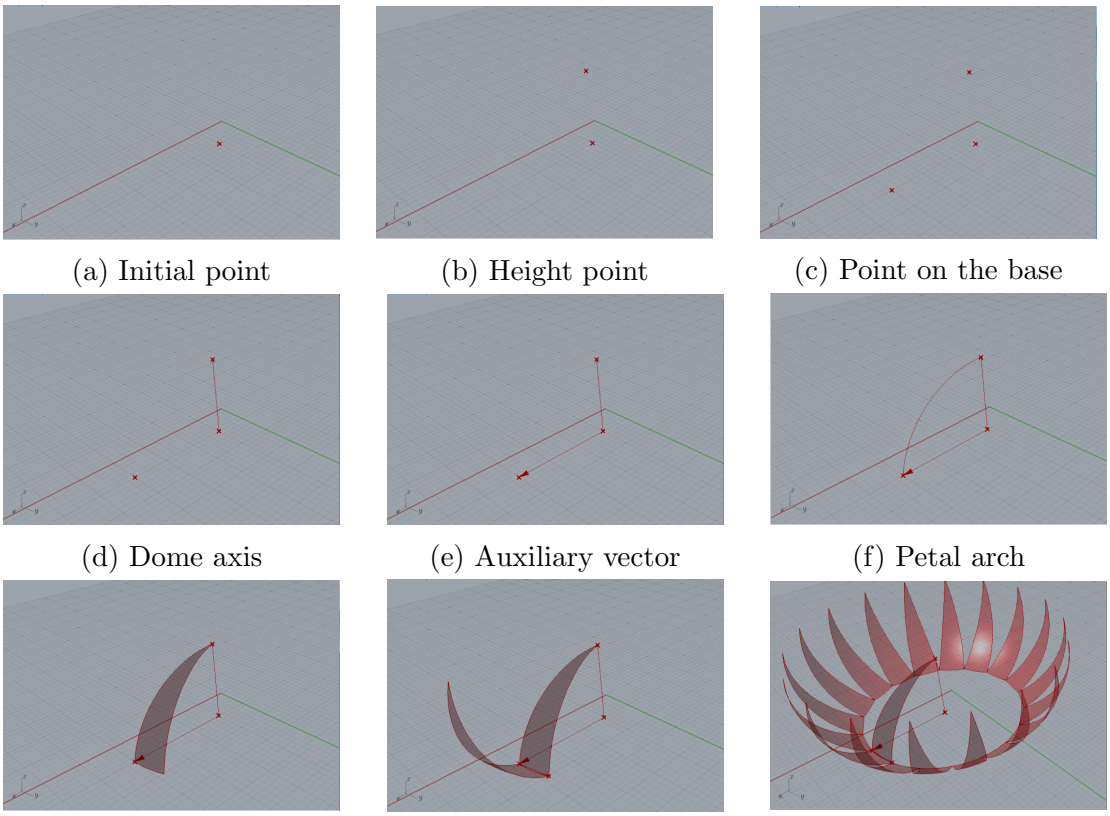
Courthouse complex in the city of Salvador. Figure 3 shows a 3D model of the complex, with the dome of the auditorium to the right and that of the main court to the left.

The first box in the flowchart in Figure 1 represents the idea of the family of forms to be generated. There are didactic advantages to presenting image references to illustrate the idea underlying the generative algorithms to be built. Instead of describing the abstract idea of a goal, the architectural examples clarify important aspects of the intended solution. The application of these elements to real projects, the aesthetic composition that emerged from the varying forms of the domes and their relation with the of which buildings they are part, and the fact that many aspects of the algorithm can be visually identified in the processes, can be mentioned as exemplars. A discussion of a one-sentence definition of the underlying idea could be conducted in class, or one such sentence could be proposed by the teacher to motivate discussion. The summarized idea should point students' attention to the goals of the algorithm being studied. "A dome, divided into petals that pivot around axes on their base" is a good example of a definition of the idea.

5. Defining the parameters

Starting from the idea, the pivoting domes shown in Figures 2 and 3 should be translated into algorithms that can automatically draw variations of these forms. Analyzing the domes in Figures 2 and 3, the similarities and differences between the chosen solutions can be listed:

- All domes are, when closed, spherical caps.
- There are varying numbers of petals in the examples.
- The base radii and heights of the spherical caps in the examples are different.
- The bases of the petals are at the bases of the spherical caps.
- The tops of the petals are at the maximum height of the sphere.



(a) Initial point (b) Height point (c) Point on the base

(d) Dome axis (e) Auxiliary vector (f) Petal arch

(g) Petal surface (h) Rotated petal (i) Full dome

Figure 4: Steps of the one-petal algorithm

Based on this visual observations, many elements of the algorithm can be defined. The parameters, for instance, can be the base radius, the center of the base, the maximum height, and the number of petals. Another parameter should be added to the algorithm to simulate the pivoting movement of the dome: a rotation angle parameter.

It is important to note that other parameters can replace the selected ones to draw the same forms. The radius of the full sphere and the differences between the cap height, the center of the sphere, and the sphere radius, for example, can replace the maximum height, the base center, and the cap base radius generating the same geometries. Although equivalent results can be accomplished by the proposed strategies, it is good practice to think about the characteristics of the construction element that are intended to be explored, and how the parameters to be manipulated fit into the process. Let us assume that in the example, the main characteristics to be explored are the area of illumination and ventilation provided by the dome when open, and the aesthetic relationship between the radius and the maximum height. This assumption justifies the proposed parameters, as they directly affect the characteristics under consideration.

