Incessant Replication

Computational Floor Plan Generation

Ron Aasholm
Incessant Replication

Computational Floor Plan Generation
Ron Aasholm

Master's Thesis
2015
Aalto University Department of Architecture
While studying architecture I came across two types of attitudes towards *computers.*

That their usage is revolutionary, swift and organized or that nothing is more fallacious, inept and worthless.

Yet neither of them seemed to originate from a relationship with *computing.*
# Incessant Replication: Computational Floor Plan Generation

**Contents**

- Abstract ................................................................. 7
- Glossary ................................................................. 11
- 0.Prologue ................................................................. 15

## 0. Prologue

- 0.1 Research framing .................................................. 21
- 0.2 Research methodology ............................................ 25
- 0.3 Research structure ................................................. 28

## I. Systems, models and procedures

1.1 Indeterminate computing

- Computer Aided vs. Computational ................................ 37
- Algorithm ....................................................................... 40
- Scripting ....................................................................... 42
- Parametric(ism) ................................................................. 47
- Computerized feasibility analyses ....................................... 49

1.2 Before the digital was the digital .............................. 58

- Ontology of models ......................................................... 64
- Classification of models ..................................................... 65

1.3 Models for computer generated architecture .................. 67

## II. History of computation in architecture

2.1 Early history ............................................................ 71

2.2 Generation I 1946-1959: Tedious experiments ............... 74

2.3 Generation II 1960-1963: Into mainstream .................. 77

2.4 Generation III 1964-1975: Loss of innocence .................. 80

2.5 Generation IV 1975-ongoing: Personal Computers ......... 83

2.6 1990s: Deleuzian remedy ........................................... 90

2.7 2000s: Morphology of the emergent ............................ 93

2.8 2010s: Parametricism .................................................. 98

## III. Floor plan representation

3.1 Syntactic approach ...................................................... 101

3.2 Syntactic lineage ........................................................ 105

3.3 Computational Generation

- Primitive methods .......................................................... 110
- Procedural ...................................................................... 112

3.4 Floor Plan Representation ............................................ 115

3.5 Conclusion .................................................................... 118

3.6 References .................................................................... 123

3.7 Acknowledgements ....................................................... 126

3.8 Index ............................................................................ 128
Additive Space Allocation 125
Permutational Space Allocation 130
Heurist 133
Analogical methods 133
Case based 139
Expert systems 140
Shape grammars 140
Advanced 143
Genetic Engineering Approach 146
Hierarchical Growth Approach 152

IV. Computing solutions 159
4.1 Illustrating the model setup 163
State-action graph 166
Phase space 168
Fitness landscape 171
4.2 Model search 180
The scope of problems 181
Metaheuristics 182
Nature-inspired or non-nature inspired 184
Population-based or single point search 184
Dynamic or static objective function 184
One or various neighborhood structures 184
Memory usage or memory-less methods 185
Evolutionary computing 186
Description of the Individuals 187
Evolution Process 187
Neighborhood Structure 188
Information Sources 188
Infeasibility 188
Intensification Strategies 189
Diversification Strategy 189
Galapagos 190
Overview 191
Fitness Function 197
Selection Mechanism 201
Coupling Algorithm 201
Coalescence Algorithm
Mutation Factory
Maintain High Fitness

V. The experiment

5.1 The prototypical building
5.2 Experiment construction
5.3 Model anatomy
   Element area
   Element shape
   Element adjacencies
   Overlap of elements
   Bounding perimeter
5.4 Solver settings
   Fitness function
   Population
   Breeding
5.5 Technical demonstration conclusions
   Setup interface
5.6 Demonstration reflections
   Novelty

VI. Thesis Contemplations

6.1 Methodology from computation
   Negative generative approach
   Change in design initiation
   Genocide of darlings
   Peer reviewed architecture
   Metaheuristic influences
6.2 Future implementation
   Near future tools
   6.2.2 Leapfrogging technology
6.3 Postscript

Acknowledgements
Appendix
Bibliography
Abstract

The supposed *algorithmic architecture* has recently emerged as a distinct field within architecture. It is infested with ambiguous research while no rudimentary study of how to practise computational design in general exists. This thesis thus studies the process of promptly harnessing a computational design methodology and applies it to floor plan design.

The research methodology consists mostly of literature review however a demonstration computation is carried out to support it. The research is divided into six parts, elucidating the process of constructing a computational model for generating floor plans.

There exists many misconceptions about design computation in architecture and it is mostly because methodology is acquired from computer sciences through informal studies.

Curvilinear geometry and generative methodologies are not intrinsic to computational architecture. The digital computer however provided significant processing capacity that elevated computational design into a field of its own.

Much of the innovation in computational design took place in 1970s. Only the recent developments in technology and software, especially the graphical algorithm editors, have provided new momentum and wider acceptance for computational design.

There exists formal computational precedent models capable of generating floor plans. The scope of their quality varies from designs barely recognizable as a floor plan to designs that have succeeded in architectural competitions. There is a lineage of models that incorporate higher-level algorithmic constructs: Metaheuristics. The subject is introduced and evolutionary metaheuristic are explained in general since they are most widely used in architectural computations. A specific evolutionary algorithm, found in Galapagos, is analysed and applied as it used in the model demonstration.

Floor plan generation has many drawbacks, yet it is possible to generate schematic designs of architectural program with reasonably primitive setup. The computational methodology however has multiple connection points to design methodology in general and were it perceived as such it could significantly benefit the processes of architectural design.
Aalto-Yliopisto
Taiteiden ja suunnittelun korkeakoulu

Tekijä Ron Aasholm
Työn nimi Incessant Replication: Computational Floor Plan Generation

Laitos Arkkitehtuurin laitos
Professuuri Rakennussuunnittelu
Professuurikoodi A-8

Työn valvoja Pirjo Sanaksenaho
Työn ohjaaja Otso Helenius

Vuosi 2015
Sivumäärä 294
Kieli Englanti

Avainsanat pohjapiiros generointi, automoitu tilallinen suunnittelu, tila allokaatio, laskennallinen suunnittelu, metaheuristiikka, evolutiivinen algoritmi, Galapagos
Tiivistelmä

Niin sanottu algoritminen arkkitehtuuri on viime vuosikymmenen aikana erottautunut selkeäksi alakategoriaksi arkkitehtuurissa. Se tuottaa kuitenkin epämääräisiä ja hyvin tilannesidonnaista tutkielmia, joissa metodologiaa lähinnä sivutaan. Siksi tämä tutkimus pyrkii esittämään yksinkertaisen metodin siitä kuinka tietokoneen laskentatehoa voidaan hyödyntää pohjapiirrosten generoinisessa.

Tutkimus koostuu kirjallisuuskatsauksesta ja kokeesta jossa generoidaan pohjapiirros. Kokeen tarkoitus on tukea kirjallisuutta ja luoda toisenlainen näkökulma aiheeseen. Tutkimus on jaettu kuuteen osaan, jotka selvittävät pohjapiirrosten generointiin tarvittavan tietokonemallin rakentamista.

Laskennalliseen arkkitehtooniseen suunnitteluun liittyvät paljon harhakuivia, koska metodologia on tuotettu epäformaalien arkkitehtuuritutkielmien avulla.

Vapaamuotoinen arkkitehtuuri ja generatiivinen suunnittelu eivät ole synonymeitä laskennalliselle suunnittelulle. Digitallinen tietokone kuitenkin omaa merkittävän laskennallisen kapasitettin, joka on korostanut laskennallisen suunnittelun asemaa itsenäisenä metodologiana.


Pohjapiirrustus generointi on raskas ja ongelmikas prosessi, vaikka se on kuitenkin täysin mahdollista hyvin yksinkertaisilla järjestelyillä. Laskennallisella suunnittelulla kuitenkin on suoria yhtäläisyksiä perinteisen arkkitehtisuuonittelun kanssa. Niiden parempi käsittäminen hyödyttäisi koko suunnittelun prosessia merkittäville ja yllättäville tavoilla.
Glossary

Algorithm |ˈalgərɪð(ə)m| noun
- A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

Code |kəʊd| noun
- A system of words, letters, figures, or symbols used to represent others, especially for the purposes of secrecy.
- A series of letters, numbers, or symbols assigned to something for the purposes of classification or identification.
- Computing program instructions: assembly code.
- To write code for a computer program.

Compute |kəmˈpjuːt| verb
- Reckon or calculate a figure or amount.

Computing |kəmˈpjuːtɪŋ| noun
- The use or operation of computers.

Emergence |ɪˈməːdʒ(ə)ns| noun
- The process of becoming visible after being concealed.
- The process of coming into existence or prominence.
Evolution |ˌiːvəˈluːʃ(ə)n, ˈɛv-| noun
- The process by which different kinds of living organism have developed from earlier forms during the history of the earth.
- The gradual development of something.

Derivatives: evolutional, evolutionarily, evolutionarily, evolutionary, evolutive.

Exhaustive |ɪgˈzɔːstɪv, ɛg-| adjective
- Including or considering all elements or aspects; fully comprehensive.

Function |ˈfʌŋ(k)ʃ(ə)n| noun
- A basic task of a computer, especially one that corresponds to a single instruction from the user.
- A relation or expression involving one or more variables.
- A variable quantity regarded in relation to one or more other variables in terms of which it may be expressed or on which its value depends. The magnetic field has varied as a function of time.

Generative |ˈdʒɛn(ə)rətɪv| adjective
- Relating to or capable of production or reproduction.
- Denoting an approach to any field of linguistics that involves applying a finite set of rules to linguistic input in order to produce all and only the well-formed items of a language.

Geometry |dʒɪˈɒmɪtri| noun (pl.geometries) [mass noun]
- The shape and relative arrangement of the parts of something.

Haptic |ˈhaptɪk| adjective
- Relating to the sense of touch, in particular relating to the perception and manipulation of objects using the senses of touch and proprioception.
Heuristic |ˌhjʊ(ə)ˈrɪstɪk| adjective
- Enabling a person to discover or learn something for themselves.
- Computing proceeding to a solution by trial and error or by rules that are only loosely defined.

Model |ˈmɒd(ə)l| noun
- A three-dimensional representation of a thing or of a proposed structure, typically on a smaller scale than the original.
- A simplified description, especially a mathematical one, of a system or process, to assist calculations and predictions.

Parameter |pəˈramɪtə| noun
- A numerical or other measurable factor forming one of a set that defines a system or sets the conditions of its operation.
- A quantity whose value is selected for the particular circumstances and in relation to which other variable quantities may be expressed.
- A limit or boundary which defines the scope of a particular process or activity.

Programme |ˈprəʊgram| (US program) noun
- A series of coded software instructions to control the operation of a computer or other machine.
- Provide a computer or other machine with coded instructions for the automatic performance of a task.
- Input instructions for the automatic performance of a task into a computer or other machine.

Stochastic |stəˈkastɪk| adjective
- Technical having a random probability distribution or pattern that may be analysed statistically but may not be predicted precisely.
Structuralism |ˈstrʌktʃ(ə)r(ə)lɪz(ə)m| noun
- A method of interpretation and analysis of aspects of human cognition, behaviour, culture, and experience, which focuses on relationships of contrast between elements in a conceptual system.
- The doctrine that structure is more important than function.

Syntax |ˈsɪntaks| noun
- The arrangement of words and phrases to create well-formed sentences in a language.
- A set of rules for or an analysis of the syntax of a language.
- The structure of statements in a computer language.

System |ˈsɪstəm| noun
- A set of things working together as parts of a mechanism or an interconnecting network; a complex whole.
- A set of principles or procedures according to which something is done; an organized scheme or method.

Technocracy |tɛkˈnɒkrəsi| noun (pl.technocracies)
- The government or control of society or industry by an elite of technical experts.

Topology |təˈpɒlədʒi| noun (pl.topologies)
- The study of geometrical properties and spatial relations unaffected by the continuous change of shape or size of figures.
- The way in which constituent parts are interrelated or arranged: the topology of a computer network.

Derivatives: topological
Prologue

“It is popularly assumed that any architectural research of a mathematical nature must have functionalist aims ... that it seeks to devise ways in which the design of a building can be formulated as a mathematical ‘problem’, and mathematically ‘solved’.”

- Philip Steadman

After the breakthrough of computers into the conventional society during the 1960s they have gradually been integrated into architectural practises as well. Yet, this development has mostly been concerned with computer aided design and not computation itself. Architects have not harnessed the computational power of the computer for design, alas they have merely used it as a design aid. It also appears that the scene of computing in architecture has collapsed into its own sphere with open ended research and exclusive thinktanks with their exclusive studios, who promote the revolution of the digital with ambiguous design research. These vague outcomes are then used to assert the supreme abilities of computations for solving issues. Even though there exist many

004 Derix, ‘Mediating Spatial Phenomena through Computational Heuristics.’, 61.
publications on this alleged design research and on the subject of computational design in architecture, most of it is hard to approach without extensive prior knowledge of the subject. Even so, majority of it is theoretical or philosophical writing, occasionally intertwining biology and technology. Practically no formal instance — namely school, thinktank or office — is disclosing their actual inner workings and knowledge on the subject.

Frankly, one obtains the impression that this exclusion is further aggregated as digital design in architecture — whether computational or aided — seems to be more or less opposed by some of the professionals of architecture that practice outside the sphere of digital architecture, as either futile or fallacious. Large part of the criticism is delivered by those with next to no knowledge, or experience in computational design. The situation is sadly polarized, while instead, there could be a lot of productive exchange. Both parties should be reminded of the definition of scepticism — “critical examination, evidence-based scientific inquiry, and the use of reason in examining controversial claims” — and the ideological conviction — basing one's “conclusions on a priori convictions” and to “flatly deny the results” without examination. Namely, it seems that the supporters of the digital fail in the critical scrutiny of their preachings, that is in their scepticism, while the opposing majority base their opinion on misconceptions, the ideological convictions.

This current situation might suggest why major implementations of computation into practise are still on their way, even though most of the reasoning and theoretical ground were set already several decades ago. It is an interesting notion that in many other fields of science, algorithms have been a keen part of the design or decision making processes already since the 1970s. In some cases, the debate has even gone to the lengths of people making ethical arguments for their use instead of human intelligence: researchers have argued that making decisions based on intuitive judgements is unethical, if a working algorithm is

006 (Krauss, 2014)
007 Burry, Scripting Cultures: Architectural Design and Programming, 78.
Yet, there exists a third clique besides the alleged digital evangelists and digital deniers: A scientific community of researchers within the architectural profession who, contrary to the first two groups, are quite open about their studies and findings. Their work, however, remains highly unattainable for a designer or an architect with a typical education of art and design. The inherent scientific approach and the actual findings in the research require advanced knowledge of mathematics, computation and even biology for one to fully appreciate them.

Resulting from this situation, this thesis is not an attempt to be a part of the excluded scene of digital computation within architecture, and bolster its alleged merits once more in the face of the sceptical majority. Needless to say, it is not joining the ranks of the deniers, nor is it capable of being an avant-garde research of the technical kind, as it is, in the end, merely a thesis study. Instead, it tries to frame itself in the junction of the above mentioned three groups to form an objective starting point for general understanding of computation in architectural design.

At the moment there is no clear dialogue between these groups. No publication or a research has clearly gathered relevant general information within one publication. It seems that the task is considered as too elementary a status for designers or scientist. Instead they continue with the development of specific case studies and applications. If the above mentioned is the case, then a thesis research should be rudimentary enough to address that task.

As such, this thesis seeks to proceed along the lines of the general stances represented by each of the affiliated three groups. First, it is a research about computation in architecture, which is then the focus area it tries to understand and elaborate. Secondly, it tries to be as explicit as possible, and easily understandable for architects and other designers who might lack the prior technical knowledge. Thirdly, it seeks to remain sceptical about its subject, in order to improve the status of computation within architecture and thus, to allow meaningful dialog of it usage in design to emerge.

Thus, in short, the aim of this thesis as a publication is to serve as a precursory introduction to formal design computation.

---

Kahneman, Thinking, Fast and Slow, 229.
This approach is taken because the critical theory and building science have both proven to provide neither “immediate influence on practical designs or designers”\textsuperscript{010}, nor significant research on the analysis of their impact.\textsuperscript{011}

The paperback format, through which this thesis manifests itself, is a statement in itself of the excessive amount of graphical hegemony attached to computation in architecture.

\textsuperscript{010} Coates, The Architecture of Programming, 1.
\textsuperscript{011} Lynn, Archaeology of the Digital, 11.
Research framing

“Yet digital tools per se do not necessarily impose malumian shapes.”

- Mario Carpo

To narrow down the research topic of computing in architecture, this thesis takes on art historian August Schmarsow’s definition of architecture as a *spatial art* instead of style and construction. Therefore, architectural design in this research is mostly referring to the design of program and spatial configurations. Any other architectural aspect, such as structure, materiality, social logic, fabrication, and aesthetic style is not of similar relevance. As this thesis is about computation, the capabilities of computing are always reflected on spatial layouts, and more generally speaking, on floor plans. The question this thesis is trying to answer is how computers could design floor plans. What kind of a setup is needed to carry out this kind of a process? In other words, the formal research question here is: How to construct a computational model for generating floor plans?

There exists a wide body of terms referring to the subject at hand, the floorplan generation. Such examples include: *spatial layouting, space allocation, automated spatial synthesis, quadratic assignment formulation, automated floor plan generation and space layout planning*. This research seeks to use the term *floorplan*

012 Carpo, The Alphabet and the Algorithm, 85.
generation, as it is derived from common architectural language and all the processes, no matter what term is used to describe them, are eventually used to construct a floor plan. In this manner, there is little room for confusion about the subject being studied. Although in some cases, especially when describing methods and precedents, it is needed to refer to layouts, when it is more descriptive of the geometries produced by computers. A layout can be seen as an intermediate representation of a floor plan.

By taking on a general subject such as the floor plan design, an approachable common ground between the earlier mentioned polarized groups can be found. Meaning, all architects, no matter what background, share an interest in the programmatic logic of architectural design regardless of the scale, style, construction, or social approach. In the end, it is the spatial design that is often considered the essence of architecture - at least since the end of renaissance. The architectural plan drawing representing spatial layout design is also abstract and simple enough to illustrate the workings of the actual computation, that is, how the computer proceeds to transform the geometry in order to create a design. In this manner, an overall understanding of computation is easier to form. It is also easy to maintain a critical stance on advantages and disadvantages of computational abilities, when they are presented through such a general framework, as the design of spatial configurations. The logic of the computer is then also easily contrasted with the human intelligence.

It should be noted however that this thesis research is a study of floor plan design from a specific angle. It perceives a floor plan as an aggregate of closed perimeter cells. What each cell in this kind of a floor plan representation symbolises can have different interpretations, a cell outline does not always have to symbolize walls. It can merely symbolize an area perimeter, for example empty space. This kind of a floor plan representation is highly simplified however it is highly attainable method to be used in computational floor plan generation.

As a simplistic representation of floor plans its architectural merits are controversial. As a mere collage of rectangles it ignores much of the contemporary architectural

014 Derix, ‘The Space of People in Computation.’, 15.
discourse such as the flow between spaces. The articulation of space and spatial qualities are qualities that by far are hard if not impossible to translate into numerical forms. In engineering the task of optimization is more straight forwards as engineering design task involve more easily quantitative processes such as optimization for maximum (or minimum) sunlight or the form of a structure according to affecting forces.

However, precisely because of its simplistic representation of architecture, it is easy to see cell models being used by developers and engineers on more industrial type and scale of building projects. Majority of the constructed buildings in the world are not designed by architects. If architectural qualities could be ingrained at least partly into softwares like these there would be a more significant architectural impact on the overall design of buildings. For the urbanizing world, computational, simplified approaches of architecture could help to better the situation of unplanned urban agglomerations and informal buildings.
Research methodology

“Computer technology is revolutionizing the way that architectural design is done, but the theoretical presuppositions underlying computer-aided architectural design systems are rarely made explicit — and when they are, they often turn out to be shaky and inconsistent.”

- William J. Mitchell

The sphere of computer sciences, from which applications for design computation are derived, is overwhelmingly vast. It encompasses a wide body of categories, such as methods, approaches, programs, strategies, to name a few that have been implemented into architectural design practices. If one is interested in elucidating all the computational processes that can be used in floor plan generation, the body of research is too large to be covered within a single study. Furthermore, there is not much sense in doing that either, since knowledge derived from such a vast study would not be significant in relation to the amount of information to be processed.

As an analogy, a driver does not need to understand the mechanical constructs of the car, or a smartphone user does not need to know how to repair the circuit boards of it. Excessive knowledge can help, however it is not required, to take advantage of a given innovation, whether it is a car, a smartphone or a computer algorithm. What this means, is that the methodology of this...
research is constructed so that the root phenomena of the given subject are ostensibly explained to avoid detrimental confusions, and then one area is more thoroughly studied. This logic is repeated in all the chapters until the connection between theoretical ideas of computation and practical workflow of floor plan generation is established. In general, the main defining characteristics here are the decisions to study floor plan generation, evolutionary algorithms and later on, one specific evolutionary algorithm implemented in Galapagos within Grasshopper® to carry out an exemplar design process. The reasons to study evolutionary algorithms is detailed later.

Thus, the logic of acquiring information and the method to carry out research in this thesis can be described as a consecutive set of funnels, meaning that general information of a given subject is first introduced, and then only a specific branch or subgenre is further elaborated for specific purposes.
Research structure

“The third class is composed of the ‘middling’ problems, those that cannot be tackled using symbolic logic — they contain too many elements — or statistical methods — they contain too few. To this third class belongs the design of buildings, and that’s where the fun with information technology starts. “

- Ludger Hovestadt

Each chapter in this thesis corresponds to a segment or a part of the overall process of generating a floor plan. As mentioned earlier, the research question for this thesis is: How to construct a computational model to generate floor plans? This essentially translates into a question of how information or data can be transformed into geometry, and later into an architectural floor plan. This procedure in turn reveals at least five modules of procedures that are responsible for the transformation process. Then, and finally, to fully comprehend these processes, certain background knowledge needs to be presented first. Figure 01 on page 32 illustrates this logic of forming the research structure more thoroughly. In general, the structure is sixfold and presented in the following.

The first part proceeds to clarify much of the confusion surrounding computational design in architecture. The aim is to dismantle the misconceptions about the processes of computing. The

chapter thus presents the general methods and design procedures of computing that were originally developed in computer sciences and mathematics. In this manner, an objective view of computational design can start to emerge. In the first chapter, a general outlook on design approaches that advanced into computing are also presented to show that computation serves a certain formal methodology of design, and not merely intrinsic value of itself, that computer usage is not necessarily instrumental for the computational methodology however it is instrumental for the practical implementation of such methodology.

The second part continues on clarifying issues, as the general history of computing in architecture is unfolded. This is necessary in order to place the issue of floor plan generation into the general context of computational design in architecture, and to understand the terminology in general. Also, there are various computational projects and research that overlap with the issues concerning floor plan generation.

The third part consists of a review of significant precedent studies and introduces the syntactic design methodologies in computing. It elaborates on the construction of the geometrical representation and the logic used to compute desired configurations of geometry.

The fourth part then presents the procedures, or approaches of computing that are used to operate the geometrical construction, i.e. how to transform or convert the geometries into a well formed design. This, in detail, translates into the study of metaheuristic algorithms and the logic of fitness function among others. Evolutionary algorithms are studied in detail as they constitute the majority of metaheuristics used in architectural computation. Design methodologies influenced by evolution appears to be significant field of study in the contemporary architectural research.

The fifth part presents a design experiment of floor plan generation and as such it is considered to represent the solutions component of the research area in figure 01. The experiment is carried out in order to produce knowledge to support the literature review, and to provide practical information about a computational

017 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 11.
Figure 01.1: Derivation of the research structure.
design approach.

Finally, the sixth chapter contemplates on the overall issues and ideas of architectural computation as a design methodology. It compares general design methods to the computational methods and proceeds to form a synthesis between them.

This kind of a structure could be criticised of being too far reaching to encompass all relevant issues. First of all, this research is not meant to be a comprehensive opus for the syntactic or structuralist approach in architectural design computing, but rather all one needs to know about harnessing computation as a tool and beginning to implement its virtues into floor plan design. Thus, the research overlaps with many different ideas and concepts in order to fully elucidate the harnessing of computation into design processes.

Figure 01.2: The focus areas.
I. Systems, models and procedures

“... be it a drawing, ... or a computer program. ... no analogy, analogue, mapping, model or simulation can ever present a one-to-one relationship, complete in every detail, with what is being represented. There would not be much point in setting it up if it could.”

- Geoffrey Broadbent

The most important purpose of this chapter is to simply clarify issues and form a base for the research to be build upon. It starts with the overview of the digital scene of architecture and continues with the alleged analog or natural computing that rose as a design methodology without the use of a digital computer. Obviously, these two methodologies have many similarities.

The digital scene of architecture has its own jargon. At least, it is how it presents itself before one has delved on the subject for a while, and finds oneself using the same abstract terminology, until realising that there are some nuances yet to be understood. Thus, the overall terms, and misconceptions relating to them, are first described in this chapter. These include issues, such as the one of computer aided and computing, algorithmic and parametric, scripting and coding, and optimising and simulating. A more technical glossary of terms specific to this research was presented in

the introduction in the previous chapter.

The latter part of the chapter then presents the computational approaches of design without the use of computers. Researchers constructed physical scale models, much like people construct digital 3D models these days, that were to be used as design aids or generators of design. Probably the best exemplifying description is provided by Antonio Gaudi’s wire systems, where weights were hung to create inverted structural studies for church designs. In general, what designers like Gaudi were doing was that they took advantage of the natural physics, and harnessed them into their models to compute designs. I.e. they were influenced by the analogy of laws of physics as computational processes taking place in natural world. The forces in Gaudi’s wire systems acted as such computational processes and thus it is obvious why computers were harnessed later by this kind of design methodology.

Thus as computing can be seen to happen in digital and natural spheres, the ontology of these constructs must be analysed. This will prove to be important when the digital computing is further elaborated throughout this research.
Indeterminate computing

“‘Curiously, while historians and theorists have gravitated from canonic modernism towards mid-century or postwar modernist positions and have been acute in their analysis of the impact of media, publications, television, film and advertising on design and construction, a similar study of digital technology has been very clearly avoided.”

- Greg Lynn\(^\text{002}\)

A technological actor — the computer — has been described as “an inanimate actor who takes different forms and names: machine, computer, manual, software, code, script, etc.”\(^\text{003}\) This abstract definition is descriptive of the overwhelming ambiguity of terminology that plagues the publications and projects of the digital scene of architecture. Many of the descriptive terms seem to denote same properties and actions. However the mere existence of them is implying that there should be a difference in their nuances. Since all approaches and tools in digital scene of architecture are derived from the computer sciences, there are no industry standards as it is when using common architectural software such as AutoCAD, ArchiCAD or Revit. Many architects might describe one and the same computational design process with differing terminology. For example, David Rutten has listed categories or titles that have been used to describe the well known Grasshopper

\(^{002}\) Lynn, ‘Archaeology of the Digital.’, 11.

\(^{003}\) Ibid, 5.
software: *Node-based design, logical modelling, visual programming, programmatic modelling* and even *spaghetti wiring* to name a few. This is understandable, since computing in architectural profession is more or less unprofessional when compared to the pure computer sciences. Yet when proceeding to educate others or sharing information, the loose terminology serves only the few and confuses the rest. Also, it implies that there is one absolute method or a whole that constitutes computing. However, the evolutionary lineage of significant computer programming languages, portrayed in the [figure 02], shows that there are multiple ways to perform computations with computers.

The confusion continues at even more general level of discussions when terms, such as *parametric, algorithmic* and *scripting* are used interchangeably. It further proves that little is known of the actual structures of these processes run in computers. This vagueness might also be why there is much misunderstanding of how computerized designs proceed in other architectural spheres outside the digital. If one is to make sense of the floor plan generation, first thing to address is the general misconceptions relating to it. That is why this chapter will serve as an extended glossary in order to clarify misconceptions and vagueness in terminology of computing in architecture.

---

Figure 02: The development lineages of major programming languages.
Computer Aided vs. Computational

“To understand computation, and its relevance to architectural design, one must understand the distinction between computation and computerisation. In principle, this can be broken down as methods which either deduce results from values or set of values, or simply compile or associate given values or set of values. One increases the amount of specificity of information, while the other only contains as much information as is initially supplied.”

- Sean Ahlquist & Achim Menges

The definitions of *computational design* and *computer aided design* have had varying and overlapping meanings. In contemporary language though, the computer aided design refers mainly to the representation and production of the geometry of the final design, and is therefore defined formally as to encapsulate data into symbolic representation. Computer aided design is usually carried out with the standard conventional software sold as a ready-made product. The sole purpose of such software is therefore purely to increase the efficiency of design and drafting workflow or to

---

006 Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking, 10.
007 DİNO, ‘Creative Design Exploration By Parametric Generative Systems In Architecture’, 211.
008 Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking, 11.
automate certain parts of it.\textsuperscript{009}

Computation however is a more broader term as it is referring to “the representation and use of knowledge to support or carry the synthesis of designs”.\textsuperscript{010} It can therefore be defined formally as producing specific data which is realised out of an abstraction.\textsuperscript{011} Its motivation is to give rise to novel design approaches in which computation is regarded as an actor in the design process to explore design solutions.\textsuperscript{012} In this regard, this thesis is mainly concerned with computational design and not as much with computer aided design.

The subjugation of computing in general can be derived from the tendency of architectural practise in its short-sightedness for using the computer for mere automation of workflow, or for representational purposes during different design phases. Alas, computers are not used to develop or enhance significant architectural aspects and qualities in design.\textsuperscript{013}

Software that intertwines different scales, parameters, and stakeholders are increasingly seen to correspond what computers can only do for designers. Such examples of these software approaches include the life cycle design and building information modelling (BIM). These approaches are projected only from the point of view of management rather than the initial design. This kind of outlook has further strengthened the requirem of automation of design mechanism and phases, and thus subjugated computing into the status of a tool rather than a design methodology. Computation is taught in the form of technical courses on how to use a given software, and not on how to philosophize as a methodology.\textsuperscript{014}

\textsuperscript{009} DİNO, ‘Creative Design Exploration By Parametric Generative Systems In Architecture’, 207.
\textsuperscript{010} Ibid, 207.
\textsuperscript{011} Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking, 11.
\textsuperscript{012} DİNO, ‘Creative Design Exploration By Parametric Generative Systems In Architecture’, 207.
\textsuperscript{013} Derix, ‘Mediating Spatial Phenomena through Computational Heuristics.’, 61.
\textsuperscript{014} Ibid, 61.
Algorithm

“Not only does one still come across ardent critics who perceive as sullying the computer’s incursion into a world of practice uncontaminated for centuries by reprographic machinery of any kind, but there are several levels of nostalgia-based discontent over the choice of tools with which to inscribe our thoughts.”

- Mark Burry

The name algorithm is originally derived from Arabic, and the usage of algorithms dates back to the 7th century India and 9th century Persia. Formally defined, an algorithm is a set of repeatable unambiguous instructions that when confronting identical problems they reliably produce the same answer. More ambiguous definition for algorithm would simply describe it as a set of instructions for solving a given problem. The basic logic of an algorithmic construct can be seen in informal examples, such as furniture assembly instructions, cooking recipes or origami folding steps. Figure 03 on the next page is provided to illustrate this.

017 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 1.
020 Kalay, Architecture’s New Media: Principles, Theories, and Methods of
First, operations - here, the actual folding - needs to be defined and described, written or illustrated. Second, all the operations must be listed in a concise and correct manner. Thus, constructing algorithms requires the designer to possess expertise in the problem at hand, but also in laying out the instructions in an effective way.

Figure 03: The instructions for origami folding can be seen analogous to an algorithm.

In digital computation however, the list of instructions must be truly unambiguous as the computer in most cases is incapable of interpretation since all processes in them are deterministic — they can not handle any chance or randomness.

Computer-Aided Design, 47.
022 Ibid, 47.
023 Jackson, Folding Techniques for Designers from Sheet to Form, 118.
024 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY
Thus, in computing, an algorithm is defined as a linear sequence of operations, a finite set of exact step-by-step rules for operations to be carried out that seek to realize a solution in a finite number of steps. In general, an algorithm incorporates boolean (true or false) operators as gates to inhibit the sequential process. This in turn means that it can be actually a loop structure as the sequence will be returned sometimes to an earlier phase of the it. This loop will keep on going until all certain criterias are met and the algorithm is allowed to terminate. Figure 04 is provided to explain the above mentioned process. An algorithm thus processes a value or a set of values in order to produce a different value or a set of values i.e. it requires certain inputs to transform them into certain output. An algorithm can accept a variety of data, but usually it involves data constructing geometric forms, design variables, data structures, and mathematical and logical operations.

The scale and complexity of an algorithm can blur the boundary between it and the computer program we have become familiar today: Algorithm can also refer to the textual signs of a computer code. When an algorithm has a complex structure and a myriad set of boolean operators, some of them user controlled, the whole linear sequence is blurred, and the algorithm becomes an active interplay between itself and the user. In this light it is more descriptive to characterise an algorithm as a program.

WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 2.
027 Broadbent, Design in Architecture: Architecture and the Human Sciences, 305.
Figure 04: A generalized example of an algorithmic sequence.

In this manner, algorithmic processes producing specific outputs from specific inputs can be seen analogous to constructing from architectural working drawings. The history of computer science is largely a continuum of developing methods — similar to architectural methods — to compose “large instruction sets tractable, reliable, efficient, and effective”.

Algorithms have been categorized or characterised according to their properties and abilities, however the set of categories is infinite and informal and many algorithms can belong to many categories. Technically speaking, every digital algorithm operates through a fixed set of procedures and it is not capable of diverting from that, thus it does not allow any random quirks while it operates. This is why characteristics of heuristicity or randomness are only superficially or pseudo descriptive. The characteristic categories imply nothing of the quality of the algorithm, rather they should be preserved from the point of view that different kinds of algorithms have different kinds of usages or ways to handle a problem. Situations, such as the amount of data involved in the problem, the state of data sorting, that is whether it is all random or somewhat readable, allowed usage of computer memory or whether the algorithm is capable of producing incomplete results, are indicative of the algorithms properties.

The following describes some categories that are descriptive in the light of this research.

Most of the algorithms used today are specific. They are constructed only for solving individual problems. Thus they are usually fast and reliable. Approximate algorithms, that are sometimes described as heuristic algorithms, can find a reasonable, or at least some sort of a solution for a given problem. Because of its loosely defined rules it has no guarantee the solution it finds is of any high quality. Algorithms that allow external entities, that is humans or other algorithms, to affect their proceedings are called open. Some algorithms include abilities that allow them to produce solutions that are somewhat unpredictable. They are called stochastic algorithms.

032 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 2.
034 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 2.
Scripting

“Scripting, as opposed to its more hard-core sibling computer program writing, has been around for as long as there have been computer users ...”

- Mark Burry

Scripting, as many other terms in computing, seems to lack a clear definition. Although it refers to computer programming, scripting can denote different levels of it. Low level scripting can be seen as the ability to customise and configure software around user’s own likings, while more demanding scripting involves combining preconfigured algorithms i.e. certain libraries of functions in an open-source spirit. Thus, scripting includes more extensive engagement with the computer than the intended typical usage, making the workflow more efficient. To make things a bit more confusing, in relation to computer languages, scripting languages often translate to programming languages. Further, if scripting is considered as “the means by which the user gives highly specific instructions to the computer”, it starts to resemble the definition of applying algorithms.

In general scripting however, refers to lower-level

WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 2.

036 Ibid, 8.
037 Ibid, 9.
038 Ibid, 9.
computer programming\textsuperscript{039}, and it is used to indicate simple automation tasks and is reserved to describe non-professional programming. Otherwise, it can be used to describe programming by people who are not formally qualified as computer scientists. A person who is considered to perform scripting is not in the business of designing large programs or operating systems\textsuperscript{040}, rather he or she is automating more or less routine procedures\textsuperscript{041}.

In conclusion, scripting can allow the designer to spend more time on design thinking by making the workflow more efficient. Customisation of the ready made software allows the designer to bypass the restrictions of the available software and in this manner, the designer can develop approaches not thought of by the original developers of the software. through scripting the designer can become a toolmaker or the software engineer\textsuperscript{042}.

\textsuperscript{039} Ibid, 8.
\textsuperscript{040} Coates, The Architecture of Programming, 183.
\textsuperscript{041} Burry, Scripting Cultures: Architectural Design and Programming, 8.
\textsuperscript{042} Ibid, 9.
The definition of parametric has caused the most turmoil within the realm of computation in architecture. The term parametric originates from the 1830s mathematics when mathematicians started to use it for describing different geometric representations. When architects later acquired tools that started to resemble these mathematical representations, termed parametric, the definition was loosely applied. In contemporary architectural language the term parametric can refer to the design parameters such as budget, site, material properties found in every project, the design method, the tools or software used or even the style or movement in the arts, namely parametricism.

The history of parametric design can be seen to have its roots in generative design, a method that emphasizes the logic of

---

044 Ibid, 19.
048 DİNO, ‘Creative Design Exploration By Parametric Generative Systems In
design production processes over a single design solution produced by it. Origins of it can be traced all the way back to the design systems used by Aristotle, and later in architecture by Leonardo da Vinci and J. N. L. Durad Figure 05 on page 51 present the schema used by Da Vinci and figure 06 on page 52 the schema used by Durad. Other significant architectural precedents are Louis Sullivan’s plates describing the processes for reproducing floral ornamentation, and even Le Corbusier’s famous Five Points of Architecture can be seen as a generative design schema.

An abstract way to define a generative system is that it is essentially combinatorial and meant to operate to produce a variety of potential design solutions. The word system, however, can be seen misleading since a generative system can be used to describe the workflow or the process of design work itself, rather than the construction of an actual physical model. A familiar example of a generative design system in architectural design could be the production of multiple scale models through a set of same design operations, but each with a varying emphasis of different operations. Once all the model are ready, that is they have been produced by the design operations, they can be compared. With the help of comparison the positive and negative properties of different models become more evident.

At first, it would seem that parametric design can be described merely as a digital genre of generative design. However, certain shortcomings of this rendition need to be clarified in order to better understand why such differing titles have emerged.

Architecture’, 208.

053 Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking. 152.
Figure 05: Given elements in rows are combined (processed with an algorithm) with elements in columns to generate geometry.

Figure 06. Generative studies on combinations of geometric elements.
A parametric model consists of equations,\textsuperscript{057} and other kind of operators, that process the input variables. Thus by varying the input values the model produce different results.\textsuperscript{058} Unlike in generative design, designer in parametric design is able to modify the system producing designs in greater detail. For example a physical tool used to produce certain things might be too robust for its designer to quickly alter and modify. However, as parametric designs are constructed in digital environment, the functions remain explicit and they are easily modified. Therefore it is not the presence of parameters — which could be argued to be present in every kind of design\textsuperscript{059} — but rather the explicit relationships of parameters and functions, and the easiness to modify them, that separates parametric design from generative design. Therefore, in contemporary language, parametric is meant to emphasize the ability of easily changing the once generated design.\textsuperscript{060}

To complicate matters, it has been argued that parametric models are inherently algorithmic and thus form a subcategory of algorithmic design. The general idea supporting this argument is, that since a parametric model executes specific operations on its inputs, the parameters, to produce an output, principal processes are then quintessentially identical in the sequences and constructs of an algorithm.\textsuperscript{061} I.e. every parametric relation, as described earlier, is essentially formed through the use of functions which define the relationship of certain input and output values and thus comply with the definition of an algorithm. The actual computing taking place within an algorithmic or a parametric model is the same: the elemental components found in a parametric model form the algorithm — which is sometimes called the schema or definition — itself.\textsuperscript{062}

It seems that the only difference between an algorithm and a parametric model lies in the visual manifestation of each.

\textsuperscript{057} Burry, Scripting Cultures: Architectural Design and Programming, 258.
\textsuperscript{058} Coates, The Architecture of Programming, 180.
\textsuperscript{059} Davis, 'Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture.' 24.
\textsuperscript{060} Ibid, 24.
\textsuperscript{062} Ibid, 210.
Traditionally, algorithms manifest in written form, that is they are written in some programming language. Parametric models are not that uniform, however they all emphasize the explicit ability to directly manipulate inputs, parameter values, in order to produce quick changes in the form of the given design. Yet, this difference is highly dependant on the software used. There exists a variety of programming languages (see figure 02 on page 39) that can be used to write algorithms, however in the recent history, the graphical editors of algorithms have emerged. Ironically, the epitomized software of parametric design, Grasshopper®, is defined as such graphical algorithm editor. Other contemporary programming languages and environments incorporate ready made, open-source components that can be combined without thorough knowledge of the given language (remember the characterization of scripting). Such is the case of Processing® where one can import different libraries (made by other designers) into Processing® sketches (i.e., schema or design).

All in all, the difference which allows the categorization of something as algorithmic or parametric is ambiguous. And if the difference only manifests in the way values are allowed to be altered by the user during the design process, then it tells something about the differences of specific softwares and not the differences of design methodologies. In a sense, parametric design has a more narrow definition and relates more to computer aided design than algorithmic design which is purely computing. In this sense the distinction between algorithmic and parametric design is done based on the looks of the software. This is described in the following.

Parametric design softwares usually incorporate multiple viewports — the 3D view to display the geometry of the model, and the actual editor to create the algorithm — to make the design process more tangible, whereas the traditional textual editors used to edit algorithms per se do not. (It should be noted here that this is also the reason why BIM is sometimes considered as a “fridge
case” of parametric modelling.) However, one can create software environments characteristic to parametric softwares with the textual editors. I.e. the traditional textual editors of algorithms have the same potential as the ready made parametric modelling software. One has to construct them by oneself. For example in Processing, one can incorporate graphical sliders, buttons other interface elements into the viewport.

The algorithm editor in a parametric software is either textual or visual. I.e. the algorithm is constructed in terms of an actual graph data structure (see the figure 30 representing the general process of an algorithm on page 45). In the case of the graph, the nodes represent components that produce a certain transformation of the data, and the paths connecting them form a one-directional flow of data. The separation of the 3D output and the algorithm editor allows the editing of data flow that is not manifesting itself strictly in the geometry. This is particularly useful when non-spatial contextual parameters can be processed with the software and only later used to generate the actual geometry. This simultaneous view of output and the editor might have also been a key issue why parametric software are described as intuitive and easily taken up.

068 Ibid, 216.
069 Ibid, 216.
Computerized feasibility analyses

“... the art of modelling being based in large part on the ability to tell significant from the insignificant.”

- Manuel DeLanda

There exists ways to describe the manner in which different computational processes are operated. It is common to read descriptions of how a model is created to simulate something or that a form is being optimized by using a certain procedure. It is informative to understand what such descriptions of simulation and optimization denote since much like the terms parametric and algorithmic they too can be easily used interchangeably.

In general the terms used to describe the operation of some digital model are essentially describing the analyses process that are derived from feasibility analyses. There exists ways in which a given design can be analyzed, i.e. combinations of performance properties, that is a design, can be ranked according to some predefined criteria. Operational research (see history chapter on page 75) has been highly influential for the development of feasibility analysis.

Traditionally feasibility analyses have relied on designer's experience and professional intuition. However today there exists many kinds of software that are described to carry out these

070 Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking, 145.
operations.\textsuperscript{072} They can be used to morph geometry in accordance with demands, run a form finding simulation or optimize forms and shapes. The appliance of feasibility studies however require that certain decisions in design have already been fixed and merely the overall interaction of them is examined. These analytical operations are then not capable of generating any novel designs, alas they are merely enhancing an already existing design.

In general feasibility analyses are commonly categorized as \textit{morphological analysis, optimization and simulation}.\textsuperscript{073} Morphological analysis denotes the process of identifying either feasible or infeasible parts which are resulting in a whole. In general, it consists of examining pairwise compatibilities of all combinations of parts and finding the best solution to form a whole.\textsuperscript{074} Optimization in contrast is the prioritizing of solutions, and it therefore requires criteria by which solutions can be ranked accordingly.\textsuperscript{075} Simulation is helpful when the comparison of different solutions is difficult to establish and one can not separate goals and constraints. In simulation the interaction of all constraints and goals are examined at once.\textsuperscript{076}

Despite their analytical origins, feasibility analyses and the terminology attached to them are used to describe the operations of computational processes.

\textsuperscript{072} Ibid, 386.
\textsuperscript{074} Ibid, 218.
\textsuperscript{075} Ibid, 218.
\textsuperscript{076} Ibid, 218.
Before the digital was the digital\(^{077}\)

“... we were working in a way which we now know the computer to do, today. We weren’t conscious we were doing that.”