6. Geometric constructions and constraints

In the environment of a geometric constraint solver, the definition of the rules (constraints) can only be conceived by planning the geometric constructions. The first modeling strategy presented in this paper is to model one petal, define a rotation axis (associated with the rotation angle parameter), and copy it around a circle. Figure 4 shows the steps of the algorithm, which are discussed below.

Starting from the **center** (Fig. 4a) of the base circle, we make two copies of the point: one along the z-axis, by the distance entered as the cap height parameter, called the **height point** (Fig. 4b), and the other along the x-axis, taking the value of the base radius as the module of the translation vector, referred to as a **point on the base** (Fig. 4c). A line between the **base center** and the **height point** is also drawn, and is called the **dome axis** (Fig. 4d).

The next step is drawing the arc on one side of the petal. As the arcs are drawn independently, the rules that define them should constrain the closed dome as a spherical cap. One of the ways to effectively constrain this construction is to make the arc be a tangent, at the height point, to a line or vector parallel to a line drawn from the base center to the point on the base. Figure 5, illustrates this idea by showing that only the blue arcs, which are tangent to the green vectors at the height point, together form a circle arch.

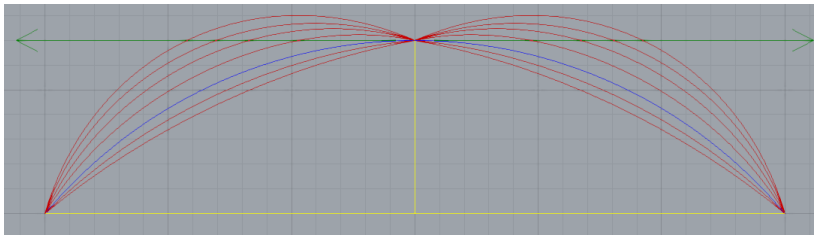


Figure 5: Constraining the arcs as part of a spherical cap

In this step of the algorithm, an opportunity to debate some aspects of the geometric constructions applied in the process should be noted. More precisely, we can reflect on how this arc is defined in classical geometrical drawings, and how this construction is translated into the computational environment.

The traditional steps to draw a tangent arc are shown in Figure 6. The construction is based on two properties of circular arcs. One is that the tangent at every point is perpendicular to the radius at the same point. The other is the most basic property of a circle—that every point on it is equidistant to the center.

By starting with the two red points and the tangent vector, represented by the green line in Figure 6, and drawing a line perpendicular to the vector at the point where the vector is tangent to the arc, it is safe to say that this perpendicular line (yellow) intersects the center of the sphere. The next step is to find a point on the perpendicular line equidistant to the two points of the arc. This can be done by drawing a segment that connects the two points (cyan), and a perpendicular line through the middle (magenta). The point of intersection

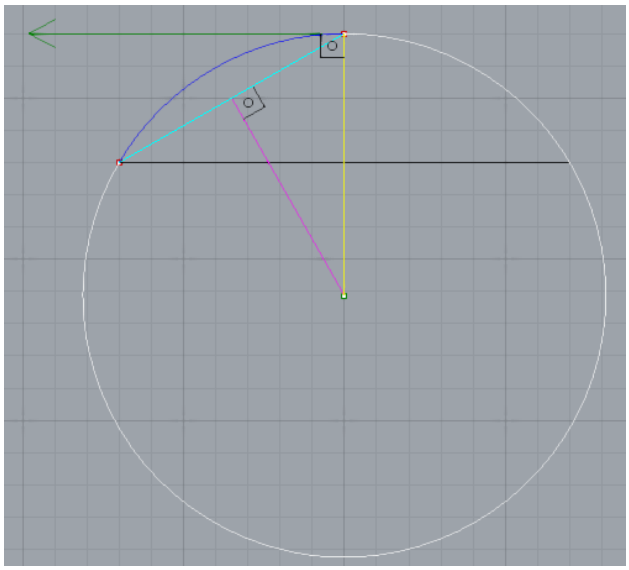


Figure 6: The traditional method to draw a tangent arc

of the line perpendicular to the tangent vector and the line drawn through the midpoint of the line connecting the two known points of the arc is equidistant (as are the sides of an isosceles triangle) to these points. Hence, with a compass centered on this intersection, and starting from one of the known points to the other, the arc can be defined (blue).

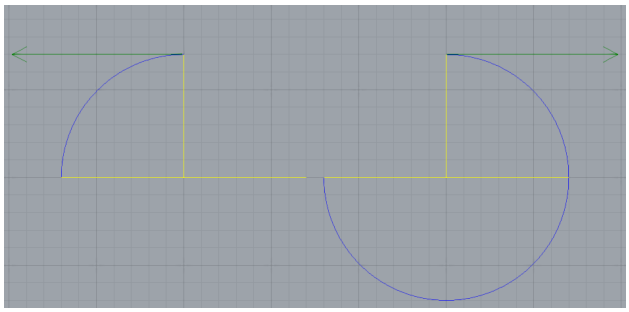


Figure 7: The computational method for drawing the tangent arc

Is important to note that the white arc in Figure 6 is also tangent to the vector, and contains the two known points. The decision to draw one arc or the other, in the traditional method, is made manually, and relies on an understanding of the intended construction. An arc perpendicular to a line is drawn by considering the orientation of the line but not its sense. In the computational environment, when this method of drawing an arc based on two points and a line is implemented, the sense of the line is used to define the sense in which the arc should be drawn. Figure 7 shows the arcs generated by the method using vectors with opposite senses.

In the proposed algorithm, the reference vector is defined by an ordinate segment from the **base center** to the **point on the base** (Fig. 4e). The command that generates an arc from two points and a vector has, as inputs, the **height point**, the **point on the base**, and the vector of the tangent to the **height point** (Fig. 4f). Constructing the arc in this manner constrains the closed dome to a spherical cap, with the height and base radius assuming the values defined by the respective parameters.