- Peter Eisenman\(^{078}\)

Architects in the pre-digital era were already exploring processes in design that computers would later merely enhance and automate. In other words, work produced by some architects were “pre-conscious of the computer.”\(^{079}\) This kind of a working method is also sometimes called \textit{natural computing} as it refers to the myriad computations that can be seen to take place in natural processes.\(^{080}\) As implied earlier whether one talks about generative design, parametric design or algorithms in general, the history for such tools greatly predates the digital era. In fact, generative logic and systems are present throughout the history of philosophy, literature, music, and let alone architecture.\(^{081}\) Greg Lynn has even asserted that Peter Eisenman actually invented Parametricism\(^{082}\) even though Eisenman himself continues to denounce the merits of computer usage.\(^{083}\)

Complex and fluid architecture has existed before the

\(^{077}\) Lynn, ‘Archaeology of the Digital.’, 49.
\(^{078}\) Lynn, ‘Archaeology of the Digital.’, 56.
\(^{079}\) Lynn, ‘Archaeology of the Digital.’, 54.
\(^{080}\) Coates, Programming.architecture, 164.
\(^{082}\) Lynn, ‘Archaeology of the Digital.’, 55.
\(^{083}\) Ibid, 54.
digital advent. Such architecture is demonstrated by Frei Otto, Pier Luigi Nervi, Felix Candela and Antonio Gaudi, to name a few.\textsuperscript{084}

As stated throughout this thesis, much of the irrational dislike relating to the use of computers in architecture is result from poor understanding of the processes of computational design. Thus it is vital for the process of dismantling these attitudes to understand that the logic applied within computers is not inherent to the digital or the computer: they have been present in architecture long before. Although the capabilities of computers, namely the processing efficiency and tirelessness,\textsuperscript{085} have revolutionized the generative design methods and the tool making and thus risen into a specific field of its own. This chapter thus presents some architectural design approaches that can be considered as precedents from the pre-digital era.\textsuperscript{086}

As stated previously, a common misconception is that computers are solely responsible for the appearance of complex and fluid architecture. This is easily refuted as one glances across the history of significant buildings. Yet in their time of appearance, their design and construction resulted in great controversy. However, the designer managed to develop techniques to overcome the difficulties.\textsuperscript{087}

Since computers were not readily available, designers had to resort to physical models instead of digital to carry out the simulations. This kind of process was an exhaustive and inefficient method to carry out design. Calculations and findings based on scale models had to undergo difficult transitions before they could be imported into the actual scale and materiality. This process relied completely on the expert skills of the designers, as a formal method for this was absent, namely the contemporary computerised simulations.\textsuperscript{088}

It is thus easy to see how these kind of design approaches have

\begin{flushleft}
\textsuperscript{085} Kalay, Architecture’s New Media: Principles, Theories, and Methods of Computer-Aided Design, 78.
\textsuperscript{087} Ibid, 210.
\textsuperscript{088} Ibid, 210.
\end{flushleft}
merited significantly from computational approaches. The design of complex geometry in particular can be observed to take advantage of parametric abilities and feasibility analyses provided by computers. In short, computer diminished the need to construct non malleable physical scale models and sped up the trial and error process, thus making the overall process more efficient, precise and detailed.\textsuperscript{089}

As stated earlier generative logic has been present in architectural design throughout the history. However the above mentioned need for computation can be exemplified by the design methodology, sometimes referred to as \textit{inductive morphological system}\textsuperscript{090} through the works of Antonio Gaudi and Frei Otto. Gaudi used upside-down catenary models in order to calculate the angles and vaults for the sloping columns of the Sagrada Familia.\textsuperscript{091} Otto used natural materials, such as strings, glue, water, and soap in combination with gravity and other forces to design objects that became optimally shaped while exposed to natural forces modifying them according to the properties of each material.

A more recent and complex example asserting clearly that computers are not necessary, is provided by Peter Eisenman, who in 1967 despite the existence of computers, used analogous transformational rules for architectural design synthesis. Eisenman's concept was a system, often described as analogous to a language, that generated an infinite number of expressions out of finite means. As such Eisenmanns methodology is highly influenced by theory developed by Noam Chomsky in linguistic. Chosky and the theories he developed are described in detail in the third chapter. Eisenman illustrated this design concept with series of houses, namely Houses I-X. Eisenman described the houses as being not "an object in the traditional sense - that is the end result of a process - but more accurately a record of a process".\textsuperscript{092} Despite the fact that Eisenman's design included various descriptive illustrations, it remained loosely algorithmic\textsuperscript{093}, however it is clear how the

\textsuperscript{089} Ibid, 210.
\textsuperscript{090} Coates, Programming.architecture, 164.
\textsuperscript{091} Ibid, 164.
\textsuperscript{092} DİNO, ‘Creative Design Exploration By Parametric Generative Systems In Architecture’, 209.
\textsuperscript{093} Coates, Programming.architecture, 113.
logic was similar to the later digital counterparts of algorithmic design.

What these designers and design methodologies had in common was that they were hermeneutically or heuristically applying processes of computing — whether taking advance of the ones seen to take place in the natural realm or as generative record of logical process — into design generation. As stated earlier, these kind of generative processes are sometimes referred to as inductive design methods. Induction in this relation is defined as “the inference of a general law from particular instances” Thus they can be described as emphasising some specific process or method that becomes the new environment for the design question to inhabit. The later emerging design solution is justified as it comes from the environment of the new inductive model. The design question is thus substituted with a simpler one, and used as a design criteria for the end result, the final design.

Obviously the above mentioned logic for design gained criticism. First, the critics asserted that the inductivist methodologies could never be certain that adequate data was at hand to produce a synthesis for the design method in the first place. Second, that the creativity in design was simplified by the use of the

095 Coates, Programming.architecture, 164.
096 Apple dictionary Version 2.2.1
097 Coates, Programming.architecture, 164.
analogous processes for it to be viewed systemically, and whether the final design was able to meet the standards of the initial design question. Computers, at the time, were seen to solve these two main points of criticism as they incorporated advanced processing capabilities.

The computer was seen to be efficient and thorough in creating a variety of design solutions while maintaining objectivity. This reasoning lead to the idea that with the help of computers the processes in inductive methodologies would not have to be simplified anymore, since the computers could deliver an enhanced version of the simulation. This line of thinking advanced the implementation of computers and can be seen as an early phase of the research leading up the development of evolutionary algorithms. It also allowed the generative methodology to take a leap from the analog world to the digital world of computing. In this manner, the systems or models found in both worlds share the same methodologies and categorizations. They both proceed with certain limitations and simplifications of the reality. Therefore before commencing on the study of the history of computing, the ontology of models is described.

---

098 Ibid, 163.
099 Coates, Programming.architecture, 164.
Ontology of models

“Platonicity is what makes us think we understand more than we actually do. ... Models and constructions, these intellectual maps of reality, are not always wrong; they are wrong only in some specific applications. “

- Nassim Nicholas Taleb

The real life phenomena are overtly complex by nature and therefore cannot be replicated into a model, as that work would be time consuming, inefficient and evidently impossible. Rather, one has to reflect the model to objects or phenomena that have resemblance to it. Therefore, a model becomes a complex analogy defined by the designer in order to describe the structure of a given phenomena. Acknowledging the fact that human brain is capable of seeing patterns when there is none, there is always the risk that the model is representing something that was not there in the first place.

Thus it can be said that the model is set out to distort the structure of the phenomena it was originally supposed to clarify.

The data structures of a computational model can also become

---

100 Taleb, The Black Swan: The Impact of the Highly Improbable, xxx.
102 Kahneman, Thinking, Fast and Slow, 115.
vast in no time. It has been estimated that a model representing practical building design could “easily consist of many hundreds of thousands, or even millions of variables”\textsuperscript{104}. This fact alone steers the designer using computer to construct a highly superficial model.

Then again that is the advantage of creating models. There would not be any sense in designing a model that would be as complex as the real thing in itself since the underlying logic would then also be as complex. For our purposes here, if we were to design a model that would generate a spatial layout, we would need to simplify the design process taking place in human mind. Even if we ignore the aesthetic cultural traits engraved in our architectural profession, we would still be left with a series of design decisions with feedback loops. That is we have professional habits and routines that require us to accompany some design decisions with others. The reason to design models lies in the fact that gaining understanding of given phenomena can be greatly enhanced through analogies — and thus with the help of models — but in the end they are only applicable to some cases and utterly deceiving in some.\textsuperscript{105} Combining models would only blur the insight gained from a simple model and a sum of combined models would lead to a common denominator.\textsuperscript{106} This would also create a new scale of subjectivity, as the designer chooses which phenomena are combined and allowed into the model, and what kind of a relationship these two models have, ie. is the other in a main role dominating the other, or are they equal? How the designer defines this relationship?

\textsuperscript{104} Mitchell, Computer-Aided Architectural Design, 42.
\textsuperscript{105} Broadbent, Design in Architecture: Architecture and the Human Sciences, 88.
\textsuperscript{106} Ibid, 89.
Classification of models

“If scientists had to build models that captured all scales simultaneously no scientific field would ever have succeeded in explaining anything. We would be trapped in a block universe in which every aspect is inextricably related to every other aspect ...”

- Manuel DeLanda

There are multiple ways to define and categorize models, digital and physical. One of the earliest and simplest categorizing methods groups models into three categories: analogous, iconic and symbolic depending on how they generate information and thus designs. For the purposes of this research the symbolic models are of most concern since the computing processes are symbolic by their nature.

Analogous systems are constructed by representing original properties with entirely different ones. A simple example would be a site plan where different lines represent buildings and roads, hatches certain areas and contour lines the terrain. Another common analogous model is a graph since it can represent time and all kinds of numbers. Analogous representation, on the other hand, could also include mechanical parts such as wheels, dials and

107 DeLanda, Philosophy and Simulation: The Emergence of Synthetic Reason, 14.
sliding columns.\textsuperscript{110} A profound example of analogous model is again Antonio Gaudi’s hung wire frame system of weights to construct different vaulting systems for the design of Sagrada Familia and the Guell Chapel.\textsuperscript{111}

Iconic models have the same characteristics as the object or phenomena it seeks to represent.\textsuperscript{112} An architectural scale model is an example of an iconic model. Iconic models only show particular potential states of the design.\textsuperscript{113} Operations used in iconic models include addition and deletion, rotation, scaling of elements to name a few. Iconic models could also be a combination of the above mentioned operations, and thus another example could be a series of different facade tilings on a fixed envelope, or configurations of a given spatial program.

Again the model cannot have all the properties of the original, like in the scale model example the model might be the same shape but made out of different material and in different accuracy of detail. In this way it might be more appealing and interesting than if it had the corresponding materials of the actual building. In this manner the designer has decided to represent certain properties according to the purpose of the model.\textsuperscript{114}

Symbolic models represent potential designs by taking advantage of words, numbers, symbols and mathematics and are therefore the basis for computing.\textsuperscript{115} Symbolic models consist of data structures that are infested with design variables.\textsuperscript{116} A design variable is thought of as a named container in which different values can be assigned. These variables can then allow the symbolically modelled data structure to represent different possible states of the model.\textsuperscript{117}

\begin{thebibliographystyle}{plain}
\begin{enumerate}
\item\textsuperscript{110} Mitchell, Computer-Aided Architectural Design, 39.
\item\textsuperscript{111} Ibid, 39.
\item\textsuperscript{112} Broadbent, Design in Architecture: Architecture and the Human Sciences, 89.
\item\textsuperscript{113} Ibid, 38.
\item\textsuperscript{114} Ibid, 38.
\item\textsuperscript{115} Broadbent, Design in Architecture: Architecture and the Human Sciences, 89.
\item\textsuperscript{116} Ibid, 90.
\item\textsuperscript{117} Mitchell, Computer-Aided Architectural Design, 40.
\end{enumerate}

66
Models for computer generated architecture

“Computation does not function with the explicit description of form of finality. Therefore, the designer is posited as the author of the rules as implicit descriptions for the development of form. ... It is therefore important that a critical approach towards computational design exacts knowledge from the historical and practical foundations of these interrelated field.”

- Achim Menges & Sean Ahlquist

In many cases architects and designers are blinded by the given software they are using, and define the whole design approach according to that specific program. It seems that the contemporary hegemony of Parametricism has epitomized parametric design tools at the top of the food chain despite the fact that there is much argumentation for describing it as merely a subcategory. Thus it might be argued that visual algorithm editors have defined the look of parametric design and the textual terminal look is signaling of scripting and algorithmic design.

In general, genres are only descriptive of the general characteristics of something that is to be explained. No method or process falls within single category. Usually a genre is descriptive of which properties are emphasized enough to epitomize something and which properties are subjected or nearly invisible. Such is also the case of computing tools. Many programs and softwares

118 Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking, 28.
have abilities that transgress the genre limits between parametric and algorithmic. The analysis of algorithms, scripts, parametric and simulation revoke at least three gradients which can be used to describe the emphasis of properties in software used in computational design.

Figure 08: The gradient between properties describing digital tools. All tools have differing emphasis on each property.

In order to avoid further confusion, from this point on in this research algorithmic, parametric or scripted architecture is referred to as computer generated architecture as a root term applicable for all these subcategories. When the term is not denoting the computational properties, but rather the scene or discourse, the term digital architecture is used.
The methodology of a given design approach, whether analog or digital, that is concerned of computing, is always taking advantage of certain constructs to generate designs. These constructs are interchangeably referred to as models or systems. For the sake of clarity, from now on in this research, these constructs are referred to as models since whatever the formal method of design methodology is, it is always describing a model of conduct.

Since different branches of design, (such as parametric design), can most times be described using other processes (with parametric it is algorithmic), the legitimacy of general computation in design should then first be presented before bolstering some specific approach or method of it.\textsuperscript{119}

Thus the history of computing is unfolded in the following chapter to complete the clarification of misconception in digital architecture.

II. History of computation in architecture

"'Invent' is a loaded word. Most young computational designers are ignorant of the history of algorithmic design and think they invent everything. If you are coding now, and you think you invented something amazing, I guarantee you someone at MIT or SIGGRAPH in the 60s, 70s & 80s has already done it”

- Nici Pisca, Gehry Technologies

Now that the major ambiguities and misconceptions have been clarified, one can more easily relate to the general environment of computer generated architecture. However, instead of proceeding straight to the analyses of precedents in floor plan generation, the general history of computing in architecture is first unfolded. This is done to further elaborate on the task of clarifying issues of computer generated architecture. It is presented to provide a general account of the past computational approaches and how the contemporary status of computing in architecture can be derived from them. Thus, this chapter will present an introductory account of the history of computational design in architecture. The purpose is to present those significant events and developments that are typically regarded as responsible for the development of computer generated architecture as it is known today.

As stated in the introduction earlier, there is much debate about terminology when describing computerized design

001 Burry, Scripting Cultures: Architectural Design and Programming, 54.
processes. The history chapter thus tries to clarify these issues by recalling when, and in what kind of a context, different branches of computational design approaches or methods in architecture have departed or intertwined. Such a case could be for instance the development path of computer aided design and computational design. Although these kinds of points in history are often notoriously retroactive, and thus malleable to interpretation, the general information however allows the reader to better distinguish between the contemporary approaches present in computer generated architecture. In this manner, the misconceptions about the algorithms and other computational processes could start to disintegrate.

As with all new design approaches, including the computational, one has to look for shifts in philosophical attitudes pervading the whole culture, specifically science and technology. This is why each paragraph is divided roughly into three parts that appear in every paragraph. First, each paragraph tries to prescribe the technological advancements during the given era. Second, relevant design theories to the development of computation are presented. The developments leading up to the contemporary understanding of computational architecture, such as the systems theory, cybernetics, syntactic and structuralist approaches are in focus.

Also interdisciplinary phenomenon with ties to computational architecture are presented in short. These theories have been influential for the development of certain computational methods and they are still touted in contemporary discourse.

It is important to form an understanding between technological and methodological statuses or conditions of a given time, and then to reflect that relationship into the contemporary one. In many cases, the other is vastly more advanced than the other, and thus most of the eras have had different dynamics. A clear example is the era of the 1960s when the possibilities of computing were greatly speculated, however the technology was primitive. In contrast, the contemporary era might be seen as a time of great computational abilities while the design methodology and theory are disorganized within the academy. As such it is quite the

---

opposite of the situation in the 1960s.

Third, since floor plan generation is the main computational issue of this thesis, important precedents from that field are placed accordingly within the history paragraphs. Although they are described in greater detail in the third chapter, it is relevant to see them on a time-line and to compare their development and abilities with the general discourse.

The developments of computer programming are touched upon. This should help the reader to understand the degree of vagueness when people are referring to the computational design conventions such as coding or scripting. The underlying theme of this thesis is to disclose the fact that there is no absolute way to practise computing. Choosing the tools — and thus programming environment — is a subjective choice that will have characteristic influence on the outcome.

Structurally, the history chapter is divided into nine parts. After the introduction, the four paragraphs are dealing with the four generations of computer technology. The latter four are organized according to the shifts in theoretical and methodological discourses, as the era of the fourth generation of computing is still considered to prevail. The current prediction of computers taking advantage of quantum mechanics will most likely define the new generation of computers, which in theory could perform computations, currently taking up to millions of years, in seconds.003

003 Wadhwa, ‘Quantum Computing Is about to Overturn Cybersecurity Balance of Power.’
Early history

“Architecture, as a practical form of art, has been in need of computation — and computational aids — since ancient times.”

- Yehuda E. Kalay

In early history, spanning from ancient times to the 18th century, the technological developments can be described in brief: primitive devices such as tokens in clay were replaced by more advanced devices, exhibiting clear signs of algorithmic properties, in accelerating speed, first by the abacus, then by the mechanical calculator, the steam powered machines and finally the vacuum-tube technology. However, the usage of these machines and the theory of how to operate them has been significantly more varied. During the time when agricultural societies started to emerge, devices able to make calculations and manipulate information — the computing devices — appeared. Some of these predate even the invention of writing by 3000 years. These devices in part made the construction of monumental architecture, such as the ziggurats in Mesopotamia, the Egyptian pyramids and temples of the Greeks and the Romans, possible.

During the Renaissance architects became obsessed with the search for the most appropriate geometric relationships in building and to do this, they developed new methods which can be

---

005 Ibid, 27.
interpreted to resemble algorithms. Examples of such algorithms among others include finding the square whose perimeter is equal to the perimeter of a circle, for calculating the dimensions of a space given its width, and to calculate the golden section. Tools the Renaissance architects used to perform these processes were mostly the compass and the straightedge.

Systematic use of algorithmic properties in design are first visible in Leonardo da Vinci’s design process of a church plan. Later in 1803 J.N.L. Durand illustrated how building elements can be used to generate plans and elevations of neo-classical architecture through the appliance of varying combinations of building elements. This systematic generation of design solutions was discussed earlier in greater detail in the second chapter on page 50. Figures 05 and 06 on pages 51 and 52 show the generative schemas used by these designers.

In 1822 new inventions paved the way for more advanced computers as the steam-powered computing machine was created to aid the production of arithmetic tables. It was the size of a locomotive and it received its input instructions in the form of a perforated card. Later in 1945, the first computer considered modern, a vacuum-tube computer the size of a large contemporary classroom, was built. The instructions for it were produced by rewiring the components of the whole machine in different combinations. These inventions did not result in significant adoption in architecture, rather they are viewed as transitional inventions leading to the technology that would later be of interest in architecture.

A paradigm of design called Operational Research emerged in the 1940s. It asserted that the best design procedure for a given task could be defined by solving a large number of conflicting operations. This kind of managerial approach was applied to

007 Ibid, 63.
010 Ibid, 27.
011 Ibid, 45.
architectural design, as it too was perceived as managerial all along. The computer at the time was regarded as a convenient tool to solve conflicting operations, and this idea survived through history as it matched the view vendors and large practices’ had of design.012
Starting from the 1940s, four generations of computer technology are visible. The next four chapters are derived from them.

012 Coates, Programming.architecture, 161.
Generation I 1946-1959: Tedious experiments

“The whole history of computing is a slow ascent from mind-bendingly tedious machine code, where every procedure must be written anew for a new machine, to portable languages like Java, which are designed to run on an imaginary computer.”

- Paul Coates

The first general-purpose electronic digital computer designed for commercial use, sized 17 by 17 meters, was of vacuum-tube construction that had to be stored inside an air-condition perimeter. It consumed excessive amounts of power, it was slow in computing, and it received instructions in machine language — a language specific to a machine consisting of alphabetic symbols and memory addresses — which was tedious and error prone to write. In the end, 45 machines were ever built.

Subsequent first generation computers were more great in numbers and more widely used. The variety of programming languages also started to increase, as it became evident that machine languages had proven impractical. These more advanced languages allowed programmers to write algebraic expressions for arithmetic operations. The following can be seen as a general overview of the most relevant ideas that started to affect the computational design

015 Ibid, 50.
and, through that, the general architectural theory.

Cybernetics, along with General Systems Theory, is a discipline that can be seen to originate in 1939, and which can be generalized as being part of a general critical shift in science, namely the Third Science. The earlier shifts are first the Rational science of the Greek — self-evident truths — and the second the Renaissance — observations and experiments. The shift to Third Science for the first time emphasized time and process, and how change and indeterminacy are fundamentals of reality. The founding of Cybernetics is often credited to Norbert Wiener.

A fundamental characteristic of cybernetics was the drawing of analogies between organisms and machines. A popular theme was the problem of how computers could be utilised to expand human intelligence. Cyberneticians were interested in certain feedback loops that were present in living bodies, such as the one where some organ senses a change in the environment and transmits a signal to the brains which in turn decides what to do, and further transmits a new signal to other organs for taking action. The above described process is illustrated in figure 09.

It should be noted that Cybernetics and General Systems Theory are disciplines encompassing wide bodies of theory and practice, and there is no chance nor need to present them thoroughly in this research. However, the influence of Cybernetics to design up until the 1950s in general was the general change in atmosphere relating to creativity: design was seen malleable to complete rationalization and purification from personal values and opinions.

018 Menges and Ahsquist, Computational Design Thinking: Computation Design Thinking, 11.
020 Ibid, 321.
The Cybernetics also built upon Noam Chomsky’s grammatical constructions and other linguistic theories to put forward a structuralist design approach. Originally in 1956, outside the realm of computing, Chomsky argued in linguistics that a finite set of vocabulary in conjunction with a finite set of rules for combining them can generate an infinite number of sentences in a given language. This argument proved influential for the development of many computer programming languages (such as LISP, BASIC and FORTRAN) which at the time were “ill-structured, inconsistent and clumsy.”

These developments in turn started to emphasize computing in design in the coming years. Also the Cybernetician feedback-loop is already exhibiting references to algorithmic information processing retrospectively entailing the development and use of digital algorithms in architectural design methodology.

---

023 Coates, Programming.architecture, 2.
024 Ibid, 27.
Generation II 1960-1963: Into mainstream

“Creative design was, under their view, a repetitive cycle of representation, analysis and materialization, where ‘creative’ moment was always followed by a ‘mechanical’ one: a symbiosis where ‘a designer and a computer can work together as a team on design problems requiring creative solutions’.”

- Daniel Cardoso Llach, MIT

What separated the second generation computers from the first ones was the replacing of vacuum tubes by a transistor. The use of the transistor allowed computers to be smaller, however still within the scale of a room not an object, more efficient and reliable, less heat-emitting and power consuming. Second generation computers had also higher-level programming abilities, they allowed the use of English-like syntax and were machine-independent, namely they could be used in operating different machines. Programming languages started to be considerably easier to approach as it was considered that schoolchildren and amateur programmer could learn them.

By the mid-1960s financial information in large companies was readily processed with computers. A new field of industry

---

027 Ibid, 51.
emerged, and new forms of employment such as programmer, analyst and computer system expert appeared.028

Early 1960s saw the rise of multiple design ethos purported as their own distinct disciplines and the ones that rose into fame earlier gained momentum as well. Such disciplines were the systems engineering, ergonomics, operational research, information theory and cybernetics.029 During this time people were also generally optimistic about the upcoming utilization of computers. Many of the ideas feral in contemporary design were first thought of during this timeframe.030 People visioned automated architects, design with evolutionary principles, self-replicating geometry and cellular automata, Bézier curves and such.031 Yet in 1962 a typical commercial computer cost the equivalent of 3,5 million US dollars which is the reason only few institutions were able to acquire one and why much of the future upheaval was not integrated into architectural practises.032

Cybernetics, as one of the design disciplines, came to its true prominence in the 1960s especially in architecture as it coincided with Automated Design, a theory in architecture which at the time was speculating on computer programs’ capabilities of generating specific architectural solutions based on a given brief.033 These speculations included contemplating issues of how computers could mimic human thought processes, and if they even should in the first place. The combined effect of Cybernetics and Automated Design now provided the means for an approach where architecture could be perceived and pursued as a system. I.e. architecture was seen as a complex interrelation of material parts and form shaping social engagements.034 The focus thus shifted from the outcome to the emphasis of processes.035 As a system, capable

030 Burry, Scripting Cultures: Architectural Design and Programming, 78.
032 Weisberg, ‘The Engineering Design Revolution.’
033 Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking, 11.
034 Ibid, 11.
035 Coates and Derix, ‘The Deep Structure of the Picturesque.’, 33.
of producing outcomes, architecture started to have analogies with information theory, and thus similar approaches in contemporary computer generated architecture increasingly seen to have close ties to information technologies were now visibly exhibited.


037 Hovestadt and Danaher, Jenseits Des Rasters -- Architektur Und Informationstechnologie/Beyond the Grid -- Architecture and Information Technology, 18.
Generation III 1964-1975: Loss of innocence

“Rather than face the responsibility of these difficult questions, designers turned instead to the authority of resurrected “styles.” The architectural decisions made within a style are safe from the nagging difficulty of doubt, for the same reason that decisions are easier to make under tradition and taboo than one’s own responsibility.”

- Christopher Alexander

Technical innovations in circuit boards allowed the third generation computers to be the small, efficient and cheap enough to be fitted within the context of a regular office. This migration from single centralized computer center with experts to regular offices integrated with several computers operated by office workers was later defined as the advent of distributed computing.

In 1964 when Christopher Alexander published the highly influential book Notes on the Synthesis of Form, which put forward a computer-based architectural design method that would be systematically utilized, the interest in using computers in architectural design rose to prominence in the academy. Alexander also developed programming in architecture which resulted in HIDECS, Hierarchical Decomposition of Systems, a program.

038 Alexander, Notes on the Synthesis of Form, 10.
040 Ibid, 66.
041 Carranza, ‘Programs as Paradigms.’ 68.
that decomposes given design context into a tree or semi-lattice graph.\textsuperscript{042} Alexander’s work has been seen important in advancing the algorithmic approach among the studies of form generation although he later repudiated his mathematical approach\textsuperscript{043} and many other design methods.\textsuperscript{044} Alexander also propagated for the demystification of design processes with logic and holistic scrutiny. He referred to this disclosure as the loss of innocence.\textsuperscript{045} Computers were seen instrumental in doing that, and subsequently multiple different conferences were organized around similar topics as the ones proposed by Alexander. The development of computer-aided design systems solely for architecture had now began.\textsuperscript{046}

By the mid-1960s interest in architectural computing had risen and a great amount of programs had been developed to enhance architectural design. The figure 10 on the page 86 presents a chart that present the variety of areas in design that were thought worthy at the time.\textsuperscript{047} Retrospectively, one exceptional program for floor plan generation called SLAP\textsuperscript{1} in 1965 was developed by a student of civil engineering Thomas Anderson. The program sought to minimize the total cost of circulation by locating activities in relation to one another on a rectangular grid. Since then it has been upgraded and rewritten in different programming languages.\textsuperscript{048}

Later in 1967 a prominent intellectual in architectural computing by the name of Nicholas Negroponte started to develop applications with artificial intelligence approach\textsuperscript{049} at MIT where he formed the Architecture Machine Group and the Media Labs.\textsuperscript{050}

\textsuperscript{042} Broadbent, Design in Architecture: Architecture and the Human Sciences, 310.
\textsuperscript{043} Coates, The Architecture of Programming, 172.
\textsuperscript{044} Broadbent, Design in Architecture: Architecture and the Human Sciences, 289.
\textsuperscript{045} Alexander, Notes on the Synthesis of Form, 8
\textsuperscript{047} Broadbent, Design in Architecture: Architecture and the Human Sciences, 309.
\textsuperscript{048} Homayouni, ‘A Literature Review of Computational Approaches to Space Layout Planning.’ 8.
\textsuperscript{050} Ruwan, ‘Representations for Evolutionary Design Modelling.’ 23.
Negroponte’s seminal book *The Architecture Machine* (1970) acknowledges three premises for how machines can help the design process: First, the rudimentary practices can be automated, second, that designers should alter their ways of working to that of a machine, and third, that the design process should be made readable to the machine for a mutual growth to develop.\(^\text{051}\) Negroponte was also the leading promoter of “the infusion of computer processes to architecture” and putting forward the idea of interdependence of artificial intelligence and human intelligence and how they can enhance each other.\(^\text{052}\) However, his ideas are sometimes seen to seek the elimination of the architect as an “unnecessary middleman between continually changing needs of the inhabitants and the continuous incorporation of those needs into the environment”.\(^\text{053}\) In general, Negroponte was after a computational device that could initiate procedures of its own based on a given context; a device that would be capable of sensing the needs of the inhabitants.\(^\text{054}\)

---

\(^{051}\) Ruwan, ‘Representations for Evolutionary Design Modelling.’ 38.

\(^{052}\) Menges and Ahlquist, *Computational Design Thinking: Computation Design Thinking*, 84.


\(^{054}\) Ibid, 68.
Figure 10: Computer programs and the amounts of them developed for a variety of aspects in architectural design.  


<table>
<thead>
<tr>
<th>Category</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage and retrieval of information for use in design including...</td>
<td>22</td>
</tr>
<tr>
<td>catalogues of components, performance specifications, etc.</td>
<td></td>
</tr>
<tr>
<td>Structures against which design data can be classified</td>
<td>8</td>
</tr>
<tr>
<td>Computer models of city structures</td>
<td>7</td>
</tr>
<tr>
<td>Site surveys, contour maps, etc.</td>
<td>10</td>
</tr>
<tr>
<td>Space allocation and circulation in buildings, including pedestrian,...</td>
<td>58</td>
</tr>
<tr>
<td>vehicular, service, pipework and drainage layouts</td>
<td></td>
</tr>
<tr>
<td>Estimating, cost control and quantities</td>
<td>13</td>
</tr>
<tr>
<td>Auditorium and lecture room design</td>
<td>4</td>
</tr>
<tr>
<td>Simulation of lift performance</td>
<td>3</td>
</tr>
<tr>
<td>Environmental control</td>
<td>3</td>
</tr>
<tr>
<td>General</td>
<td>9</td>
</tr>
<tr>
<td>Thermal</td>
<td>4</td>
</tr>
<tr>
<td>Lighting</td>
<td>1</td>
</tr>
<tr>
<td>Structural calculations</td>
<td>6</td>
</tr>
<tr>
<td>Building construction and component selection</td>
<td>8</td>
</tr>
<tr>
<td>Interactive graphics and visual display</td>
<td>10</td>
</tr>
<tr>
<td>Perspective drawing</td>
<td>8</td>
</tr>
<tr>
<td>Preparation of design and production drawings</td>
<td>11</td>
</tr>
<tr>
<td>Specifications</td>
<td>5</td>
</tr>
<tr>
<td>Schedules</td>
<td>5</td>
</tr>
<tr>
<td>Management in general</td>
<td>5</td>
</tr>
<tr>
<td>Office accounting and payroll</td>
<td>1</td>
</tr>
<tr>
<td>Network analysis and scheduling</td>
<td>5</td>
</tr>
<tr>
<td>Assignment of manpower</td>
<td>6</td>
</tr>
<tr>
<td>Computer control of building in use</td>
<td>1</td>
</tr>
</tbody>
</table>
In 1968, a Hungarian botanist named Aristid Lindenmayer published his seminal book *The Algorithmic Beauty of Plants*.\(^{056}\) He presented the Lindenmayer System, often referred to as the L-systems, a formal description of development in plant growth.\(^{057}\) The L-systems are symbolic notations of formal expressions that consist of a set of production rules. In general, they produce tree structures and because of that they were practical in the construction of complex computer programs. However, they were only extensively used later in the 1980s\(^{058}\) in this purpose when efficient computers were made accessible for general public.\(^{059}\) L-systems represent the kinds of computational models that were derived from the natural world, as discussed earlier on page 50. The L-systems can be seen as a conceptualisations of the myriad parallel computation taking place in nature. Others being the Cellular automata or a Voronoi diagram to name the most known ones. It was now computation that allowed the natural world to be observed from a new perspective and thus a new field of epistemology could be seen to form.\(^{060}\)

The 1970s are sometimes described as the time when most of the fundamental algorithms for computational architecture were declared.\(^{061}\) Shape grammars are one of the most famous ones. They were developed by George Stiny in the early 1970 and a book of his PhD thesis dealing with them was published in 1973.\(^{062}\) Shape grammars are similar to the structure grammars developed by Chomsky (see pages 79 in this chapter and 107 in the chapter 3) in a sense that instead of using an alphabet of symbols to generate a language, an alphabet of shapes can be used to generate a language of shapes: a shape grammar.\(^{063}\) Although Stiny was not interested in computational approaches, the study of shape grammars was

\(^{056}\) Coates, *Programming.architecture*, 73.
\(^{057}\) Burry and Burry, *The New Mathematics of Architecture*, 261.
\(^{058}\) Coates, *Programming.architecture*, 73.
\(^{059}\) Rutten, ‘Galapagos: On the Logic and Limitations of Generic Solvers.’
\(^{060}\) Coates, *Programming.architecture*, 164.
\(^{061}\) Ibid, 183.
\(^{062}\) Ibid, 184.
feral during the following 20 years\textsuperscript{064}, as shape grammars proved to be graspable for people while being suitable for use in computer programs. \textsuperscript{065} Shape grammars are discussed in more detail later in chapter 3 (on page 141) as they have been used in floor plan generation as well.

Many seminal researchers in the field of architectural computation appeared during the 1970s. Paul Coates was one of the most influential figures in the field of architectural computation. Ever since the early seventies he has introduced multiple computational techniques and founded an organization promoting architectural computing. He spent his life searching for geometric rules that could communicate the process of human occupation of space.\textsuperscript{066}

Another influential figure in architectural computing and especially evolutionary computing in architecture was John Frazer who developed a myriad of different design approaches into architecture that all proceeded to translate morphogenetic and evolutionary principles into architectural design.\textsuperscript{067} Frazer recognised architecture as a part of natural phenomena, since human environment had become a major part of the global ecosystem.\textsuperscript{068} In general, Fraser was arguing that architecture should be a participant in the evolutionary process where it is in exchange with its environment, it responds to the feedbacks from it, and is acting like a metabolism.\textsuperscript{069} His vision was far reaching and abstract, but it is easy to see how his ideas have been greatly influential to many architects developing evolutionary methods in design computation. He has been considered a pioneer of the development of coherent use and development of shape space, self-organization, complexity and emergence in architectural computation.\textsuperscript{070}

\textsuperscript{064} Coates, The Architecture of Programming, 184.
\textsuperscript{065} Gips, Shape Grammars and Their Uses: Artificial Perception, Shape Generation and Computer Aesthetics, 1.
\textsuperscript{066} Coates and Derix, ‘The Deep Structure of the Picturesque.’ 33.
\textsuperscript{067} Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking, 20.
\textsuperscript{068} Frazer and Johnston, An Evolutionary Architecture: Themes VII, 10.
\textsuperscript{069} Ibid, 20.
\textsuperscript{070} Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking, 149.
A seminal book for the development of mathematical approach in architecture named *The Geometry of Environment* was published in 1970 by Lionel March and Philip Steadman. The book described representation of plans, reports on theorems for establishing the enumeration of patterns, and properties of symmetry to name a few. Their work is seen highly influential as it provided the needed methodology for the development of design method research during the 1960s and 1970s.\(^{071}\)

Computer systems started to appear in architectural offices in the early 1970s, even though the mainstream diffusion of the computer aided design (CAD) systems as we have learned to know them did not take place. The first generation of these CAD systems only started to develop during these times. Their development is divided into two branches: a geometric modeling branch and a building-specific branch.\(^{072}\) The first addressed the drawing of complex curves and shapes, thus it had an emphasis on geometric modelling, while the latter was oriented to supporting the needs of building construction, arguably an early BIM approach. Special Interest Group for Graphics (Siggraph) was an annual conference where graphics was a specialised issue. It grew out of early computer meetings and it reached its prominence in the 1970s.\(^{073}\)

\(^{071}\) Coates, Programming.architecture, 161.


\(^{073}\) Coates, Programming.architecture, 183.
Generation IV 1975- ongoing: Personal Computers

“If we talk in terms of Inventors, Masters and Diluters (as Ezra Pound classifies inventive activity), then we saw most of the Inventors in the early 60s, the Masters in the early 70s and then we got the Diluters in the mid 90s ... .”

- John Frazer

There is no generally accepted distinction that would separate the fourth generation of computers from the third. As circuit board technology advanced from what is commonly called large scale integration (LSI) to very large scale integration (VLSI), the computers again became more efficient, small and more like the desktop computer we have learned to recognize. Coincidentally, the computers we use today are considered to originate from these technological innovations made in the late 1970s. After that, there has been numerous advances in technology, software and interface.

Only in the late 1980s did the software start to have capabilities to support professional architecture, but the overall conclusion from the development of CAD was that architects acquired capabilities for drafting and rendering, and lost the whole analytical ability of design computation that was the main reason

076 Ibid, 34.
for introducing computers into architecture in the first place.

In the 1980s, computer aided design developed into a distinct field of its own.\textsuperscript{077} By the mid 1980s, when the development of graphical interface in computers had made every day task innate, drafting and sketching with computers were seen viable. This created a need to develop new second-generation CAD software which unfortunately in the beginning remained too limited to allow professional architectural drafting. This limitation was for the sake of intuitive utilization and ease to learn these new softwares.\textsuperscript{078}

The structuralist design theory continued to develop under the title of \textit{Space Syntax} which approaches were highly similar to the structuralist ethos. For example Philip Steadman is sometimes mentioned as a influential predecessor.\textsuperscript{079} The Space Syntax approach has most likely been the most well known design approach that has influenced architectural computation and computer-aided spatial analyses. Since its inception in the 1970s, it has developed into one of the most advanced sets of methods for morphological analyses for cities and buildings\textsuperscript{080} despite some harsh criticism it has received.\textsuperscript{081} Syntactic methods base themselves on syntactic structures that are used for computation as well as for methodological purposes. Combined, they constitute a theory of a generative nature of space.\textsuperscript{082}

The use of graphs for representing floor plans in computational floor plan generation received much attention during the late 1970s and early 1980s. The so called \textit{GRAMPA} program which took advantage of this method was published by John Grason in 1971 and similar programs have since been in development.\textsuperscript{083}

In 1988, the evolution influenced design approach took a seminal leap forward as the publication of Richard Dawkins’ \textit{The
Evolution of Evolvability\textsuperscript{084}, and later The Blind Watchmaker coincided with the technical advancements of personal computers.\textsuperscript{085} Dawkins described essential aspects found in computational algorithms influenced by evolution, which influenced designers to take on more complex computational methods, including evolutionary methods.\textsuperscript{086}

It was also in 1988, when mathematician Samuel Geisberg and his Parametric Technology Corporation created Pro/ENGINEER, often credited as the first commercially successful modelling software that in contemporary language could be described as parametric.\textsuperscript{087} It was a history-based modeller, i.e. it kept track of the phases of how the geometry was created by the user, and thus allowed to make changes later in the tracked record of modifications.\textsuperscript{088} However, it should be noted that it was not specifically developed for architects, and it should not be confused with programs that are equipped with a graphical interface where algorithms manifest in the form of flowcharts. Despite the fact Pro/ENGINEER is considered a seminal precedent in the history of software affiliated with computer generated architecture.

\textsuperscript{084} Coates, The Architecture of Programming, 95.
\textsuperscript{085} Rutten, ‘Galapagos: On the Logic and Limitations of Generic Solvers.’
\textsuperscript{086} Coates, The Architecture of Programming, 95.
\textsuperscript{088} Ibid, 23.
1990s: Deleuzian remedy

“In a vacuum of theory designers using computation began to describe their process as “experimental” without stating the hypotheses on which they were founded, making any results impossible to evaluate qualitatively.”

- Greg Lynn

The turn of the 1980s and 1990s are marked with rampant computational hegemony which was later saved by (or, it can also be argued that it actually saved) an avant garde rhetoric of the time, as their trajectories crossed somewhat accidentally.

The use of personal computers and their image processing capabilities continued to soar. However, CAD was taught and perceived merely as a productivity tool by individuals who had previously been involved in technical drawing, detailing, and such.

During the 1990s CAD became an everyday tool in architectural offices and the first developments of parametric modelling, and early BIM took place. Computation in architecture reached hegemony and started notoriously to promote the “digital” as a means of pursuing the overall goal. As a result, the historical and theoretical

---

089 Lynn, Archaeology of the Digital, 11.
090 Carpo, ‘Ten Years of Folding.’, 91.
091 Carpo, The Alphabet and the Algorithm, 90.
092 Burry, Scripting Cultures: Architectural Design and Programming, 35.
093 Ruwan, ‘Representations for Evolutionary Design Modelling.’ 61,
continuum was superseded by technological practices.\textsuperscript{994} Meanwhile, The French philosophers - most prominently Gilles Deleuze - gained interest in architectural circles of American critical theory, when architects such as Peter Eisenman and Greg Lynn started to device architectural versions of Deleuze’s work.\textsuperscript{995} In his book, \textit{The Fold}, Deleuze contemplates on the idea of continuity which he then extends on Gottfried Leibniz’s work. It is the \textit{fold} — “a unifying figure in which different segments and planes are joined and merge in continuous lines and volumes”\textsuperscript{996} — that in Deleuze’s discourse became of special importance.\textsuperscript{997} Eisenman elaborated on Deleuze’s theory of folds into notions dealing with change and continuity of variations,\textsuperscript{998} and emphasised folding as a process, not a product in architectural design.\textsuperscript{999} Figure 11 presents a stereotype of a “folded” building.

Lynn at the time was arguing for theory of continuities for all types: visual, programmatic, formal, technical, environmental, socio political, and symbolic.\textsuperscript{100} The rise of the formal continuity can be seen as a reaction against the deconstructivist \textit{fracture}\textsuperscript{101} that had gained popularity already in 1988.\textsuperscript{102} Together with the fold, the \textit{Deleuzian ontology for design} was also introduced and it helped the evolutionary design explorations to gain momentum. In general, this ontology similar to evolution emphasised the generating mechanism of an outcome, a design or a lifeform. Thus, instead of a single design there was now a population of designs, and the identity of each design was brought about by the morphogenesis instead of an identity defined by its essence.\textsuperscript{103} Especially Manuel De Landa developed Deleuzian ontology for design.\textsuperscript{104}

\textsuperscript{994} Lynn, Archaeology of the Digital, 8.
\textsuperscript{995} Carpo, ‘Ten Years of Folding.’, 14.
\textsuperscript{996} Carpo, The Alphabet and the Algorithm, 86.
\textsuperscript{997} Carpo, ‘Ten Years of Folding.’, 14.
\textsuperscript{998} Carpo, The Alphabet and the Algorithm, 87.
\textsuperscript{999} Carpo, ‘Ten Years of Folding.’, 16.
\textsuperscript{100} Carpo, The Alphabet and the Algorithm, 89.
\textsuperscript{101} Ibid, 91.
\textsuperscript{102} Schumacher and Schumacher, The Autopoiesis of Architecture: A New Framework for Architecture: V. 1, 121.
\textsuperscript{103} Ruwan, ‘Representations for Evolutionary Design Modelling.’ 20.
\textsuperscript{104} Ibid, 67.
Figure 11: Elevation of Max reinhardt House, the "folded" building by Peter Eisenman.

The interesting part in the pursuit of the formal continuity in architecture is that architects affiliated with it benefited remarkably from the computational capabilities developed at the time. Since computers could easily provide tools for manipulation of continuous architecture, they were readily harnessed in the conception, representation and production of it. It is through this peculiar overlap of interests — a call for continuous geometry and the ability to efficiently manipulate it — that intertwined the computational hegemony and critical theory into one evolved discourse about theory of continuity. It is asserted that without

105 Adams, ‘The Eisenman-Deleuze FOLD.’ 164
106 Carpo, The Alphabet and the Algorithm, 90.
107 Ibid, 91.
the theory of continuity in architectural form, the developments of computing in the 1990s would not have inspired new form, and likewise without computers the theory of folding would have possibly died out. Instead, the process started a development that later progressed from the 1990s folds to fully smooth and curvilinear *blobs* of 2000s.