The petal is created as a ruled surface, generated by a rotation around the **dome axis** in an angle called the **petal angle** (Fig. 4g). It is obtained by dividing a 360° angel ($2 \times \pi$ in radians) by the **number of petals**.

To simulate the pivoting movement that opens the dome, a rotation axis connecting the base points of the petal needs to be created. This can be done by rotating the **point on the base** around the **dome axis** by taking the **petal angle** as the extent of rotation. A line is drawn between the two points, and is called the **pivot axis**.

The petal surface is rotated around the **pivot axis** by an angle defined by the **pivot rotation** parameter (Fig. 4h). To complete the algorithm and generate the **full dome** (Fig. 4i), a polar array of the **rotated petals** is generated in a 360° angle around the **dome axis** by taking the number of petals as the number of elements of the array.

7. The code

Although virtually any programming language can be used to implement the code, the CAD software Rhinoceros 3D (version 5.11) and its plug-in Grasshopper (version 0.9.0076) were chosen due to the intuitive nature of programming through the connection of graphical components, as well as the fact that the forms would hence be created inside a CAD system that can accommodate architectural projects.

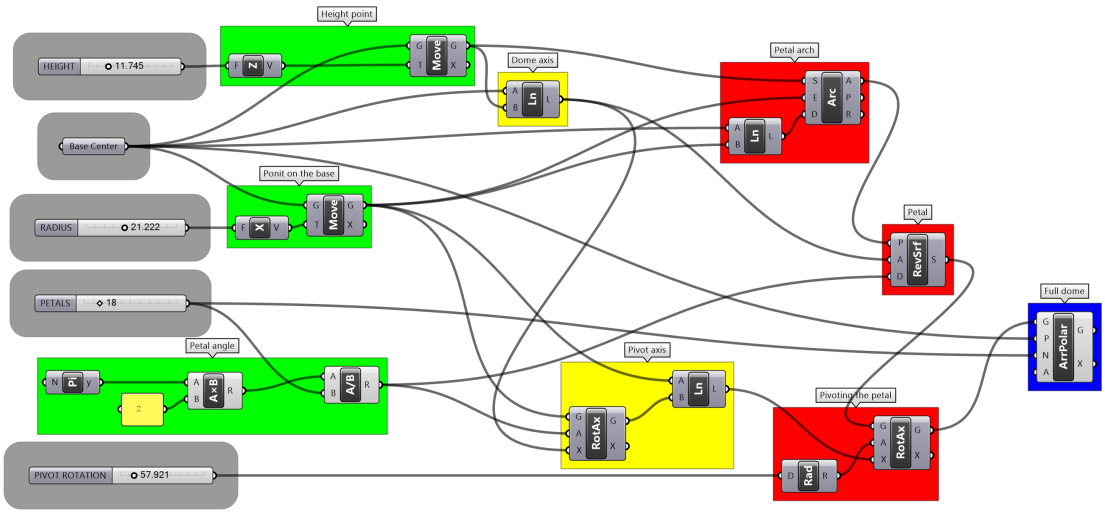
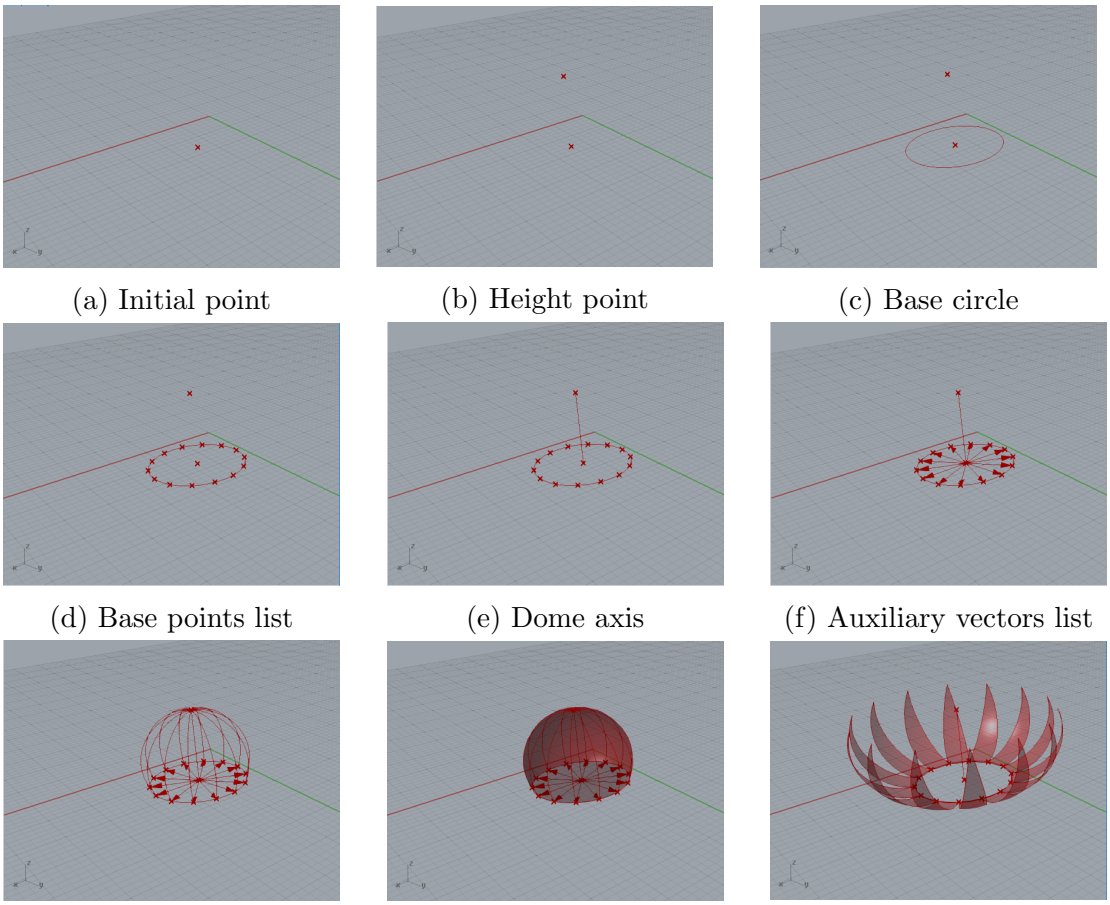