![Figure 12: Kunsthaus Graz, a ”blob” building by Colin Fournier and Peter Cook.](image)

Considering strictly floor plan generation, in 1995 John Gero and Vladimir Kazakov developed a spatial layout generating method influenced by genetic algorithms (see page 146). In general, they developed a method that could adapt to a predefined set of design parameters to automate the design task. In later versions the system was designated the ability to learn stylistic features from given design examples. Thus the program was able to change the representation it was initially seeking.

In 1999 Scott A. Arvin and Donald H. House started to develop a program that generated floor plans by simulating physical forces and properties (see page 134). Instead of creating geometrical primitives, the user creates the plan with the help of certain objectives, pseudo physical forces. By defining the state of these objectives the plan will emerge.

---

108 Ibid, 92.
109 Ibid, 92.
111 Ibid, 11.
2000s: Morphology of the emergent

“Consider the way these digital design research centres acquire odd names or unhelpful acronyms, almost as a subversive move to obfuscate what really goes on inside.”

- Mark Burry

Similarly to the 1990s, the computer technology continued to develop steadily and the architectural discourse of continuity was accompanied by the research of emergence in the 2000s. Some would argue that the period of folds and blobs would still be continuing. However, there was ground breaking developments taking place in the architectural software development which would later truly introduce digital computation into the mainstream.

The former employees of Parametric Technology Corporation, who developed the Pro/ENGINEER in 1988, set out in the year 2000 to create an architectural building modeler, described at the time as parametric, called Revit. In Revit, the parametric relationships were hidden within the program, and thus the program itself was already the model to be used. Unlike contemporary software referred to as parametric, the emphasis was on the use of a model, not the creation of models.

---

the user would use objects and adjust their parameters to achieve certain effects, such as altering the pitch of the roof which in turn would revise (or **revit**) all drawing accordingly.\textsuperscript{116} However, after their acquisition by Autodesk in 2002, Revit has been described as a software for Building Information Modelling (BIM), not as parametric modelling.\textsuperscript{117}

In the late 2000s, two seminal softwares developed for architects that integrated a visual interface for editing algorithms were released. Bentley Systems and Robert Aish released *Generative Components* in 2003 which was followed by Robert McNeel & Associates’ *Explicit History* developed by David Rutten in 2007, which later became known as *Grasshopper*.\textsuperscript{118} Unlike Revit and Pro/ENGINEER, Generative Components and Grasshopper integrated graphical interfaces of flowcharts representing algorithmic processes and emphasized the creation of models, not the use of existing models, namely objects within the software. These kind of editors have become widespread as their use does not require high level of technical knowhow in comparison to the traditional programming languages.\textsuperscript{119} Since the release of modern visual algorithmic editors, the amount of computer generated architectural designs and research has gone feral.

Speculative computation research has existed in the academy ever since its inception in the 1960s, most notably in the Architectural Association (AA), the Bartlett, SCI-Arc, MIT, ETH and RMIT University\textsuperscript{120} and continued to do so throughout the 2000s. Although, towards its end, design research practices and novel studio teaching truly surfaced and has remained in the forefront of academic discourse.\textsuperscript{121} One of the programmes or groups that has gained much of publicity that was created in the year 2000 by Michael Hensel and Michael Weinstock. They formed a new masters programme called the *Emergent Technologies* at the AA. Later in 2002, Achim Menges joined in. The group has since

\begin{itemize}
\item \textsuperscript{116} Ibid.
\item \textsuperscript{117} Davis, ‘Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture.’ 25.
\item \textsuperscript{118} Davis, ‘A History of Parametric – Daniel Davis.’
\item \textsuperscript{119} DİNO, ‘Creative Design Exploration By Parametric Generative Systems In Architecture’, 216.
\item \textsuperscript{120} Burry, Scripting Cultures: Architectural Design and Programming, 15.
\item \textsuperscript{121} Ibid, 22.
\end{itemize}
been epitomizing the domain of computational design research in architecture with the help of multiple publications and visibility of the AA. Their work encompasses a wide array of computational approaches, fabrication methods, material system studies, and evolutionary strategies to name a few.  

The word emergence, along with evolutionary, has been touted in architectural publications in such a quantity that by now it can refer to myriad different topics. The research of emergence within architecture, which consciously started in the 2000s, is actually referring to the science of emergence — the study of “the behaviour of complex systems and the mathematics of their processes” — and how it can be implemented into processes of design. Typical emergent approach derives from studies of natural processes which are then represented using computation. However, at this point it is impossible to elaborate further the consequences of the 2000s. It can be argued to still continue. It is also not clear what is the true value of speculative computing in terms of culture and economy.

In 2004, The Computational Design and Research group at Aedas R&D was founded. The group decided on focusing on “the application of artificial life and intelligence based on self-organization, bottom-up processes and distributed representation to architectural design.” These kind of processes “for design computation of spatial configurations” have been on the rise ever since the economical crisis forced the construction industry to focus on reconfigurations of existing building space. The Computational Design and Research group at Aedas R&D might be credited the status of continuing the work started by Frazer, Coates, The Space Syntax, and thus the structuralist theories.

122 Hensel, Menges, and Weinstock, Emergent Technologies and Design: Towards a Biological Paradigm for Architecture, 5.
123 Hensel, Menges, and Weinstock, Emergent Technologies and Design: Towards a Biological Paradigm for Architecture, 11.
124 Ibid, 11.
127 Ibid, 43.
2010s: Parametricism

“The theory of architectural autopoiesis claims that Parametricism is the great, new, viable style after Modernism. (Postmodernism and Deconstructivism are interpreted as transitional episodes.)”

- Patrick Schumacher

There is no consensual understanding nor appreciation for the path leading up to the contemporary status of computing in architecture. Many researchers and experts consider computing in architecture to be still in its infancy, despite its long history since in the 1960s. However, visions for future computation in architecture are somewhat polarized: It might be argued that the current period is still part of the experimentation era that started in the 1990s and developed through the 2000s — large part of the most cutting edge thinking within computational architecture is still processing biomimetics — or that we are on the verge of a revolution affecting all domains of architecture, not just computing. The latter vision is commonly credited to Patrick Schumacher.

130 Ibid, 55.
131 Ibid, 27.
133 Ibid, 27.
Schumacher. The final segment of the history chapter will introduce Schumachers’ and the contrasting predictions of the future.

Parametric design secured its acceptance from practice, research and education during the 2000s as its generative and analytical properties were increasingly seen beneficial.\footnote{134} Referring to the infancy mentioned above, several researchers cling to the necessity for even more compact, expressive, and comprehensible programming languages, or more intuitive and easier software for the creation of computational models of architecture.\footnote{135} Some are even after ready made packages that include capabilities to simulate environments and thus to harness “the emergent complex dynamic systems” into the processes of design without the need to construct them from scratch at all.\footnote{136} Desolate voices are pointing out that a universal common language applicable to all design tools is yet to be developed. They also state that discipline-specific algorithms should be written out instead of acquiring them from other fields. All this might however lead to excessive and vain efforts “by architectural reinvention of wheels” which are as good as the ones already in use in other fields.\footnote{137}

Patrick Schumacher took possession of the parametric and the role of the imprudent protagonist of the computational avant-garde when he declared that “postmodernism and deconstructivism were mere transitional episodes”, and that “the great new style after modernism” will be parametric by its nature.\footnote{138} Schumacher interprets the postmodern aesthetics as the “rejection of the aesthetic values of homogeneity” and “the celebration of diversity, collage and fragmentation”. According to Schumacher, they are signalling the departure from Fordist mass production and thus marking the beginnings of a “new urban complexity of post-fordist pattern of economic development”.\footnote{139} Schumacher asserts that “the systematic incorporation of layering, the articulation of gradients,
the employment of morphing to produce morphological series“ are all examples of concepts that “serve to facilitate the articulation of contemporary societal patterns”. All this can be refined by parametric design techniques, which sanctions parametric methodology to claim the throne of the contemporary style: Parametricism. As it remains, it is impossible to say how these claims will play out.

The current developments in architectural computing are far reaching and increasingly more diverse. Many of the developments go undetected as there is vast amounts of information circulating throughout the internet and its online communities. Formal studies however are buried in the studios of different schools and companies. The direction of the general discourse is multifaceted and complex. Any prediction of the future should thus be treated with great scepticism.

This concludes the history chapter of computational design. The precedent analyses of the following chapter however could be considered as a more in depth look at the history most relevant to this thesis, namely the one of floor plan generation.

140 Ibid, 129.
III. Floor plan representation

“... [a grammar should be able to generate] all and only all the well-formed sentences in a language.”

- Noam Chomsky

In the first two parts this research has addressed general issues of computing in architecture. It has been necessary groundwork to form a better understanding of the abilities and objectives of design computation before delving into a specific field within it, namely the floor plan generation. This chapter presents methods of constructing the geometry of a computational floor plan, that is the geometry representing architecture in the overall process of floor plan generation. Figure 13 presents a recap of the thesis structure i.e. the whole process that is needed to study in order to generate floor plans using a computational method. The way in which geometry is constructed has significant influence on the generated outcomes.

This chapter commences by introducing a structuralist or syntactic design methodology. However, the part constituting majority of this chapter is a survey of the most notable precedents in floor plan generation. These prototype models will demonstrate the objectives and processes common in floor plan generation. Their methods will be analysed in order to frame and influence the experiment carried out in the fifth chapter of this research. This chapter thus presents the elementary base for developing an

001 Coates, The Architecture of Programming, 2.
understanding of the aspects of architecture emphasized when generating floor plans i.e. which aspects are brought into the model — and which are intentionally left out — in order to derive purposeful results (see the model discussion in chapter 1, on page 63).

It should also be clarified that quite often the models are meant to simulate human design processes and thus provide representations of intermediate designs rather than final detailed ones. That is their scope of detailing resembles more the illustrative and schematic designs rather than construction documents. This is also partly because the complexity of the model will drastically increase when more details are embedded into it. Simple models allow computers to process them more thoroughly, and so the likelihood of an novel design scenario to appear is greater as the details are not burdening the general process.

Figure 13: The process of research this far.

Models that are used to generate floor plans and represent certain aspects of it, require generally a formalization of the logic architects use in solving architectural problems. Since the computer is able to exhaustively generate vast amounts of permutations, the history of floor plan generation models can be somewhat reduced to the quest of reducing the amount of appearance of unfit solutions. The above is why the theory of syntactic or structuralist methodology, that was introduced in the history chapter, is further elaborated here before going into details of precedent prototypes of floor plan generation.

Structuralist or syntactic methodology is often described with the help of analogy to grammar. Grammar


in traditional sense provides abilities to produce wide sets of meaningful language. The aspect of grammar that is of interest in relation to the structuralist or syntactic methodology, is its mechanism of decomposing a text into its syntactic components. 

As presented in the history chapter, Noam Chomsky asserted that the rules applied in deconstructing a given text could be further applied to the resulting parts. This process of applying rules to deconstruct could be repeated until the basic phonemes of the text were derived. Figure 14 illustrates this process. The significant realisation, however, is that this process could also be inverted; from the basic phonemes to correct sentences. Chomsky referred to this process as generative grammar.004

A syntax, according to its definition, is the arrangement of words and phrases to create correct sentences. However, syntactical correctness does not indicate semantic correctness, though syntactical incorrectness does indicate semantic incorrectness as the information provided does not make sense.005 The analogy to architectural design then is that when creating models (to generate floor plans or any other architectural aspects), architects need a way to eliminate the appearance of meaningless designs and shapes during computation, and thus create an architectural vocabulary that could be applied when a specific context is confronted.006 An example of an architectural analogy of generative grammar is provided in figure 15 where the hierarchy of a classical building is decomposed or constructed, depending on which is preferred.

005 Ibid, 2.
Figure 14: The construction/deconstruction of a sentence.

Figure 15: The construction/deconstruction of classical order in architecture

008 Ibid, 140.
In the end, Chomsky’s argument was that it is mathematically possible for a finite lexicon in combination with a finite syntax to produce infinite number of correct sentences: not only syntactic but semantic as well.\textsuperscript{009} If one restructures the design process in this manner, it is easy to see how computing can help to produce infinite sets of coherent designs. The development lineage of the floor plan generation precedents presented in this chapter, implies that the strengthening of this logic is the indicator of enhanced capabilities to produce novel design.

\textsuperscript{009} Coates, The Architecture of Programming, 2.
Syntactic approach

“A building may therefore be defined abstractly as a certain ordering of categories, to which is added a certain system of controls, the two conjointly constructing an interface between the inhabitants of the social knowledge embedded in the categories and the visitors whose relations with them are controlled by the building. All buildings, of whatever kind, have this abstract structure in common ... “

- Bill Hillier and Julienne Hanson

The theory of transformational generative grammar has provided a strong metaphor for the logic of generating and analysing spatiality. Although, it is not a grand unified set of constructs, but rather a characterizing method, that has been best presented in the works of such architectural pioneers as John Gero, Paul Coates, Philip Steadman, Lionel March, William J. Mitchell, Bill Hillier, and more recently Christian Derix and Åsmund Izaki, to name at least a few. However, the approach these designers have helped to develop has been loosely categorized under the title of structuralist, or the syntactic approach epitomized by the alleged Space Syntax. Some of them have been influential in the early developments of the groups inception of which is later merited

010 Hillier and Hanson, The Social Logic of Space, 147.
012 Coates, Programming.architecture, 160.
to Bill Hillier (see the history chapter on page 91) around the late 1970s and early 1980s. The definitions presented in the next chapter are then paraphrasing these structuralist, spatial syntactic ideas.

In general, with the syntactic approach, the social aspects found in architecture are simplified into mere physicalities in a technocratic manner, and space is constructed in two dimensional plane, that is as lines. Thus in syntactic approach, the spatial variability and richness can be seen as undeveloped compared to traditional intuitive design methods. In this manner, syntactic floor plans, from a very critical point of view, are nothing but layouts determined by accessibility or adjacency of territories.

Therefore, it is no coincidence that this kind of an approach of perceiving things in purely physical terms has received a lot criticism for not enhancing its fundamental principles itself. Nevertheless, syntactic approach has a strong follow up of research that still continues to develop it. Despite the criticism it has received, as a tool in general, it is applicable for certain uses with limitations. It provides a coherent method of constructing models that are appropriate for computational purposes. It should also be noted that the following model construct is not a space syntax approach, but rather one constructed along the principles of syntactic, structuralist kind approach.

015 Ibid, 164.
016 Ibid, 162.
Syntactic lineage

“Architecture is considered as a form of artificial life, subject, like the natural world, to principles of morphogenesis, genetic coding, replication and selection.”

- John Frazer

Floor plans have exhibited different features throughout history, as buildings they have been representing have served different purposes and different users. Therefore, the characteristics of the spatial elements, such as rooms and corridors, making up a floor plan are in constant typological change. However, the packing methods or the topological relations of these elements are visible in somewhat every floor plan, that is, they are constituting for the characteristics of a floor plan. In a similar manner, internal relations in buildings can be seen to be developed by syntactic means.

A formal way of describing the earlier mentioned generality found in a floor plan can be rephrased as in the following. A building can be seen as an ordering of categories, which are treated according to a certain system of controls to form an interface for its inhabitants.

A perimeter that defines a space also defines its usage as the perimeter itself manifests in a certain way such as a wall, fence, or the edge of floor. Therefore the inhabitation taking place in a space characterizes it and places it in a certain category.

017 Frazer and Johnston, An Evolutionary Architecture: Themes VII, 9.
018 Hillier and Hanson, The Social Logic of Space, 147.
category is used to emphasise the social intrinsic nature of space as the perimeter defining it can manifest in different forms. However since the perimeter controls and maintains the existence of the given inhabitation, it is a integral part category. Buildings, and thus floor plans, separate collections of categories from the outside world by constructing spatial separation that controls the “system of social categories” and thus forms a system of control.

All buildings in their varying forms can be seen to follow this abstract definition and form types using different emphasis of varying relations i.e. certain typologies are exhibiting certain emphasis by varying the syntactic parameters. If a building constitutes of the above mentioned ordering of space, and thus allows internal permeability, then every part of the building is accessible without going outside it. So, it follows a general property of a building: the presence of a continuous outer boundary.

The most simple definition for a building is a closed cell with “a permeability defining a contiguous open cell”. The cell is the spatial layer of the category mentioned above. The cell is synonymous to the spatial element. As such it can refer to such spatial elements as a room, corridor or merely a defined area. This cell has then throughout history evolved into different forms and compositions through “the internal logic of social solidarities”, from which certain syntactic and parametric transformation rules can be extracted and an abstract genotype for a building formed. Building typologies are seen as a set of amplified or restricted relations, making up a characteristic combination of certain genotypic transformation i.e. a certain arrangements of syntactic dimensions, or parameters.

Often these dimensions consist of distributed and non-distributed components of a building and a typology starts to emerge as either one is emphasized or subjugated. Distributed

019 Hillier and Hanson, The Social Logic of Space, 19.
021 Ibid, 147.
022 Ibid, 147.
023 Ibid, 176.
024 Ibid, 177.
025 Ibid, 182.
components here refer to non-hierarchical, accessible spaces for everybody, dwellers and strangers alike, whereas non-distributed components include spaces accessible only for the inhabitants, such as the staff or inhabitants.\textsuperscript{026} An example of a building with a maximized distributed space with a shrunken non-distributed could be a store or a shop, while an example of an maximized non-distributed with a shrunken distributed could be a rowhouse or a detached house.

Despite the fact that these elements can be used to spatially program any given building, or a floor plan, their statuses can change, as the users accessing the elements have different profiles. Such examples are schools, prisons and museums where the people, categorized as inhabitants, have access only to the distributed parts.\textsuperscript{027} This social logic is not a concern for the computation, but it is relevant to understand the generalization of a floor plan in order to make sense of the potential of the floor plan generation model.

\textsuperscript{026} Hillier and Hanson, The Social Logic of Space, 182.
\textsuperscript{027} Ibid, 182.
Computational Generation

“We are not interested in a machine that will simply parrot a human designer, nor are we interested in machine that will have an autonomous existence by which to mimic and replace an architect. An Architecture Machine will feature a dependence. An artificial intelligence is in fact an interdependence.”

- Nicholas Negroponte

As a problem of arranging given objects, floor plan generation relates to a class of problems that encompasses many fields outside architecture. Such fields include organizing packages in cargo industries, and design of circuit boards, to name a few. In short, floorplan generation can be defined as the task of uncovering the arrangement of defined objects where distances between them are in accordance with the given adjacencies.

Problems relating to floor plan generation are often ill-defined — meaning that they are not defined well enough, as the constraints are not clearly formulated — and over-constrained — that there is no single solution, but several possible ones.

028 Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking, 84.
Moreover, solving ill-defined problems means searching for a set of \textit{design constraints} — methods that are able to sort out or categorize good solutions from a large set of possible solutions — which can be continuously modified in order to refine the problem definition.\textsuperscript{032} Also one needs to somehow address the \textit{topological} properties — how the relationships between individual elements in the floor plan are determined — and \textit{geometric} properties — how the element dimensions are determined.\textsuperscript{033}

Various prototypes for generating floor plans have been developed and their numbers grow exponentially, as software becomes increasingly more attainable and the internet is extensively utilized for both collection of information and sharing of designs. Many of these modern day prototypes go unnoticed, and they are not accompanied by formal research. The online communities are filled with informal designs purported to provide novel results in any given task. Therefore, this study analyses only those prototypes and their methodological characteristics, which exhibit a clear lineage of developments and which are accompanied by formal research. This is the criteria to be considered a precedent in this research.

The general logic of the construction of the precedent models and how they proceed to generate floor plans are presented in the remainder of this chapter. The actual computer language is not scrutinized, as it is only relevant to the computational qualities and does not affect the architectural qualities. In other words, the computer language only affects the efficiency of computation. Therefore the \textit{pseudo code} of precedent prototypes is presented to understand the logic of how these prototypes proceed to create architecture.

To a large extent, the results produced by these prototypes are generalized mappings of spaces rather than clear presentations of architectural floor plans. Figures 16-19 on pages 119-120 clarifies this by showing what is commonly needed in order to transform computed outcomes into traditional architectural drawings. Usually the generated layouts are only addressing adjacencies and

\footnotesize
\textsuperscript{032} Arvin and House, ‘Modeling Architectural Design Objectives in Physically Based Space Planning.’ 214.
\textsuperscript{033} Ibid, 216.
excluding other design conditions, such as site-specifics, views, or orientation.\textsuperscript{034}

The prototypes can be seen exhibiting different methods by which they have been categorized.\textsuperscript{035} These categories however have varied throughout the history.\textsuperscript{036,037,038} Each method or approach to construct a model can be seen as an attempt to resolve the obvious shortcomings of its precedent. However, this kind of categorizing implies an historic continuum of bygone simplistic methods and prevailing more complex methods, and may be confusing when presenting a holistic view of different properties. There is always use for an elementary method as the requirements of a given task change. For example it would be overwhelming to apply an complex algorithm framework to process a rudimentary task when a simple algorithm can achieve acceptable solutions.

For the sake of clarity, the somewhat accepted categorization of floor plan generation methods — \textit{procedural}, \textit{heuristic} and \textit{evolutionary}\textsuperscript{039} — is altered in this research: The procedural and heuristic methods are categorised under the title \textit{primitive} and evolutionary under \textit{advanced}. This is because the procedural and heuristic approaches encompass methods developed in accordance with human socio-cultural customs, and intuition. Thus they do not take significant advantage of advanced processing and storage capabilities offered by the computer.\textsuperscript{040} The third asserted category, evolutionary, is problematic because of at least three reasons. They are described in the following.

First, evolutionary is a widely and ambiguously used term. This is why it is only partially descriptive of labeling the methods


\textsuperscript{035}Koenig and Knecht, ‘Comparing Two Evolutionary Algorithm Based Methods for Layout Generation: Dense Packing versus Subdivision.’, 286.


\textsuperscript{037}Ruwana, ‘Representations for Evolutionary Design Modelling.’ 53.

\textsuperscript{038}Broadbent, Design in Architecture: Architecture and the Human Sciences, 309.

\textsuperscript{039}Koenig and Knecht, ‘Comparing Two Evolutionary Algorithm Based Methods for Layout Generation: Dense Packing versus Subdivision.’, 286.

of construction, or drawing of geometry. Expressions such as emergent or morphogenetic might denote the same kind of processes manifested in the models as well.