Figure 8: One-petal algorithm



(a) Initial point (b) Height point (c) Base circle

(d) Base points list (e) Dome axis (f) Auxiliary vectors list

(g) Petals arcs list (h) Petals surfaces list (i) Full dome

Figure 9: Steps of the list manipulation-based algorithm

Figure 8 shows the code implemented on the programming interface as well as a flowchart of the algorithm. Data flows from left to right through wire-like connections between components.

The grey boxes show the parameters. From top to bottom: **height**, **base center**, **radius**, **number of petals**, and **pivot rotation**. The top green box generates the **height point**, the middle green **the point on the base**, the bottom green calculates the **petal angle**. The yellow box on top generates the **dome axis** by taking the **base center** and the **height point** as inputs, and the bottom yellow generates the **pivot axis**. The **auxiliary vector** and the **petal arch** are calculated in the top red box, the middle red box performs the revolutions that generated the **petal surface**, and the bottom red box rotates it around the **pivot axis**. The blue box is responsible for the polar array operation that generates the **full dome**.

8. List manipulation-based algorithm

The one-petal algorithm presented in the preceding sections satisfies the conditions and objectives of the proposed generative algorithm. It is an efficient and linear way of defining rules. The list manipulation-based programming strategies are currently related do the LISP programming paradigm proposed by John McCarthy in 1959 [37]. Tightly associated with the artificial intelligence field [38], list manipulation also has many applications in CAD environments.

Let us consider as an example a line-creation command having as inputs points A and B, and draw a line from start point A to end point B. If the second input is substituted by a list of points $B, C, D, andE$, the result of this list operation is a list of lines $\bar{AB}, \bar{AC}, \bar{AD}$, and $\bar{AE}$ created by the application of a single line command. If two lists are used as input ($A, B, C, D, andE$, and $F, G, H, I, andJ$) the result will be a list of lines between equivalent points in each ordinate list ($\bar{AF}, \bar{BG}, \bar{CH}, \bar{DI}$, and $\bar{EJ}$).

A variant of the implementation of the dome algorithm based on list programming is presented in this section to exemplify and introduce the concept. In the one-petal algorithm example, the steps generate one element at a time; in list manipulation-based algorithms, lists of elements are created such that corresponding elements of every petal are drawn simultaneously, and all petals are generated and modified in parallel.

Figure 9 shows the steps of the list manipulation-based strategy. It starts from the **base center** (Fig. 9a) and creates the **height point** (Fig. 9b) in the same way as in the previous example. A circle is drawn using the **base center** and **base radius** parameters as inputs (Fig. 9c). The point at the base of the first algorithm is replaced by a **base point list** (Fig. 9d), all of them restricted to the radial distance from the **base center**. The dome axis (Fig. 9e) is generated as in the previous rule set. Instead of a single vector, an auxiliary vectors list (Fig. 9f) is generated by connecting the **base center** to the **base point list**. The **petal arcs list** (Fig. 9g) is drawn by taking the **height point** as the start, the **base points list** as the end, and the **auxiliary vectors list** as the tangent directions and sense.

Until this point, we have used the ordered lists to execute commands that operate on elements of the lists, and no explicit manipulation has been required. A simple but important operation on the **base points list** is part of the proposed rules, and is used to generate the forms and exemplify the applications of this programming methodology. The idea is to extract the first element of a list, and append it to the end of the list. In the proposed points list represented as A, B, C, D and E, this operation outputs B, C, D, E and A. Generating lines between equivalent points in the two lists results in a list of lines that connects adjacent points: ($\bar{AB}, \bar{BC}, \bar{CD}$, and $\bar{DE}$). In Figure 10 this operation is represented in the bottom green box by taking the **base points list** (middle green box) and generating the **shifted points list**.

The **shifted points list** is directly used in the two steps of the algorithm. The first is used to create a **pivot axes list** by drawing lines between the **base points list** and the **shifted points list** (Fig. 10 bottom yellow box). The second is used to generate auxiliary arcs on the **petal surface list** creation (Fig. 10 bottom red box). The middle green box in Figure 10 generates the **base points list** by dividing the **base circle** by the given **number**