Second, category of evolutionary was prompted because the models started to exhibit more complex algorithmic constructs. The use of the term evolutionary is purportied because the processes of these models are counter to the human intuition. The design or generation proceeded through complex computational processes that operated the geometrical model to arrive in a desired solution. Similar complex algorithmic constructs also had a field of study in computer sciences and mathematics, metaheuristics - a higher-level algorithmic procedures designed to find or generate solutions to an optimization problem. This overlap between fields points out that metaheuristic constructs can be used for a variety of different problems, not only for floor plan generation. These algorithmic constructs are thus not inherent to floor plan generation.

Third, if the term evolutionary is used to denote the counter human logic it says nothing of the metaheuristic itself. A general misconception would most likely be that metaheuristics proceed through evolutionary logic. This is however not true, there exists a variety of different metaheuristic constructs. Evolutionary computing is only one of them.

This is also why in this research the floor plan generation precedents are separated into their own chapter and computing processes in their own. As stated earlier this chapter only presents the pseudocode for the geometry generation, not the actual computational processes behind it. The fourth chapter is dedicated to explain that since it is considered sufficient for now to explain it in general terms. It is more tangible to focus on computational processes in detail after one has a comprehension of how the architectural geometry is percieved.

Figure 16: Generalized mapping of a program into a network (16.1) that can be used as a foundation for implementing varying kind of architecture (16.2-16.4).

Figure 17: Two variants manually drafted according to a generalized mapping of an architectural program.\textsuperscript{043}

\textsuperscript{043} Shaviv, ‘Layout Design Problems: Systematic Approaches.’ 40.
Figure 18: Generalized mapping of a program into spatial objects that can be converted into a traditional plan drawing.

Figure 19: a) Typical floor plan representation. b) Coarse square grid representation of the plan. However it can only crudely represent it. A finer grid (c) is more accurate yet there is excess amount of redundant grid-lines.

044 Steadman, ‘Generative Design Methods and the Exploration of Worlds of Formal Possibility,’ 27.
Figure 20: Architecture with no clear boundaries is hard to be represented in a computational model.

Figure 21: Fluid, advanced geometries require more computational processing than primitive orthogonal geometry.

Primitive methods

“Colourless green ideas sleep furiously.”

- Noam Chomsky\textsuperscript{047}

Methods characterizes as procedural were the first ones to be harnessed in floor plan generation.\textsuperscript{048} As such, these elementary computational approaches were excessively systematic and required extensive amounts of information in relation to the given assignment. Since such a workflow is time consuming and tedious, prototypes characterized as heuristic were devised to overcome this problem by using more approximate information in the process of design generation.\textsuperscript{049}

Procedural

Procedural methods take advantage of the computational capabilities of the computer to systematically accumulate permutations into complex aggregates. Permutation here refers to a small algorithm that processes given information, and an aggregate to the set of all possible outputs of that algorithm, i.e. according to different input values. As an example, figure 22 shows an aggregate of all possible permutations generated for a six room plan. The plans

---

\textsuperscript{047} Coates, The Architecture of Programming, 2.


\textsuperscript{049} Ibid, 255.
are dimensionless, meaning that they are compiled using a given
unit and relative differences in dimensions are excluded. The exact
dimensions can be included by reassigning the accurate values to
the grid intervals if a given is chosen to be used in further design
processes.\textsuperscript{050}

This process thus requires human cognition to first create
a permutation, and finally also to assess the aggregate consisting
of them.\textsuperscript{051} The computer merely proceeds to generate all possible
alternatives of one permutation. In practise, assessing them —
either by an architect or a computer — would take insurmountable
amount of time.\textsuperscript{052} This is why procedural methods are sometimes
described as linear processes and exhaustive. \textbf{Figure 22} is also
descriptive in showing that previous permutations can not influence
new permutations. Although permutations can be ranked after the
process has run, there is no course of development based on the
previous design, nor is there capabilities to adjust a permutation.

Floorplan generation methods characterized as
procedural can be further grouped into two categories: \textit{additive} and
\textit{permutational}.\textsuperscript{053} The first one exhibits truly one directional progress,
while the latter allows the process to return to initialization, and
thus allows some adjustments.

\textsuperscript{050} Steadman, Architectural Morphology: An Introduction to the Geometry of
Building Plans, 11.

\textsuperscript{051} Kalay, Architecture's New Media: Principles, Theories, and Methods of

\textsuperscript{052} Ibid, 238.

\textsuperscript{053} Ibid, 242.
Additive Space Allocation

A simple definition of additive computing could be described using the following example of computational design process. A cell grid is first defined. Then the spatial element in the given design brief defined to have the most connections is placed in the center, so that it is accessible from all sides. Next, the element defined with the second most connections is placed adjacent to the first one. Then, the third element placed is the one which has been defined as having the most connections to both previous spaces. The placement is now performed by calculating the sum of all the

distances between spaces in the current layout and minimizing it. The remaining spaces are placed with the same logic, always using less and less connected spatial elements from the building program. The result is a core driven buildup of spaces.\textsuperscript{055}

Site-specific issues can be addressed to an extent, if the grid cells used in allocation are given \textit{desirability} values. A given cell then might have the property for a good view, proximity to a road and such. Under those circumstances, each space placed in allocation must have predefined \textit{preferential attributes} in order to be placed according to the desirability values in the grid.\textsuperscript{056}

The obvious flaw in this method is that the spaces placed first most likely will hinder the latter spaces to achieve optimal connectivity, since once placed, spaces cannot be moved. Also, the spaces placed first will most certainly be blocked from sun light.

The GRANDPA program originally created by John Grason 1971 is usually noted as the precedent for prototypes using additive methods. The program can be seen to consist of two layers: a model constructed using the \textit{dual graph} representation and an exhaustive search procedure performed on the dual graph.\textsuperscript{057} Figure \textbf{23} illustrates such a model\textsuperscript{058} and figure \textbf{27} the final outputs of it.

The dual graph methods base themselves in \textit{graph theory} developed by Leonhard Euler in the eighteenth century.\textsuperscript{059} The general idea is to overlay two graphs: the \textit{room adjacency graph} and the \textit{floor plan graph},\textsuperscript{060} hence the name \textit{dual}. Each room, and each of the four outside areas in a rectangular plot, is considered as a \textit{node} in \textit{adjacency network} and each wall that is in between them an \textit{edge}. The nodes and edges can be \textit{labeled} to contain information such as the area of the room and the length of the wall.\textsuperscript{061}

\begin{thebibliography}{99}
\bibitem{057} Homayouni, ‘A Literature Review of Computational Approaches to Space Layout Planning,’ 4
\bibitem{058} Mitchell, Computer-Aided Architectural Design, 206.
\end{thebibliography}
The dual graph representation is conventional for constructing a computational model, as the room adjacency graph can be easily read by a computer.062

Figure 23: A dual graph. The thick lines are representing walls.

A simple way to construct a dual graph representation for a floor plan can be described as following. First, define an adjacency requirements matrix. Second, test that the produced adjacency network is planar, meaning that there is no intersections between the network connections as they would result in unrealisable floor plans.063 Third, construct the corresponding edges i.e. the floor plan. Finally, the shapes and dimensions can be adjusted accordingly. Figure 24 illustrates these sequences of design.

Figure 24: The sequence of unfolding a dual graph. The thick lines can be seen to represent walls.

Unfortunately, it is common for the adjacency graph to produce unrealizable floor plan, or edge graphs, since information in labels alone do not always distinguish between realizable and unrealizable floor plan. A given network of nodes can also take multiple different planar forms as shown in Figure 25.064

Also, a single adjacency graph can further produce multiple geometries. This can be exemplified with the help of four Frank Lloyd Wright buildings, in Figure 16 on the page 119 earlier were all the buildings have the same adjacency graph, yet their architecture is different. Regardless of the architecture, weather it is constructed using triangles, rectangles or circles, the connection network they inherit is identical. This graph induced analysis was first described by Lionel March and Philip Steadman.065

To overcome the above mentioned problems, a number of additional criteria definitions have been

065 Coates, Programming.architecture, 161.
However, these computational tasks have turned out to be exhausting and time consuming. On the second level, the program proceeds to fulfill the requirements by first selecting nodes on the boundary of the envelope and assigning dimensions to them from the design requirement list. It then performs an exhaustive search of all possible design solutions by filling the envelope with possible edges allowed by the design requirements. The resulting output is all the possible generalized mappings of spaces, some of them are shown in figure 26.

Figure 25: Graph illustration of program adjacencies that can take multiple different planar forms.

---

066 Homayouni, ‘A Literature Review of Computational Approaches to Space Layout Planning.’ 4
067 Ibid, 8.
Permutational Space Allocation

To solve the difficulty of global adjustments and relocation of once placed spaces, permutational methods were developed. In general, these methods allowed spaces to be swapped indefinitely and adjusted compositions were now formed. Even though systematic swapping of all the spaces again resulted in far too many iterations and thus consumed excessive amounts of time, permutational methods had progressed to form more coherent wholes, that is aggregates of spatial elements or floor plans, than additive methods. Permutational methods became more practical by giving up some of the systematic accuracy since, in many cases the outputs of such programs were still subject to extensive manual drafting before they represented clear floor plans as seen in figure 17 presented earlier on page 122. The illustration is originally from an permutational

prototype developed by Edna Shaviv and Dov Gali that can be considered a precedent for the method as it consists of an objective function, set of constraints and consecutive phases. The objective function measures the weighted sum of distances between activities and seeks to minimize it. Various different constraints and requirements of objective function include architectural constraints, the allocation of given activities to specific floors, requirements for prescribed orientations such as natural light, treatment of noise and other disturbances and such. The prototype is run in 2D, however it understands the elevations of each 2D layer. The following is a description of the process of this prototype.

First, all activities defined by the projects architectural program are represented by equal-sized squares that are usually randomly spread in an orthogonal grid as in figure 27.1. Now activities are mutually interchanged to gain a better results in the objective function. If the weighting between units of a given activity is significant, units are kept together. However, if the weighting between them is not significant compared to the weighting between given activities, the units of might split into parts (see figure 27.2). The overall grid can also consist of more cells than the set of activities would require. In this manner, the envelope for the building to be formed can move freely as it is not restricted by the outlines of a packed grid. In general, this phase is used to create a layout for the next phase to allocate geometry.

Second, the layout provided is detached so that no overlap occurs when actual areas are placed. Then two repeated steps are carried out: contraction and interchange. In contraction the areas are drawn together. Then the interchange step proceeds to swap activities similarly as in the point model, except now, as they are given geometrical areas, they can be rotated by 90 degrees or their proportions altered. If the objective function can not produce higher quality results, that is reduced values, with new interchanges and adjustments, the step is rejected and an alternative interchange

071 Ibid, 35.
073 Ibid, 35.
074 Ibid, 36.
step is carried out.\textsuperscript{075} If all possible permutations have been tried, note the systematic nature, old layout is preserved. Lastly, the actual floor plan is drawn freehand in accordance to the layout as seen in figure 27.3.

<table>
<thead>
<tr>
<th>13</th>
<th>08</th>
<th>01</th>
<th>10</th>
<th>01</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>05</td>
<td>01</td>
<td>03</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>02</td>
<td>06</td>
<td>01</td>
<td>09</td>
</tr>
<tr>
<td>01</td>
<td>07</td>
<td>01</td>
<td>08</td>
<td>05</td>
</tr>
<tr>
<td>06</td>
<td>01</td>
<td>04</td>
<td>13</td>
<td>01</td>
</tr>
</tbody>
</table>

Figure 27\textsuperscript{076}: A plan generated according the simplified grid of program. The lines in the last image are representing walls.

\textsuperscript{075} Ibid, 39.

Heuristic

“Whether you state them or not, you often have answers to questions that you do not completely understand, relying on evidence that you can neither explain nor defend.”

- Daniel Kahneman

Heuristic methods — unlike procedural — are reasonably likely to find a reasonably acceptable solution in a short time, yet cannot guarantee such an exhaustive search process. They are often influenced by analogies, and their procedures are constructed to mimic the accumulated knowledge and tradition of architectural profession on how to confront design issues. By being approximate, they avoid being as blatantly systematic as the procedural methods.

Analogical methods

Most of the analogies in analogical methods are derived from physics. Geometry is generated by means of simulating objects’ movements by using forces that move them rather than manipulating objects geometry through direct transformation.

077 Kahneman, Thinking, Fast and Slow, 97.
operations such as mirroring, rotating or scaling. The appliance of forces is usually praised for its responsiveness, allowing the designer to easily make manipulations of which consequences can be immediately seen to propagate throughout the design.\textsuperscript{081} In this sense, by having the property of explicitness, it is immensely similar to design models characterized as parametric. Physics influenced methods often combine the speed of computation with a human friendly interface, which usually translates into graphical interface. However this notion of intuition is extended to apply the manner in which the model proceeds. For example the analogy of forces are often accompanied by the analogy of springs, as the springs can be easily understood to incorporate different properties found in actual springs, such as spring constant, damping constant and rest length.\textsuperscript{082}

A significant precedent to showcase the methods using the analogy of physics was developed by Scott A. Arvin and Donald H. House. In their model relationships between objects are defined by forces moving them either to repulse or to attract each other.\textsuperscript{083} In this prototype, nodes, similarly as in the dual graph, are the underlying components used to define the position of polygons. Nodes in turn define boundary shapes for spaces which form the basic parts for the model to build on to.\textsuperscript{084} Figure 30 illustrates this overall schema.

There are two types of nodes: the point node and the line nodes, both of which are shown in figure 28. Point nodes are used to define the centers of spaces. They are not constrained by any orientation and thus forces can move them in any direction. Line nodes, on the other hand, are constrained to move only perpendicularly to the line of a polygon it defines. Therefore they are used to to define the edges of spaces, the polygons. Springs are then used to connect the point node and the line nodes, and thus the rest lengths defines the shape.\textsuperscript{085}

\textsuperscript{081} Arvin and House, ‘Modeling Architectural Design Objectives in Physically Based Space Planning.’, 213.


\textsuperscript{083} Homayouni, ‘A Literature Review of Computational Approaches to Space Layout Planning.’ 11.

\textsuperscript{084} Arvin and House, ‘Modeling Architectural Design Objectives in Physically Based Space Planning.’, 216.

\textsuperscript{085} Ibid, 217.
As defined earlier, a space consists of a node and a polygon. However, a space can also contain any number of child spaces and thus a set of nodes. A common parent node then forms a dynamic simulation system. The parent node, exactly like a typical node, can have different envelope boundaries. A parent node is shown in Figure 29. This system of hierarchy can be seen analogous for example to an office building floor which is compiled of departments, which in turn are compiled of rooms, which finally are compiled of objects.\footnote{087}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure29}
\caption{Simulation systems (with and without a boundary) consisting of boundary shapes. The lines are representing walls.}
\end{figure}
A set of design objectives — topological and geometrical — are then used for solution generation. For purposes of this research, the logic in which these objectives proceed is unduly detailed to generate relative knowledge. An overall view of them will suffice. 089

Topological objectives all apply a vector force to define the location of spaces relative to each other. They then differ by the means in which the direction of the vector is defined. These objectives include adjacency, separation, orientation, interior, and exterior objectives to guide solution search. 090

Geometric objectives all apply a vector force to define the dimensions of boundary edges. Unlike topological objectives, geometrical objectives differ from each other in various ways, however in general, they tune the overall system into a coherent whole. The objectives include alignment, offset, area, proportion and gravity. 091

The simulation can start when a set of spaces, objectives and their variables has been defined. It first resolves the topological relationships of spaces. Though, during this phase the boundaries are represented as circles in order for them to avoid each other before they are translated into polygons. Then the process defines the geometric positions of the boundary edges. After the simulation it is still possible for the designer to interact with the model itself, instead of adjusting the initial variables. 092

089 Ibid, 217.
090 Ibid, 218.
092 Ibid, 222.
Figure 30: Collections of elements which are representing spatial areas, not necessarily walled rooms.

Another well known precedent of an analogical method, usually merited to Philip Steadman is the one which takes advantage of the *Smith diagram* i.e. the electric flow graph.\footnote{Mitchell, Computer-Aided Architectural Design, 211.} It is a method that, for a person with background in architecture, would first seem quite unintuitive and unnatural analogy for a design method.

Smith diagram can be seen as an electrical network graph shown in figure 31. The analogy from electric physics is interpreted into architecture when the graph is overlaid with a floor plan, similarly as with a dual graph. In this way Kirchoff’s Laws for electrical networks can be used to generate floor plans.\footnote{Ibid, 211.}

Walls parallel to x-axis are labeled with *terminals*, a point of connection for closing an electric circuit, displaying the *potential* in them. This potential is defined by the distance of the terminal from the x-axis. The current flowing in the *links*, lines connecting terminals, represents the width (in direction of the x-axis) of the space it traverses.\footnote{Mitchell, Computer-Aided Architectural Design, 211.}

*Figure 31*\footnote{Ibid, 211.} The thicker lines in the illustration on left is representing walls.
Kirchhoff’s first law states that current in terminal is:

\[ \text{Current} = C (V_1 - V_2) \]

where \( C \) represents conductance, \( V_1 \) larger voltage and \( V_2 \) smaller voltage. Kirchoff’s second law states that the current entering a terminal must equal the current leaving the terminal. What this means, in terms of the floor plan, is that \((V_1 - V_2)\) represents the length (in direction of the y-axis) and \( C \) the proportions of a room. Also, the sum of room widths (x-axis) of a common terminal are equal.\(^{098}\)

This model can then be used for computation. The procedure is in general a systematic permutation of room positions within the envelope in accordance with the Kirchoff’s Laws and adjacencies between rooms. The idea is simply to exclude solutions that do not match the definition.\(^{099}\)

Analogical methods are usually the most attainable methods to approach computation simply because of their analogous nature. As derivations from natural phenomena such as electricity and spring forces they usually include the definition of equilibrium which is the terminating rule sometimes hard to conceptualise in computing.

**Case based**

Case based methods take advantage of the vast memory capabilities of computers. Expert knowledge, that is information generated by architects, stored as *cases* into computer memory can be retrieved when needed by the designer during a given project. Cases consist of a *problem definition*, a *solution* and an *outcome*. They thus create an index which can be used to retrieve the needed knowledge.\(^{100}\)

Some developments incorporated freehand sketching environment where the designer could draw certain kind of diagrams that the computer was able to recognize and then retrieve solutions from

\(^{098}\) Ibid, 212.


the case library. In other words, the case based method provides complete solutions, detailing of which is then modified by the architect to fit the current problem. Case based methods represent an insignificant minority of floor plan generation methods.

**Expert systems**

*Expert systems* incorporate knowledge much like case based method. However the knowledge embedded within them is accessed through *situation rules* which are basically *if-then* constructs. The *if* clause acts as a trigger for a certain action. That is, when a given condition is fulfilled, the action will be called. For example, a layout might be taken as an input and new objects, such as a room or a fixture, depending on the given scale, are then added to it according the predefined rules. Thus, the expert system is a kit of parts from which new combinations are formed.\(^{101}\) Much like case based methods, expert systems as well represent an insignificant minority of floor plan generation methods.

![Figure 32\(^{102}\): LOOS program. Alterations on the room shape results in different arrangement of fixtures according to situation rules.](image)

**Shape grammars**

After the case based and expert system methods came to prominence their inherent properties were increasingly seen to resemble languages. They were then formalized into certain grammatical

---


\(^{102}\) Ibid, 272.
methods which were later characterized as shape grammars. However, shape grammars are sometimes acknowledged as a category of their own since the grammatical property in them was developed as a treatment to the excessive production of unfit designs produced by procedural approach. In this interpretation of history shape grammars are described as higher order syntaxes. There is no single specific case of shape grammar floor plan generation since shape grammar methods are widely used in many prototypes. Shape grammar are often accompanied by a metahueristic framework. However as they can be used without it they are presented here rather as a general category of an approach.

A grammar, as stated throughout this research, is essentially a set of rules which systemize a certain process. The rules act as clauses to initiate a transformation. Thus consecutively applied rules to a geometrical form with the help of geometrical transformations is then characterized as being a shape grammar. The rules can be constructed using geometrical objects and forms such as points, lines or volumes. The geometrical transformations then can take place for example as addition, subtraction, rotation or mirroring of geometries.

To be specific, shape grammars can be described as a data structure consisting of five factors. First, a set of shape transformation rules must be defined so if a given shape is found, it can be substituted by a predefined new shape. Second, a set of labels are defined to control the application of rules. Third, a set of transformations are defined under which rules can be applied. Forth, a set of functions that assign values to parameters in rules is defined. Finally, the initial shape and termination rule needs to be defined.

---

103 Ibid, 272.
104 Coates, Programming.architecture, 166.
105 Stiny and Gips, ‘Shape Grammars and the Generative Specification of Painting and Sculpture.’ 128.
Figure 33: Transformation rules, initial shapes and transformations that can be used to generate floor plan elements. Although there is no floor represented here, the solid lines could be used to represent one.

Advanced

“... computer simulations may be thought as occupying an intermediate position between that of formal theory and laboratory experiment.”

- Manuel DeLanda\textsuperscript{109}

Methods in this chapter seek to apply novel design processes that are not induced by human cognition. This means, that they attempt to generate geometry and find solutions with methods that are derived outside the limitations, and advantages as well, of human intelligence. This kind of methodology is possible, however the ability to recognize the unexpected solutions produced by them is more problematic.\textsuperscript{110}

The methods used to pursue novel designs are often compared to the processes of emergence which are found in the natural world: properties found in a \textit{whole} could not have been deduced from its original individual \textit{parts}. For example “... water has capacities distinct from those of its parts: adding oxygen or hydrogen to a fire fuels it while adding water extinguishes it.”\textsuperscript{111}

Similarly in computation, properties observed in a simulation

\begin{flushright}
\textsuperscript{109} DeLanda, Philosophy and Simulation: The Emergence of Synthetic Reason, 35.
\end{flushright}

\begin{flushright}
\textsuperscript{110} Kalay, Architecture’s New Media: Principles, Theories, and Methods of Computer-Aided Design, 278.
\end{flushright}

\begin{flushright}
\textsuperscript{111} DeLanda, Philosophy and Simulation: The Emergence of Synthetic Reason, 1.
\end{flushright}
could not have been deduced from the definition of the algorithm running it. To put it differently, algorithms operating continuously in parallel give rise to the higher order of observable phenomena.\textsuperscript{112}

For example, old or primitive cities have usually been purported as clear evidence of emergent order found in a design: the actual structure of the city is defined by a myriad amount of forces affecting it. Such forces could be interpreted to be the agents traversing through it, the amount of the population, cultural requirements, local environment and weather, manner of commerce and such. This chapter, however, proceeds to analyse methods that could shape individual buildings, namely their floor plans. The whole here is then considered to be the floor plan and different permutations on elements, that is rooms, to produce it as its parts.

Unlike with primitive methods, there is no clear subcategories found within advanced methods as they are all more or less experimental, and since the property which made them distinct in the first place has departed into a research field of its own: the metaheuristics. Despite this illogical chronology, the precedents are attainable even without the throughout knowledge of the metaheuristics accompanying them, it is the method of constructing geometry that is of interest for now.

Therefore the precedents described in this chapter are somewhat hybrid or transition prototypes between two fields of research. Their approach to creating geometry is emphasized in this chapter, while the metaheuristic computation is only superficially explained since it has no straightforward connection to the methods of creating geometry. Metaheuristics are a major subject in the following chapter.

\textsuperscript{112} Coates, The Architecture of Programming, 9.
Figure 34: Room(s) morphing into a building.

Genetic Engineering Approach

Algorithms inspired by the evolutionary inheritance of genes, namely genetic algorithms, were one of the first advanced methods used to create novel design. Today, they are considered a subcategory of evolutionary metaheuristic algorithms.

The precedent discussed next was developed by John Gero and Mike Rosenman. Its approach of generating geometry is only superficially described as it relies heavily on metaheuristic properties. However, it is relevant here to show the overall process to understand what novel geometry representation might look like.

In its core is the idea of making complex genes from basic genes and further using those genes to create evolved genes. Basic genes are seen as operations relating to the creation of geometry; they draw a line in the given current direction, move the pen ahead, or change the current direction. Thus they are essentially transformations of geometry and examples of them are shown in figure 35.

Next, a function is set to rank individuals. (To be specific, it is a fitness function which will be described in detail in the next chapter.) For now it is enough to understand that in this example the function rewards individuals depending on how much of the given reference floor plans, which are shown in figure 36, they resemble.
Figure 35: Example transformations that can result in spatial elements. The lines are representing walls.

Figure 36: Two reference floor plans or aggregates of spatial element. The lines are representing walls.

Random transformations form the initial population, which then evolves through a number of iterations of selection according to the fitness function towards more nonrandom population of transformations. Simple transformations start to compose more evolved transformations, however, in the end most of them are composed of combinations of evolved or basic transformations, genes. An evolved transformation later becomes an atom representing lower-level genes and as such it shields them from the disturbance of genetic operations.

After a representation based on the examples has been developed, it is used for new requirements of quality such as that no walls that do not build a closed room are allowed, and a total of 6 rooms consisting of sizes 300, 300, 200, 200, 100 and 100 units while keeping the overall wall length to the minimum. The finished floor plans are shown in figure 37.

---

118 Ibid, 17.
120 Ibid, 16.
121 Ibid, 16.
122 Ibid, 18.
Figu 37: The resultant floor plans or aggregates of spatial elements. The lines are representing walls.

In 2004 Patrick Janssen developed a generative evolutionary design framework to generate multi-story housing buildings. Although the design approach is capable of producing three dimensional buildings, a part of it is dealing with two dimensional floor plans. The geometry in this precedent is generated using a grid which a metaheuristic construction, an evolutionary algorithm, is evaluating. In general the form generation process consists of eight consecutive stages, which are illustrated in figure 38. The transformations taking place during each step are always induced by the metaheuristic algorithm as transformations that increase the quality of design are allowed to take place. The steps are described in the following.

124 Janssen, ‘A Design Method and Computational Architecture for Generating and Evolving Building Designs.’, V.
First the grid is positioned. It can be placed in relation to the site and rotated. Second, the faces of the grid are stretched by moving the grid lines. This can happen orthogonally to the grid. Third, the inclination of the envelope elevations are defined. The grid itself cannot be rotated. Fourth, a staircase is inserted. This position is defined by the grid and three constraints: it cannot touch the outer wall, its minimum measurements are met and that the area allocated for it is as small as possible. If conditions for its insertion are not met, the process will abort. Fifth, the actual spaces are created based on the grid faces. Certain faces are joined to form spaces according to the architectural program. The merging is done according to multiple constraints. Sixth, the outside spaces are defined. This is done in similar manner as with inside spaces. Seventh, the doors are placed. A door is constrained from being in the middle of the wall and it can be placed near the corners. Eight, the windows are placed. There are various methods for doing this.

The prototype developed by Janssen is one of the most complex and advanced methods to generate floor plans. The process described here is only dealing with the floor plan generation, however Janssens model is also capable of generating three dimensional forms with similar logic.

---

126 Ibid, 209.
128 Ibid, 211.
129 Ibid, 212.
Figure 38: Eight steps of the design process. The general grid is representing the division process. Actual walls are represented after the fifth step as in traditional architectural drawings.

Hierarchical Growth Approach

The first prototype described here proceeds with shape grammar rules which are implemented according to an evolutionary algorithm. The grammar is based on a vector construction that draws closed polygonal shapes. This definition incorporates different methods of transformations. The prototype was developed by Rosenman and Gero, however a varied version of it is designed by Hoda Homayouni.

In this prototype a floor plan is composed of three hierarchical levels, which are, from bottom to top: room, zone and house. Figures 39–41 illustrate them. Each level is defined by objects found on the levels directly below. To be specific, each level is a population of good quality objects which are selected from overall collection of objects from the level below. This selection is always made according to a level specific function assessing quality. The three levels form the following hierarchy: The house level consists of zones, such as living zone, bed zone, utility zone, while zones are composed of rooms, such as living room, dining room, bedroom. Finally each room is composed of a number of space units, that is, a constant unit.

![Population of rooms](image)

Figure 39: Population of rooms.

134 Ibid, 43.
135 Ibid, 19.
137 Ibid, 20.
Figure 40: Population of zones.

Figure 41: Population of houses.


As stated before, each of the three levels has its own specific function assessing quality. First, the function for room level seeks to minimize the perimeter to area ratio and the number of angles. Second, the fitness function for zone level seeks to minimize the sum of adjacency requirements between rooms. Finally, the fitness function for house level seeks to minimize the sum of adjacency requirements between rooms in a zone. This in such a manner a floor plan is generated.

Reinhard Koenig, Katja Knecht and Sven Schneider have recently continued to developed methods that deal with subdivision — the division of a predetermined area into divisions — and dense packing — the problem of arranging elements within a given space without any overlap or gaps. Subdivision methods could be characterized as procedural and dense packing as permutational. However as these prototypes incorporate metaheuristic constructs they are placed under advanced methods. These methods also complete the prototype review in this research. In the following the subdivision method is explained first, followed by the dense packing.

The process of recursive division of rectangular areas into smaller rectangular areas, namely the applying of lines parallel to the edges of an area, is defined as the subdivision method. The point of origin for a line subdividing an area can be selected randomly or according to a rule. The subdividing line originates from the longer edge and a fixed proportion ratio of areas is retained. The origin point can also be determined in a manner that subspaces end up having the same area. The advantage of the subdivision method is that there can not be any overlap between elements in similar sense as with dense packing. As such, the fundamental issue of subdivision seeks to solve is topological, namely the neighborhood relations of elements.

Figure 42.1 illustrates a case of a program on which certain weightings that determine the subdivision are imposed. The alleged leaves in the tree diagram are used as the starting point.

140 Ibid, 20.
141 Koenig and Knecht, ‘Comparing Two Evolutionary Algorithm Based Methods for Layout Generation: Dense Packing versus Subdivision.’ 286.
142 Koenig and Knecht, ‘Comparing Two Evolutionary Algorithm Based Methods for Layout Generation: Dense Packing versus Subdivision.’ 287.
for calculations of the *split values* or the element ratios. The ratio between a desired size of an element and the average element size (dividing the total area by the number of rooms) is used to weight a given area. The sum of the weightings of direct branches constitute the weighting of a node. Thus all the weightings for the elements and nodes are defined bottom up. The development of a corresponding floor plan is figure 42.2.\textsuperscript{143}

In the given case of six rooms totaling an area of 75 m², the average element area is 12.5 m² thus a room with 15 m² has a weighting of 1.2, a room with 10 m² has a weighting of 0.8, a room with 8.33 m² has a weighting of 0.67, a room with 16.67 m² has a weighting of 1.33, and the total area of 75 m² equals 6.\textsuperscript{144}

*Figure 42.1:*\textsuperscript{145} The tree diagram is a representational tool for the division logic. Below the orthogonal illustration is representing the actual plan and its lines are representing walls.

\textsuperscript{143} Ibid, 288.
\textsuperscript{144} Ibid, 288.
\textsuperscript{145} Koenig and Knecht, ‘Comparing Two Evolutionary Algorithm Based Methods for Layout Generation: Dense Packing versus Subdivision.’ 288.
Dense packing prototype builds on the prototype developed by Scott A. Arvin and Donald H. House in which forces or springs were used to pack elements. The prototype was presented on page 137. The reconditioned precedent has much similarities to it however one property is of clear significance.

Dense packing can be defined as a sought condition where an aggregate of spatial elements within a given space is packed together with no overlap or gaps between elements, i.e. the metaheuristic construction proceeds to minimize the sum of all overlapping areas. The distinction of this prototype is its ability to swap program between elements. Figure 43 illustrates this. This interchangeability and the consequent change that takes place in the overall arrangement is called a permutation. In this case it is achieved using an index of element program and rearranging that indexed list.

This concludes the study of precedent prototypes. No matter how advanced and complex certain prototypes might have seemed it should be noted that all of them are still in the research phase. It remains unclear how complex tasks, that is how complex

---

146 Ibid, 288.
147 Koenig and Knecht, ‘Comparing Two Evolutionary Algorithm Based Methods for Layout Generation: Dense Packing versus Subdivision.’ 286.
148 Ibid, 135
architectural spatial briefs, can they solve in an adequate time. This is critical since what one would expect from these models is the ability to generate novelty out of chaos that the human designer is unable to read. That is, humans are capable to solve simple architectural issues quite efficiently so what would complement this is a tool that is able to, at least heuristically, general schemes of design out of a large set of information. This ambiguity of properties in precedent models further strengthens the need to carry out a model demonstration in contrast to the literature review since specific knowledge and criticism is hard to come by according to textual studies of these models.

Despite the above, there is still much research that indicates that at least some of the prototypes are efficient and capable of producing coherent design if not novelty. As it was revealed during the research there exist higher order algorithmic constructs the structures of floor plan generation models that require more in depth look, the next chapter will thus describe these metaheuristic algorithms.

Figure 43: Permutation, or swapping of program between elements. The dotted lines represent program adjacencies. The solid lines are representing walls.

IV. Computing solutions

“Actually, the potential appears to be virtually infinite. It is leading us to compositional schemes which we could have never conceived on our own, but the computer is able to unfold for us. And yet, we program the computer; we told it what to do.”

- Chris I. Yessions

Thus far this research has clarified the common misconceptions or vagueness plaguing the digital scene of architecture, unfolded the general history of computing, and presented the significant precedents of floor plan generation. As observed in the precedent analysis of the previous chapter, there is a strain of models — namely the models described as advanced in this research — that has departed from simplicity and incorporated higher-level algorithmic constructs into itself. It is precisely these constructs that are ever so often dismissed in architectural publications as something that was merely applied or used, and as a result a novel design emerged. It seems that the idea of the computer as a tool, rather than as a methodology, is well ingrained into our minds. We think of applying algorithms analogous for applying fixed solutions to a defined problem. These kind of references to algorithms purports them as something absolute and singular, even mystified. The algorithms are presented as something blatantly objective and without bias. The fact, however, remains that algorithms are constructed and designed by human intelligence and thus, by nature, are inherently subjective takes on the given issue. The precedents for constructing geometry,
described in the previous chapter all had a specific take on how to solve the issue of floor plan generation and all of them generated differing designs. Similarly, all higher-level algorithmic constructs have a specific take to solve the issue of computing solution, and all of them will have differing results. Therefore, this chapter will describe the processes of computation, namely the processes a metaheuristic algorithm and a fitness function. They constitute the remaining two main components of the process of generating solutions, namely floor plans, before solutions can be generated. Figure 44.1 below once again highlights the research progression and figure 44.2 on page 162 recaps the overall process and framing of the research.

Figure 44.1: Process of research structure.

The structures of metaheuristic algorithms and fitness functions are complicated, and to tangibly explain their abilities and properties, a computational model framework will be constructed iteratively from ground up, by starting from simple constructs and advancing to more elaborate frameworks. This process is essentially twofold.

First, the basic combinatorial nature of the geometrical, architectural model needs to be constructed. Essentially, it focuses on the process of how the geometrical model is proceeding from an undefined state through intermediate states finally into a desired state. This is mapped using abstract illustrations since the process has no definite representation. The descriptions of model states are then accompanied with structures, that allow the ranking of the model states. This includes concepts, such as the fitness function and fitness landscape, which are described in detail. Up until this point it should be noted the model does not have any inherent logic of operation. It remains merely as a framework capable of projecting
good and bad design solutions alike. It does not have the properties or abilities to search for them.

Second, the kinds of search processes that are able to search for a desired solution out of the above described framework, namely metaheuristics, will be the subject of the latter part. This includes a general introduction to some of the most fundamental metaheuristic algorithms. All of the categories are not fully explained, since thorough comprehension of all of them is not a prerequisite for applying a metaheuristic.001 It should be remembered, that one of the original framings of this study was to find a single general method to carry out a generation of designs and thus, only an introduction to the field of metaheuristics suffices in this research. From all of the metaheuristics, the category of evolutionary algorithms is chosen for further analysis as it is the most widely used and commonly known category.002 Then, one generic solver, namely the Galapagos, includes an evolutionary algorithm is explained in detail, since it will be used in the floor plan generation demonstration in chapter five. Galapagos is a simple software development that is easily attainable, as it is integrated into the visual algorithm editor Grasshopper, which is also highly popular among architects. For these reasons it is suitable for this research as it can be promptly harnessed without an extensive knowledge of computation in general.

This chapter concludes the description of the construction of computational framework. All the general information is now considered to be provided for starting to construct a framework for generating floor plan designs.

---

001 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 14.
002 Ibid, 11.
Figure 44.1: Recap of the research framework.
Illustrating the model setup

“Effective searching procedures become, when the search-space is sufficiently large, indistinguishable from true creativity.”

- Richard Dawkins

All the models described in the previous chapter took information as an input and processed it in a specific way to form geometry, namely a floor plan layout. Thus, the purpose of the previous chapter was to discuss the methods for geometrical and visual representation. However, the processes that allowed those models to take shape were intentionally not emphasized. For example, the fitness function and other metaheuristic properties were merely acknowledged. This chapter is about elucidating those computational processes. To do that, a simple general model — a model generating rooms — is used as an example throughout this chapter to show the computational logic. Figure 45 on the page 165 illustrates this model.

In the model three variables are used to represent length, width and height of a room. The room can then have different states, combinations of scales and proportions as the inputs are changed. In this case the model is a construction of three vectors in directions of x, y and z axis. The model described thus has three inputs, d1, d2 and d3, and one output, the formed geometry, the actual room. Depending on the numeric domain and the step size of progression the model can produce different numbers of outcomes.

For the sake of an example, let us assume that the x and

003 Dawkins, The Blind Watchmaker, 66.
y vector constructing the floor area can be minimum of three meters and maximum of six. This means that the room is then at least three meters wide and maximum of six meter wide. There is no combinatorial constraint of the floor area. Then also the z value should be minimum of two and a half meters tall — quite commonly considered the minimum for a decent room — and the maximum of four — just to be modest. Let's also say that the step size between different values is half meters in order to clarify the differences between the produced rooms. This then means that the domain for x and y measurements has six possible parameter values and the room height four. Consequently, this means that there are 144 different combinations (since \( 6 \times 6 = 144 \) ) of measurements and thus 144 different rooms could be generated using the model.

In this kind of a simple computational model the designer could vary the input values for the given room measurements and quickly see the resulting design, the room. This kind of setup, can be characterized as a parametric or generative model, as described earlier. The issue one notices quite easily is how large the sheer amount of possible room designs is in comparison to the amount of parameters responsible for it. Needless to say, this is an overtly simple model with very small domains of parameters. More complex models result in more complex lineages of design solution and thus different methods to describe and map them have been developed. They are thus described in the following paragraphs.
Figure 45. A room with three input parameters.

State-action graph

The different states of a model, in the simplest form, can be described using a state-action graph consisting of lines and vertices that form a tree structure. In the graph a point represents a potential state of the model and each line a performed design action — a changed variable value — leading to another state. However, it should be noted that points in between lines constitute for incomplete solutions as the final parameter values remain unset. The state is synonymous to a design and the reason to describe a design as a state here is because it better represents the fact that it is a potential design, a state of a model according to certain variables.

In figure 46, on the opposite page, an example of a state-action graph for the room example generated by vectors x, y and z is shown. As previously stated, x and y can both be given six different values (the step size being half a meter), and z can be given four (the step size being also half a meter). In the example the z parameter is first set but it could be any of the other parameters as well. Then the x and y parameters are set consecutively. Again, they could be swapped. The state-action graph is illustrative to show the complexity of different outcomes of a generative or parametric model. It also pinpoints the variables responsible for a certain outcome and thus makes the design process explicit.

Figure 46: Six possible measures for $x$ followed by six possible measures for $y$ and finally four possible measures for $z$, the height results in 144 different rooms.

Phase space

A *phase space* — sometimes called *possibility space* or *state space*[^007] — is a collection of all terminal solutions, namely the ones that correspond the final designs in a state-action graph[^008]. Instead of a tree structure representation, a phase space representation is more traditional, using points, lines, and curves. Regarding the terminology, if a design in state-action graph was referred to as a state, now with phase space a more formal title to be used for a design would be a *tensor*.[^009] Figure 47 shows two example phase space diagrams.

Yet, a phase space is much like the state-action graph. It can be used to illustrate all possible terminal solutions of the model and in general revealing the relations of the input variables. However, it does not include intermediate, that is incomplete, solutions in its representation. Phase space is implicitly necessary concept to understand since it is used in the construction of fitness landscape, which in turn is extensively used to describe the computational processes in general.

Phase spaces can have varying dimensions. Each variable input represents a topological dimension of a phase space. These dimensions are sometimes referred to as the *degrees of freedom*, as they represent the allowed variability in the model.[^010] A tensor is a point or a coordinate in a phase space, that is constructed using the variable inputs.[^011]

A single dimension phase space would belong to a model constructed using one input variable and thus the outcomes — complete solutions produced by the model — would be mapped as points along one dimension axis.[^012] This is usually uncommon as

[^007]: Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking. 146.
[^008]: Rutten, ‘Galapagos: On the Logic and Limitations of Generic Solvers’. 133
[^009]: Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 4
[^010]: Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking. 144.
[^011]: Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 4
[^012]: Rutten, ‘Galapagos: On the Logic and Limitations of Generic Solvers’. 133
the information it provides is insignificant since the same can be deduced otherwise without the visual representation. For example, in the described simple room model if the room would have been constructed using only one variable, let’s say the x dimension, which would be given a domain of values between 3 and 6 meters with a step size of half a meter, the five floor areas (variable x multiplied constantly with four meters for example) would be plotted as five states along one dimension axis.\textsuperscript{013}

A two dimensional phase space would belong to a model that is constructed using two variables and the solutions would be plotted as series of points — that is curves or trajectories — on a two dimensional plane.\textsuperscript{014} Again, going back to the room model example, this time the model would be constructed using two variable inputs, x and y, and thus they would each prompt a dimension in the graph. The room height, z direction, would remain constant. The produced rooms, the solutions, are then plotted as points in space that construct a curve in a two dimensional plane shown in \textbf{figure 47.1}.

Following the same logic, a model with three variable inputs would constitute a three dimensional shape space. In addition to width x and length y, the height z is considered as an input as well. Thus, the rooms are again plotted, now on a three dimensional curve as in \textbf{figure 47.2}.

There is no theoretical maximum for the dimensionality of a phase space\textsuperscript{015} and thus their repertoire includes n-dimensional hypercubes.\textsuperscript{016} This implies that it is neither possible to represent them in the above mentioned manner nor is there any way that would project them in a comprehensible way for illustrative purposes. What is of interest for the purposes here is the two dimensional phase space diagram. The following will explain why.

\textsuperscript{013} Rutten, ‘Galapagos: On the Logic and Limitations of Generic Solvers’. 133
\textsuperscript{014} Menges and Ahlquist, Computational Design Thinking: Computation Design Thinking. 144.
\textsuperscript{015} Rutten, ‘Galapagos: On the Logic and Limitations of Generic Solvers’, 133.
\textsuperscript{016} Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’
Figure 47.1: Two-dimensional phase spaces.

Figure 47.2: Three-dimensional phase space.
Fitness landscape

Phase-space does not value individual tensors into any rankings, much like a state-action graph that does not rank states. Neither one of them can be used to separate good solutions from the bad instead they merely map the possible designs and provide an understanding of the potential of computational framework of the model. This is why a fitness function needs to be set up in order to compute the desirability between different outcomes of the model, and thus allow a distinction or a ranking to be made between designs, that is solutions or tensors.

The output of a fitness function will be a single numeric value\textsuperscript{017} — the fitness value — and a desired value can either be as large as possible or as small as possible. This can be decided beforehand by the designer. The fitness function itself can be defined to consist of any kind of variables and mathematical operations.\textsuperscript{018}

Since the user is dealing with a specific problem — such as the example of generating a room — the definition of fitness or a fit solution is entirely defined by a function set up by the user. For example, in the room model example, one might be interested in finding a room where the relation of the envelope and the volume is minimized. A simplified fitness function for this property would then translate into:

\[
\frac{(2xz + 2yx + A)}{xyz}
\]

The values produced by this function would give means to identify good states and the input values responsible for it. It is these fitness values that are used to construct the fitness landscape which in turn is used in the actual search for the most fit fitness value. However, a model with three input variables (x, y, and z) is already too complex to be used in a fitness landscape illustration.\textsuperscript{019} Thus the example needs to be simplified so that the room height z remains constant. In this manner it is possible to construct a fitness

\textsuperscript{017} The solver would then be set to find a fitness value as small as possible produced by the above described fitness function. 3
\textsuperscript{018} Rutten, ‘Galapagos: On the Logic and Limitations of Generic Solvers’, 133.
\textsuperscript{019} Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
landscape. This is presented in the following.