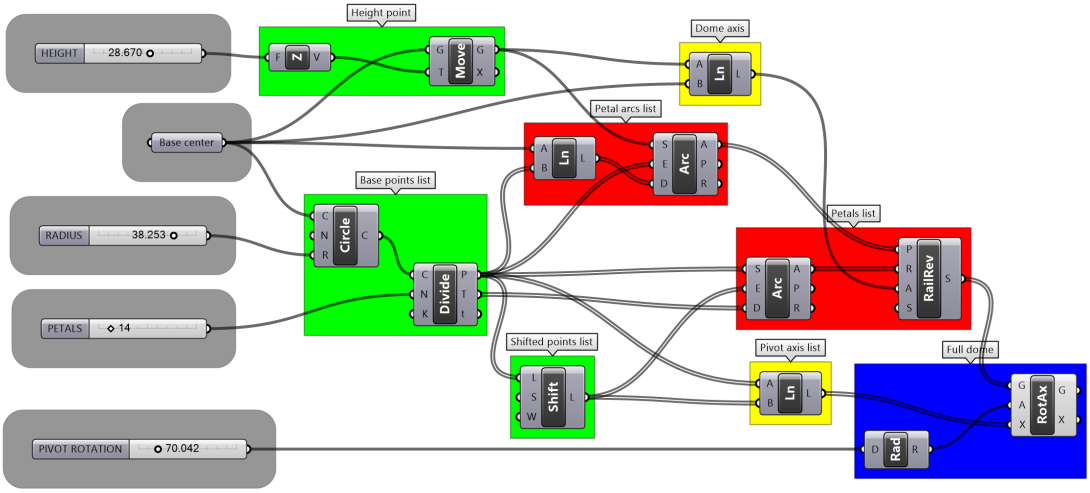


Figure 10: List manipulation-based algorithm

of petals. The Grasshopper curve division component also generates a list of the tangent vectors of the curve at the division points. Arcs are drawn with a start point provided by the **base points list**, end points defined by the correspondent elements in the **shifted points list**, and with the sense and directions of the equivalent tangent vectors. These auxiliary arcs are used in place of the **petal angle** in the first algorithm to define the magnitude of the revolution angle of the petal surface (Fig. 10 bottom red box) around the **dome axis** (Fig. 10 top yellow box). Depending on the programming environment in which the code is implemented, the strategy of using auxiliary arcs or the one that calculates the **petal angle**, presented in Section 6, could be considered more suitable to generate the petals' surface, but both produce the same results.

The petal surface list (Fig. 9h) is rotated around the correspondent **pivot axes list** by the angle provided by the pivot rotation parameter, hence drawing the **full dome** (Fig. 9i).

9. Analyzing the algorithm outputs

The flowchart in Figure 1 shows the scenario where, once the idea is established, the rules are planned, the code is implemented, and a form is generated by the first set of parameters, the designer is in charge of evaluating the result, resetting the parameters, and/or modifying the code, and continually repeats this proceedings until a constructive form is selected as the one to be built.

Figure 11 shows a set of solutions generated exclusively by modifications made over the values of the parameters. One of the decisions that the designer should make is the size of the **base radius**, which is strongly related to the amount of light and the dynamics of the air that should circulate in and out of the building. This evaluation can depend on the

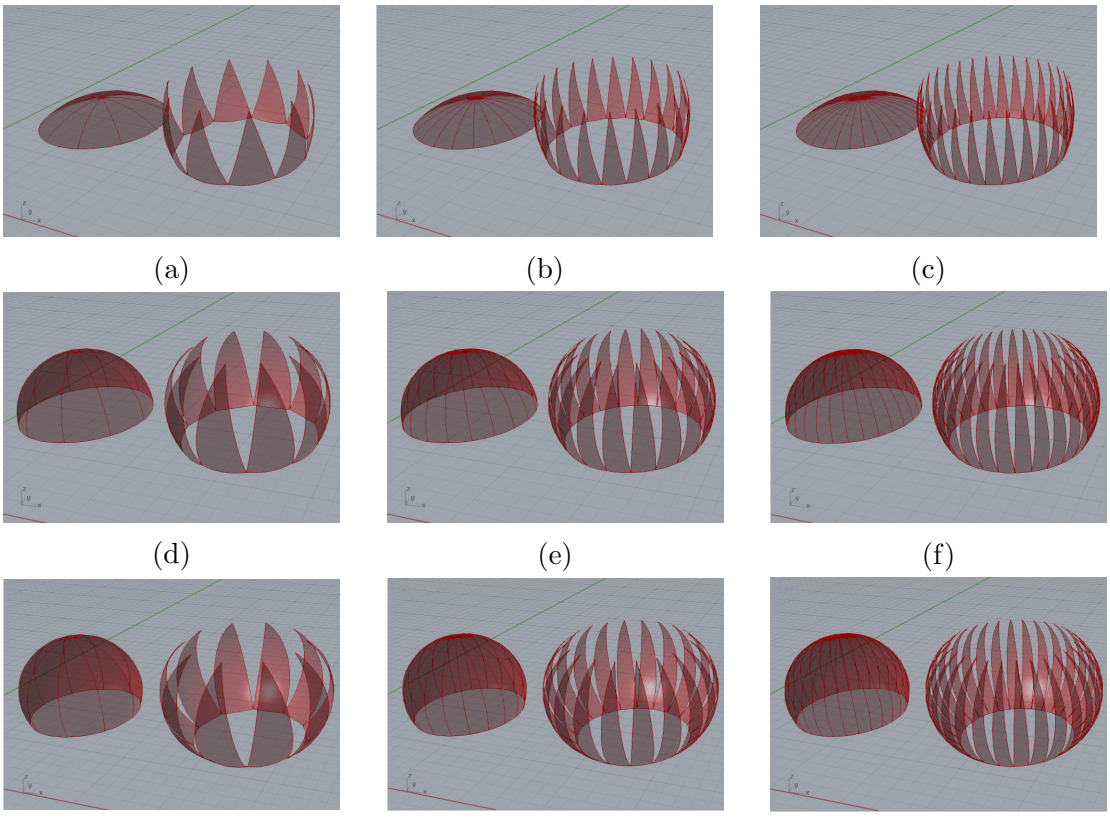


Figure 11: Variant domes created by the algorithms

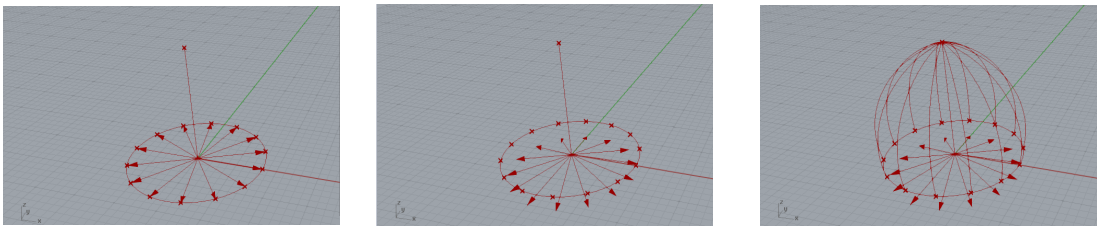
designer's experience, and intuition or calculations that can be performed without the aid of computational analysis tools. Since the variations of forms are automatically generated, a tool that analyzes 3D models tends to be more efficient for the task.

Not only the building itself, but the characteristics of the environment should be considered in this choosing process. However, some relations between the parameters result in relevant changes in the generated geometries. In the first row of Figure 11, the domes have larger values for the **base radius** than for the **height** parameter; the second line shows equal values, and in the third row, the **radius** is smaller than the **height**. This fact can be easily demonstrated geometrically.

The aesthetic impression made by the different simulations can be observed in the images of Figure 11, but is better understood when compared to the domes and respective buildings of Figures 2 and 3. The first one represents a larger value for the **height** parameter than the **radius**, producing a fluid relation between the curves of the building's forms and the dome as a finishing element. An inverted configuration of parameters in the relations between these values generates a brusque or even flat interruption of the composition of the forms. In the second figure, the domes adopt a more discrete elevation from the base, dialoguing with the almost-flat roof solution of the buildings.

While the quantitative investigations can be numerically evaluated, the qualitative aspects, as aesthetics, can only be approached and addressed by the raising and answering of some loosely defined questions. The simple analysis proposed here aims to exemplify how these subjective questions can be handled by the methodology of generative algorithms.

10. Modifying the code

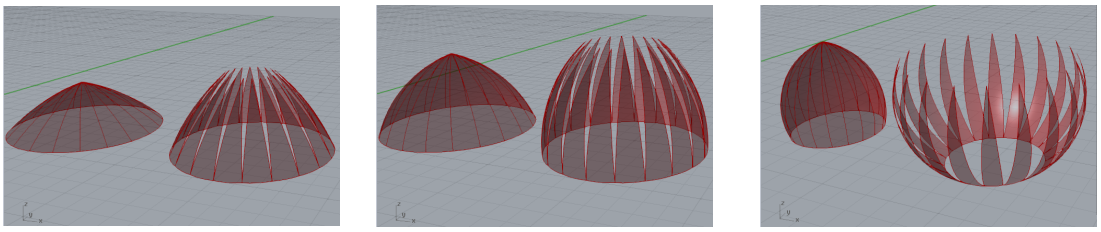


(a) Auxiliary vectors list (b) Rotated tangents list (c) Petals arcs lists

Figure 12: Steps of the lancet dome algorithm

Modification of the parameters is the fastest way to generate alternative results in a generative design system. The results registered in Figure 11 can be generated by both the algorithms presented, but the flowchart in Figure 1 foresees that the code and the rules can also be modified within the scope of the methodology proposed here. To illustrate this possibility, some changes in the list manipulation-based algorithm are proposed to enable the code to create not only spherical cap domes, but also geometries derived from lancet arcs.

A few modifications are needed to complete this task. The steps presented in Figure 9 are followed in exact order until the **auxiliary vectors** are created (Fig. 9f). Then, the vectors undergo a rotation defined by a new parameter called **tangent rotation**, and the **petals arcs list** is created by taking these new vectors as inputs. Following this, **the petal surface** list and the **full dome** are created by the same procedures as shown in Figures 9h and 9i.



(a) (b) (c)

Figure 13: Variant domes created by the lancet arcs algorithm

If the tangent rotation parameter is equal to zero, the results are constrained to geometries similar to the ones presented in Figure 11; otherwise, they assume the pointed end of a lancet arc, as Figure 13 illustrates.

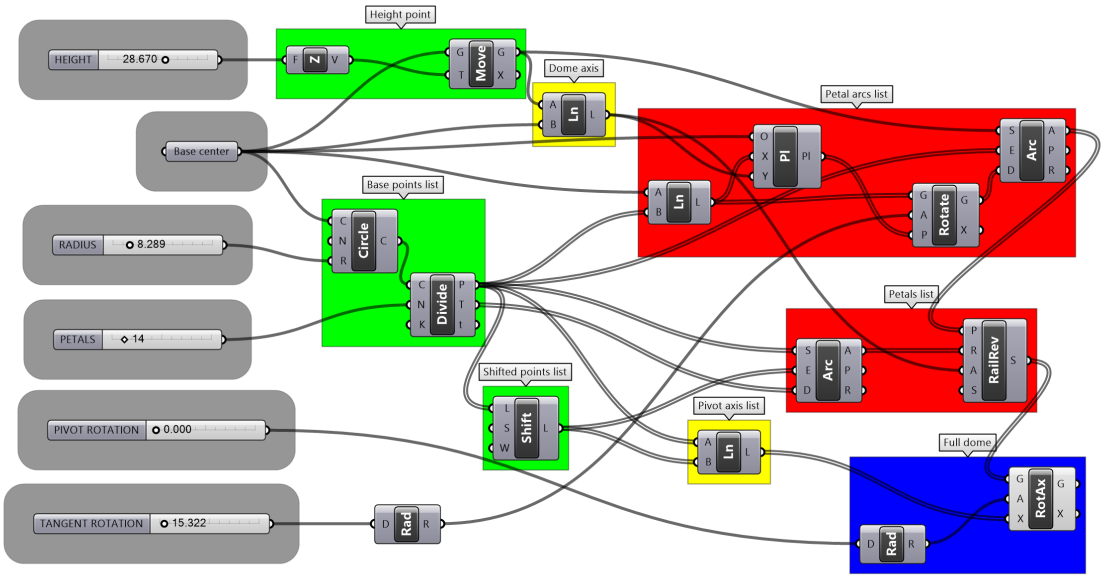


Figure 14: Lancet arc algorithm

The visual code presented in Figure 14 shows these changes in comparison with Figure 10. The **tangent rotation** parameter is added (bottom grey box), and the **auxiliary vectors** are rotated around the z-axes of a list of planes created with origin at the **base center**. The x-axes are rotated in the direction and sense of the **auxiliary vectors**, and the y-axes follow the **dome axis**-oriented line. This rotated tangents' list is entered as the vector required by the **petal arcs list** creation component (top red box).

11. Conclusion

The proposed exercise discussed in this article succeeded in approaching the basics aspects of the generative design methodology, encompassing all possible variants of the flowchart in Figure 1, explaining the most important geometric constructions used, providing an overview of some programming procedures, and presenting the manner in which constructive forms can be chosen from both qualitative and quantitative aspects.

The one-sentence definition of the idea purposely omits the fact that in the stage of analysis of the domes' references, they are all spherical caps. If this were stated, the modification to the algorithm in Section 10 would also have modified the idea. Although this modification is not described in the flowchart (Fig. 1), it is a possible move in the process of a generative algorithm. Since code reuse is a common programming activity, the case of the stated characteristic of an idea being modified along these lines should not be treated as an issue, but as a natural possibility.

A comprehensive understanding of generative design systems demands an interdisci-

plinary perspective along with the interdisciplinary nature of architecture. The scope of all the different disciplines involved in this exercise was not comprehensively covered by any means, but the effort of bringing these ideas together in a coherent context should guide future work in the area.

References

[1] S. Krish, A practical generative design method, Computer-Aided Design 43 (1) (2011) 88–100. doi:10.1016/j.cad.2010.09.009.
URL http://linkinghub.elsevier.com/retrieve/pii/S0010448510001764

[2] C. Ceccato, The Master-Builder-Geometer, in: C. Ceccato, L. Hesselgren, M. Pauly (Eds.), Advances in Architectural Geometry 2010, Springer, 2010, pp. 9–14.

[3] R. Oxman, Theory and design in the first digital age, Design Studies 27 (3) (2006) 229–265. doi:10.1016/j.destud.2005.11.002.
URL http://linkinghub.elsevier.com/retrieve/pii/S0142694X05000840

[4] A. Picon, Ornament and its users: from the Vitruvian tradition to the digital age, Le Visiteur (17) (2011) 176–180.
URL http://dash.harvard.edu/handle/1/12638041

[5] B. Kolarevic, Architecture in the digital age: design and manufacturing, London, 2003.

[6] C. Ceccato, Others (Eds.), Towards Teaching Generative Design in Architecture, Springer, 2010.

[7] D. E. Knuth, Art of Computer Programming, Volume 1: Fundamental Algorithms, Pearson Education, 1997.
URL http://books.google.com.br/books?id=x9AsAwAAQBAJ

[8] K. Terzidis, Algorithmic Architecture, Taylor & Francis, 2006.
URL http://books.google.com.br/books?id=yT7NXhZYepwC

[9] W. J. Mitchell, World's Greatest Architect : Making, Meaning, and Network Culture, MIT Press, Cambrige, 2008.
URL http://books.google.com.br/books?id=DxszH9whFSIC

[10] E. Fasoulaki, Integrated Design, Ph.D. thesis, MIT, Cambrige (2008).

[11] H. Bohnacker, B. Gross, J. Laub, C. Lazzeroni, Generative Design: Visualize, Program, and Create with Processing, Princeton Architectural Press, Princeton Architectural Press, 2012.
URL http://books.google.com.pe/books?id=tSS9uAAACAAJ

[12] I. G. DIno, Creative Design Exploration By Parametric Generative Systems In Architecture, Metu Journal of the Faculty of Architecture (2012) 207–224doi:10.4305/METU.JFA.2012.1.12.

[13] G. Stiny, J. Gips, shape Grammars and the Generative Specification of Painting and Sculpture, in: Petrocelli (Ed.), The Best Computer Papers of 1971, Auerbach, 1972, pp. 125–135.

[14] G. Stiny, W. J. Mitchell, The Palladian grammar, Environment and Planning B 5 (1) (1978) 5–18.
URL http://www.envplan.com/abstract.cgi?id=b050005

[15] J. P. Duarte, A discursive grammar for customizing mass housing: the case of Siza's houses at Malagueira, Automation in Construction 14 (2) (2005) 265–275. doi:10.1016/j.autcon.2004.07.013.
URL http://www.sciencedirect.com/science/article/pii/S0926580504000810

[16] B. Tepavcevic, V. Stojakovic, Shape grammar in contemporary architectural theory and design, Facta universitatis - series: Architecture and Civil Engineering 10 (2) (2012) 169–178. doi:10.2298/FUACE1202169T.
URL http://www.doiserbia.nb.rs/Article.aspx?ID=0354-46051202169T

[17] L. Caldas, L. Norford, A Genetic Algorithm Tool For Design Optimization, in: Media and Design, Acadia'99, Salt Lake City, 1999, pp. 260–261.

[18] L. Troiano, C. Birtolo, Genetic algorithms supporting generative design of user interfaces: Examples, Information Sciences (2012) 1–19doi:10.1016/j.ins.2012.01.006.
URL http://linkinghub.elsevier.com/retrieve/pii/S0020025512000242

[19] L. G. Caldas, L. K. Norford, A design optimization tool based on a genetic algorithm, Automation in Construction 11 (2) (2002) 173–184. doi:10.1016/S0926-5805(00)00096-0.
URL http://linkinghub.elsevier.com/retrieve/pii/S0926580500000960

[20] C. M. Herr, T. Kvan, Adapting cellular automata to support the architectural design process, Automation in Construction 16 (1) (2007) 61–69. doi:10.1016/j.autcon.2005.10.005.
URL http://www.sciencedirect.com/science/article/pii/S0926580505001494

[21] B. Ceranic, C. Fryer, R. Baines, An application of simulated annealing to the optimum design of reinforced concrete retaining structures, Computers & Structures 79 (17) (2001) 1569–1581. doi:10.1016/S0045-7949(01)00037-2.
URL http://linkinghub.elsevier.com/retrieve/pii/S0045794901000372

[22] V. M. Correia, C. a. M. C. A. Mota Soares, Refined models for the optimal design of adaptive structures using simulated annealing, Composite Structures 54 (2-3) (2001) 161–167. doi:10.1016/S0263-8223(01)00085-X.
URL http://www.sciencedirect.com/science/article/pii/S026382230100085X

[23] L. Lamberti, An efficient simulated annealing algorithm for design optimization of truss structures, Computers & Structures 86 (19-20) (2008) 1936–1953. doi:10.1016/j.compstruc.2008.02.004.
URL http://linkinghub.elsevier.com/retrieve/pii/S0045794908000448

[24] J. Bland, G. Dawson, Tabu search and design optimization, Computer-Aided Design 23 (3) (1991) 195–201. doi:10.1016/0010-4485(91)90089-F.
URL http://www.sciencedirect.com/science/article/pii/001044859190089F

[25] G.-C. Luh, C.-Y. Lin, Y.-S. Lin, A binary particle swarm optimization for continuum structural topology optimization, Applied Soft Computing 11 (2) (2011) 2833–2844. doi:10.1016/j.asoc.2010.11.013.
URL http://linkinghub.elsevier.com/retrieve/pii/S1568494610002905

[26] M. Yahya, M. Saka, Construction site layout planning using multi-objective artificial bee colony algorithm with Levy flights, Automation in Construction 38 (2014) 14–29. doi:10.1016/j.autcon.2013.11.001.
URL http://www.sciencedirect.com/science/article/pii/S0926580513001945

[27] M. Turrin, P. von Buelow, R. Stouffs, Design explorations of performance driven geometry in architectural design using parametric modeling and genetic algorithms, Advanced Engineering Informatics 25 (4) (2011) 656–675. doi:10.1016/j.aei.2011.07.009.
URL http://linkinghub.elsevier.com/retrieve/pii/S1474034611000577

[28] T. Fischer, M. Burry, J. Frazer, Triangulation of generative form for parametric design and rapid prototyping, Automation in Construction 14 (2) (2005) 233–240. doi:10.1016/j.autcon.2004.07.004.
URL http://linkinghub.elsevier.com/retrieve/pii/S0926580504000780

[29] L. Lachauer, H. Jungjohann, T. Kotnik, Interactive parametric tools for structural design, in: Proceedings of the IABSE-IASS Symposium 2011, London, UK, 2011.
URL http://www.schwartz.arch.ethz.ch/Publikationen/Dokumente/InteractiveTools.pdf

[30] S. Milena, M. Ognen, Application of Generative Algorithms in Architectural Design, in: Advances in Mathematical and Computational Methods, 2010, pp. 175–180.
URL http://www.wseas.us/e-library/conferences/2010/Faro/MACMESE/MACMESE-27.pdf

[31] C. M. Hoffmann, R. Joan-Arinyo, A brief on constraint solving, Computer-Aided Design and Applications 2 (5) (2005) 655–663. doi:10.1080/16864360.2005.10738330.

[32] R. Barták, Theory and practice of constraint propagation, in: In Proceedings of the 3rd Workshop on Constraint Programming in Decision and Control, 2001, pp. 7–14.

[33] S. Philippou, The primitive as an instrument of subversion in twentieth-century Brazilian cultural practice, arq: Architectural Research Quarterly 8 (3-4) (2004) 285–298. doi:10.1017/S1359135504000302.

[34] M. C. Ferraz, G. Latorraca, I. G. Ferraz, E. S. de Freitas, S. Birkinshaw, K. Szabó, João Filgueiras Lima, Lelé, Arquitectos brasileiros - Brazilian architects, Editorial Blau, 2000.
URL https://books.google.com.br/books?id=hHJEAQAAIAAJ

[35] G. Campagnol, Positive Distraction and the Rehabilitation Hospitals of João Filgueiras Lima, HEALTH ENVIRONMENTS RESEARCH & DESIGN JOURNAL 8 (1) (2014) 199–227.

[36] A. A. Maciel, B. Ford, R. Lamberts, Main influences on the design philosophy and knowledge basis to

435 bioclimatic integration into architectural designThe example of best practices, Building and Environ-
436 ment 42 (10) (2007) 3762–3773. doi:10.1016/j.buildenv.2006.07.041.
437 URL http://www.sciencedirect.com/science/article/pii/S0360132306003052
438 [37] J. McCarthy, Recursive functions of symbolic expressions and their computation by machine, Part I,
439 Communications of the ACM (April) (1960) 1–34.
440 URL http://dl.acm.org/citation.cfm?id=367199
441 [38] G. L. Steele, R. P. Gabriel, The Evolution of Lisp, ACM, New York, NY, USA, 1996, Ch. The Evolut,
442 pp. 233–330. doi:10.1145/234286.1057818.
443 URL http://doi.acm.org/10.1145/234286.1057818