Using a two dimensional phase-space, a fitness function, and the fitness value it produces, a fitness landscape can be constructed. A two-dimensional phase space on a plane is shown in figure 48. The individual tensors — that is the individual room designs — exist in a grid defined by the variable 1 along x-axis (the room width) and the variable 2 along y-axis (room length). The function responsible for the volume or floor area of the room is not visible in the illustration. The vertical lines in the illustration (that is along z-axis) represent the extrusion height according to the fitness values of the given tensor. Once all tensors have been extruded accordingly, they create a fitness landscape. 020

Figure 48: Extrusion of fitness values to from a fitness landscape.

021 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 5.
In a fitness landscape the peaks represent better quality states, whether fitness value is sought to be maximized or minimized does not matter, and valleys the low-quality ones.\textsuperscript{022} This then implies that planes, mountains, and valleys are the general possible characteristics of a fitness landscape, whereas horizontal caves, holes, and overhanging cliffs are not possible.\textsuperscript{023}

\textbf{Figure 49.}\textsuperscript{024} A fitness landscape

\textsuperscript{022} Rutten, ‘Galapagos: On the Logic and Limitations of Generic Solvers’, 134.

\textsuperscript{023} Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 5.

\textsuperscript{024} Rutten, ‘Galapagos: On the Logic and Limitations of Generic Solvers’, 134.
As it is now easy to ascertain, if there would any additional variables, the resulting landscape would be four dimensional and thus impossible to illustrate. Volumes more than three dimensional are also referred to as hypervolumes. Despite this drastic shortcoming, since most of the models have more than two input variables, the fitness landscape is widely used as an descriptive illustration of the processes of computing designs.

It should be noted that there is still no abilities integrated that could allow the search for best fitness values. All that is possible this far is to assign fitness values to given designs and thus begin to map the fitness landscape. It is the ability of a given metaheuristic to navigate the landscape. Each metaheuristic has an ability of their own to proceed with navigation. In the following the manner in which a landscape can be read is presented but no decision on what kind of a metaheuristic will be used is made.

In the beginning of the search for a fit solution, the terrain of the landscape is completely unknown, and it will never be totally computed as it would use excessive amounts of time. Were the landscape known there would be no need to traverse it. Again, the landscape is only an informative medium to describe the computing of those properties. There is only one way to gain information about the topology of the landscape: pick a tensor point and attach it to the corresponding fitness value.

However, in general the computer generates different solutions using variables $x$ and $y$ to produce a design which will land somewhere in the fitness landscape, as the fitness of that design is assessed by the fitness function. The computer, or the given metaheuristic, then reads the landscape and according to the terrain information, it seeks to generate a new solution that would land in a higher terrain, i.e., the solutions try to navigate up the hill according to the information read from the terrain. Ultimately, the goal is to find a solution that lands on the highest peak in the landscape — the global optima — while avoiding to be stranded on a mediocre peak — the local optima.

---

025 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
026 Ibid.
027 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 6.
028 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
Another way to describe the navigation in the landscape is
that every solution is attracted by the peaks and thus pulled uphill
because of the ability to change course according to the terrain
information. It also means that there are basins of attraction around
every matching peak. Figure 50 illustrates these. This basin is an
area consisting of points that will converge upon a specific peak.
Yet peaks can share these areas which is problematic, since the basin
is not indicative which of the peaks, if either one, is a local optima
or the global one. The figure 51 illustrates this situation. There
are many other difficult terrain landscapes that are problematic. The
basic ones are described in the following.

Figure 50: Three converging basin areas leading to different peaks.

029 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
030 Ibid.
Landscape topologies

Certain characterizations of topologies can be used to explain the problems of navigation but also to disclose unsuspected traps in the terrain that can hinder the metaheuristic procedure from working efficiently and finding the global optimum.

Narrow peaks are approachable, however the steep rise results in a small basin, and so they are easily missed when solutions map the landscape (see figure 52). They are also unstable places to land, since in a successive navigation process even a small deviation or a step size will decrease the fitness drastically and the solution will miss the summit. An even harder version of this situation would be a peak surrounded by flat terrain which would not provide any information about the direction of sloping. Or, surrounding areas that slope away from the narrow peak giving false impression of the sloping. Also, if the multiple narrow peaks are close to each other

031 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
on a plateau, only the slope of a peak that is found first is likely to be found. The others remain hidden as the flat plateau in between them represents drastically lower fitness, and peak further away is never reached.\textsuperscript{032}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure52}
\caption{Easily missable narrow peaks.}
\end{figure}

Similar to the above mention horizontal plateaus, discontinuous landscapes (see \textbf{figure 53}), which consist of these plateaus, present another kind of a problem for the metaheuristic algorithm. For the procedure of navigation to know where to move next, it needs information that a continuous landscape, such as a clear parabolic hill, provides. Thus, when confronted with a flat horizontal plateau there is no basin, so no information directing the search. Since flat areas have the same fitness value, there is no local optima either. Navigation in this kind of terrain is then based on random choices.\textsuperscript{034}

\begin{flushleft}
\textsuperscript{032} Rutten, ‘Evolutionary Principles Applied to Problem Solving.
\textsuperscript{033} Ibid.
\textsuperscript{034} Ibid.
\end{flushleft}
Terrain with a lot of steep hills and slopes — a noise saturated terrain (see figure 54) — although being continuous has an overwhelming amount of information. Thus, they are the worst terrains for the metaheuristic algorithms, as they scatter the procedure momentum and it is almost impossible to steer the search, since even small deviations will arrive at inconsistent fitness values. It is also easy for the procedure to get stuck in local optima.

Figure 53. A clear continuous landscape and a discontinuous landscape with a plateau.

---

035 Rutten, ‘Evolutionary Principles Applied to Problem Solving.
However, there is a definite set of solutions, that is states or designs, in a model no matter how many combinations define them. In a mathematical sense, as they all have their unique place in the shape-space, the architect, instead of designing, merely has to find the right solution. This is because they all already exist in the shape space. So, finding the specific design is extremely difficult since there is a myriad set of them, that is there is a vast shape-space. It is not feasible to systematically process every solution that exists. Thus, a creative process is needed to find the solution in a feasible time. In the following, these different methods are discussed on a general level.

---

037 Rutten, ‘Evolutionary Principles Applied to Problem Solving.
038 Dawkins, The Blind Watchmaker, 66.
Model search

“... there is much in common between the development of computer programs and architectural design: both are scientifically based, yet their successful accomplishment is a matter of art.”

- Yehuda E. Kalay

When a computational model is described from the point of view of a fitness landscape, it is not hard to understand how it can now be seen as a search process rather than a design process for a given state. To bypass the infeasibility of using brute force, different algorithmic concepts have been developed.

One of the purposes of this chapter is to prove to the reader familiar with architecture, that algorithms can be constructed with different methods and properties, and that no one procedure can easily be classified as the best. Similar subjectivity is found in the design of algorithms as in the design of buildings. That is, certain properties are emphasized while others are subjugated.

The scope of problems

The architectural problems exhibit a high degree of computational complexity. The problem this thesis is pursuing to solve, namely the

---

floor plan generation, belongs to the group that can be characterised as the one where a certain "minimum length connecting a set of locations" is sought. This kind of problem is also more commonly referred to as the Steiner tree problem, or the travelling salesman problem.\textsuperscript{041} As such, algorithms computing them may require excessive amounts of time.\textsuperscript{042}

It is informative to understand the severity of the problem at hand. Computer sciences have defined classes for different types of problems. To simplify matters and to make things more understandable there is four general categories, from the quickest to solve to the slowest to solve, \textit{NP}, \textit{NP-Complete}, and \textit{NP-Hard}.

The class of \textit{NP} problems, that is \textit{nondeterministic polynomial time}, can be solved within a decent amount of time. However, as the descent is referring to a subjective take on time, it is generally thought to refer to an amount of time that is not out of scale for computing to be used.

The class of \textit{NP-Complete}, that is \textit{nondeterministic polynomial time complete}, consists of problems where there is no reliable way to find a solution. If one happens to be found, it is of such quality that it can be tested, and thus it is possible to recognize it as a solution.

Similarly, \textit{NP-Hard}, that is \textit{nondeterministic polynomial time hard}, consists of problems where there is also no reliable way to find a solution nor is it possible to recognize it as a solution, since it is unclear how to test it, if it qualifies as a solution.\textsuperscript{043}

In general, the problems presented in this research qualify as \textit{NP-Hard}. This is because the correct solution is unknown, and the method to test a design for correctness is unknown. However, the only thing that is known, or which is possible, is to compare two designs and choose the more correct one.\textsuperscript{044} This methodology of search for more correct solution or design is presented in the following. I.e. the metaheuristics which have different methods to search for designs in a \textit{NP-Hard} environment, or a fitness landscape, are presented.

\textsuperscript{041} Carranza, ‘Programs as Paradigms.’ 68.
\textsuperscript{043} Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 1.
\textsuperscript{044} Ibid, 2.
Metaheuristics

“The solver must find high-ground in the landscape before the passage of the time renders the problem irrelevant.”

- David Rutten

A certain kind of an algorithm that tries to combine basic heuristic methods in higher level frameworks to explore the search space by using different strategies is called a metaheuristic. They are designed to solve complex optimization problems, such as the example presented in the earlier paragraph.

In other words, metaheuristics are strategies set to guide the search process by efficiently exploring the search space in order to find near-optimal solutions. Metaheuristics vary in their procedures from simple local searches to complex processes with learning abilities. They are approximate and usually non-deterministic, and in some cases have abilities to traverse a difficult terrain in the search space.

045 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 6.
046 Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’ 270.
048 Ibid, 239.
049 Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’ 270.
050 Ibid, 271.

182
There are many ways to categorize or characterize metaheuristics. The most important ones are described in the following paragraphs and figure 55 plots them on a map accordingly.

Figure 55: Categories of metaheuristic algorithms.

Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’ 272.
Nature-inspired or non-nature inspired

A simple way to classify metaheuristics is according to the origin of exemplar for the algorithm: nature or non-nature. Yet it can be confusing at times, since some algorithms can fit into both categories, and in some cases it is difficult to interpret an algorithm as being caused by one of the two classes.\(^{052}\)

Population-based or single point search

The number of solutions an algorithm is processing at a time can be used to classify it, i.e. if it is working on a population of solutions or on a single solution. Trajectory methods are algorithms working on single solutions and are describing a trajectory of that, while population-based ones describe an evolution of a population of solutions, hence their names.\(^{053}\)

Dynamic or static objective function

Dynamic algorithms can modify the objective function during the solution search by incorporating information collected earlier in the search. What this means is that, unlike many other metaheuristic algorithms, dynamic algorithms can manipulate the fitness landscape, and by doing that it might help escaping the local minima.\(^{054}\)

One or various neighborhood structures

During each iteration operations are applied according to existing solutions in order to generate a new solution. These new solutions are then referred to as neighbor solutions and how they manifest as the fitness landscape is also dependant on the nature of the problem at hand. The neighborhood structure then refers to one or many new solution.\(^{055}\) The definition of neighborhood structure is used to

---

052 Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’, 272.
053 Ibid, 272.
054 Ibid, 272.
denote abilities of some metaheuristic algorithms, namely the use of a set of different fitness landscapes. This allows some metaheuristic algorithms to diversify their search, i.e. they are swapping between different neighborhood structures.056

**Memory usage or memory-less methods**

A powerful metaheuristic is often recognized by its means to take advantage of the search history, to remember There are several ways of making use of memory, however in general, the meaningful distinction is whether the algorithm uses memory or not. Memory-less algorithms essentially perform according to a Markov process — they use exclusively information present in their current state to determine the next action — in order to propel the search process.057

---

056 Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’, 272.  
057 Ibid, 272.
Evolutionary computing

“*The gene, the DNA molecule, happens to be the replicating entity that prevails on our own planet. There may be others. If there are ... they will almost inevitably tend to become the basis for an evolutionary process.*”

- Richard Dawkins\textsuperscript{058}

It is predominantly thought that evolutionary principles can take place outside the biological sphere.\textsuperscript{059} To be specific, it is *Universal Darwinism* that asserts that the evolutionary processes can emerge regardless of the medium.\textsuperscript{060} It is then not surprising that evolutionary computation algorithms are one of the largest groups of metaheuristics. A variety of evolutionary algorithms have been developed. In general they can be grouped into three categories: Evolutionary Programming, Evolutionary Strategies and Genetic Algorithms,\textsuperscript{061} of which the genetic algorithms are most widely used and well known.\textsuperscript{062} The following is an overall description that is virtually applicable to all metaheuristics in evolutionary computing.

\textsuperscript{058} Dawkins, The Selfish Gene, 192.
\textsuperscript{059} Sterling, Kevin, and Lunenfeld, Next Nature: Nature Changes Along with Us, 63.
\textsuperscript{060} Janssen, ‘A Design Method and Computational Architecture for Generating and Evolving Building Designs.’, 7.
\textsuperscript{061} Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’, 285.
\textsuperscript{062} Janssen, ‘A Design Method and Computational Architecture for Generating and Evolving Building Designs.’, 9.
In evolutionary computing, a solution — also called a state, as in the state-action graph — to a given problem is called an individual, hence a set of solutions is then called a population.\textsuperscript{063}\ Consequently, evolutionary algorithms populate individuals into the fitness landscape.\textsuperscript{064} For the first iteration, individuals emerge randomly but the ones on high ground will be more likely to be chosen to breed for their offspring to emerge even closer to the global peak. Then a new population will emerge and the cycle will repeat itself. This is the simplified logic of how evolutionary algorithms proceed to seek for a fit state of the computational model.\textsuperscript{065} Even though the above mentioned logic allows one to superficially use a metaheuristic based on evolutionary computing, there are many occasions for fine tuning. The most important ones are described below.

**Description of the Individuals**

At times individuals are not seen as intact potential solutions for the given problem. It might be that they consist of partial solutions, or a set of solutions grouped as an individual. Thus they can be converted into one or more solutions. Usually solutions are represented as bit-strings or as permutations. Other complex structures such as a tree-structure are viable.

**Evolution Process**

A selection scheme is to be defined to distinguish during each iteration the individuals qualified to be checked into the population of the next iteration. If all individuals are considered eligible, then the evolution process is described to be in a *steady state*. If only the offspring are considered eligible, then the process is in *generational replacement*.\textsuperscript{066} In general, the population size is fixed allowing at least the most fit individual in the population of the given iteration.

\textsuperscript{063} Bianchi et al., ‘A Survey on Metaheuristics for Stochastic Combinatorial Optimization.’ 245.


\textsuperscript{065} Ibid, 135.

\textsuperscript{066} Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’, 285.
The population size can also vary between iterations. Such an example could be when it is set shrink until one individual is left in the population which is set to be the stopping conditions for the algorithm.  

**Neighborhood Structure**

As stated earlier, the neighborhood structure refers to the new solution or solutions that were derived from earlier solution. As evolutionary algorithms work on a population of solutions it has means of defining which individuals are allowed to breed. A certain neighborhood function is therefore responsible for this operation as it assigns individuals. In general, when all individuals are allowed to breed with all others the neighborhood structure is referred to as unstructured populations. If, on the other hand there is some criteria for breeding it is referred to as structured populations.

**Information Sources**

Typically, crossover or recombination operators are used to recombine individuals to produce offspring. A typical way to create offspring is to use information from two parents. Thus crossover operators are being used for two-parent crossover. If information from more than two individuals is used to create offspring, the recombination operators are used to produce multi-parent crossover. There are cases in more recent developments where even the population statistics are used to generate offspring.

**Infeasibility**

Some offspring might be seen as infeasible. A simple way to handle them is to reject this offspring. However, it can be difficult

---

067 Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’, 286.
069 Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’, 286.
070 Ibid, 284.
071 Ibid, 286.
to detect feasible individuals and thus the technique to penalize infeasible offspring with the help of an fitness function is more relevant.\textsuperscript{072}

### Intensification Strategies

Even though the implementation of populations is an efficient way to explore the search space thoroughly, some algorithms may apply a local search algorithm — as an intensification strategy — for every individual in the population. These algorithms are sometimes called \textit{Memetic Algorithms}. These are used to swiftly map propitious areas in the search space. Recombination operators may also be used in an intensification strategy when apt parts of individuals are sought to be combined. This kind of procedures are then called \textit{linkage learning} and \textit{building block learning}.\textsuperscript{073}

### Diversification Strategy

To avoid the premature convergence of individuals towards suboptimal solutions \textit{diversification strategies} have been developed. The most simple mechanism is to use \textit{a mutation operator} to make the population more diverse. In general, it exposes an individual to small amount of distortion which in turn introduces randomness into the population. Other strategies include \textit{crowding}, \textit{preselection}, \textit{fitness sharing} and \textit{niching} to name a few.\textsuperscript{074}

\textsuperscript{072} Blum and Roli, ‘Metaheuristics in Combinatorial Optimization.’, 286.  
\textsuperscript{073} Ibid, 286.  
\textsuperscript{074} Ibid, 287.
The properties of evolutionary algorithms in general were discussed above. Next, a specific evolutionary algorithm implemented in Galapagos to carry out these processes is analysed.

There has been two major subject matters in this research. First, to show that computation is not something that just happens and second, that everything one does with computers is subjective: the designer chooses the methods, approaches, emphasis and finally the software. This is exactly what will happen now in this research in order to carry out the experiment in the next chapter.

Evolutionary computing is widely discussed and touted everywhere in contemporary science jargon, let alone the architectural sect hailing about the dawn of parametricism. Yet the actual software is still in its infancy. The presumption in them is that a person with a background in programming is going to be their sole user. However, there is one exemption to this: Galapagos. It is a plug-in, developed along Grasshopper®, a well known graphical algorithm editor within Rhinoceros 3D® — that implements two generic solvers: a genetic algorithm and a simulated annealing which is not discussed in this research as it is a probabilistic, not

---

075 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
076 Ibid.
evolutionary metaheuristic algorithm.\textsuperscript{077}

Since this software can be used without extensive knowledge of programming and it is well known by the architects, it suits the needs of this research, namely to promptly harness the potential of computation into architectural design process. In this manner, an amount of performance is lost for intuition.\textsuperscript{078} However, this is tolerated as it then serves the original goal set for the research.

Since Galapagos is still fairly new development, there exist very little formal research or precedent studies where it had been implemented. This is the reason why the following paragraphs refer heavily to the texts written by David Rutten, the developer of Galapagos.

\section*{Overview}

The modules described in research area in the thesis introduction basically translate into groups of Grasshopper\textsuperscript{\textregistered} components making up the actual Grasshopper\textsuperscript{\textregistered} definition. Figure 56 illustrates a recap of this. First, input variables can consist of any sort of a single number. Second, the syntactic model consists of components translating the numerical inputs into geometry (points, lines, polygons, surfaces and polysurfaces) in such manners as the prototypes described in the previous chapter. Third, a fitness function needs to be defined by the user as any other definition in Grasshopper\textsuperscript{\textregistered} using components to produce the fitness value. Fourth, there is a single Galapagos component that incorporates all the metaheuristic properties. It needs to be connected directly to the inputs and to the fitness value. Finally, when Galapagos is run, the solutions will show up as Grasshopper\textsuperscript{\textregistered} geometry in the viewport.

The above dissection also implicitly reveals the nature of the algorithmic construction. The Galapagos solver, that is the metaheuristic algorithm construction, is characterized as open, and as such it does not need to embed the knowledge of the geometry,

\textsuperscript{078} Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 1.
i.e. it remains unaware of the nature of the geometry described by
the geometry (or architecture) model. It suffices that the solver is
receiving the fitness values from the fitness function and its only
task is to interpret those and guide the search process according
to them. The fitness function is then assessing the geometry. The
algorithm that is constructing the geometry can be categorized
as specific, as it is following specific instructions of the kind of
geometry it is to construct. This kind of an algorithmic construction
is easier to create, as different functions are separated and they can
be thus easily inserted into different configurations.079 Figure 56
illustrates the anatomy of the overall model. For an architect or a
designer with little knowledge of computing, this is a cost efficient
way to harness computing into design.

In order to elucidate how the evolutionary algorithm
within Galapagos solver proceeds to traverse the landscape to find
the solution, the same example presented with the state action
graph, shape space and fitness landscape is continued. There are
basically seven phases in the process the solver goes through.

In the first phase the solver starts by populating the shape
space with a random population of individuals (see figure 57) who
form the initial generation. This means that all the input values of
the model are randomly set.080 The larger the population, the larger
are the chances of getting a good solution at random. Typically a
population of 50-100 individuals is large enough to generate good
solutions.081 Since the solver is based in evolutionary computing,
individuals could also be referred to as genomes — a specific value
consisting of genes. For example, the highlighted genome in figure
58 has a genome consisting of gene x (according to variable x) and
gene y (according to variable y). Had the model been more complex
, the genome would also had been more complex which is usually
the case.

Since all the genomes in the initial generation were
randomly chosen, this generation is referred here as initial (G0)
since it is generated only once (randomly) and it is merely setting

079 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY
WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 3.
080 Rutten, ‘References about Galapagos?’
081 Rutten, ‘2010-05-11 RUTTEN, David.’ 0.22.08
192
the scene for the actual evolutionary loop. Further generations ($G_n$) are distinct, since they subjected to more complex procedures that will be detailed in following paragraphs.

The second phase is the point when $G_0$ is seen to become $G_n$. It is merely to mark the beginning of general loop and to provide coherence to the phase structure.

---

Figure 56: A recap of the research areas which also denote the Grasshopper definitions structures. The progression in corresponding chapters is also indicated.

---

082 Rutten, ‘References about Galapagos?’
083 Rutten, ‘References about Galapagos?’
084 Ibid.
Figure 57. Random generation of individuals.

Figure 58. Worst genome are excluded.

---

085 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
086 Ibid.
194
In the third phase the genomes are evaluated according to the fitness function and a landscape can be extruded.\(^{087}\) This is also the most time consuming step in the loop.\(^{088}\) Since it can be assumed that the genomes locating on higher ground are closer to the fit solution than the ones on lower ground, the worst genomes can also be dismissed in this phase.\(^{089}\) Note that the landscape shape is based on the locations of genomes and everything else in the illustration is unknown. The complete landscape is drawn only because of its illustrative value.

In the fourth phase \(G_n\) populates \(G_{n+1}\). It is unlikely for genomes to be of high fitness as the exclusion of the low ranking genomes is not an accurate sieve. This is why the most fit genomes (figure 59) of the \(G_n\) are selected for breeding in order to form the first generation \(G_{n+1}\) that is not entirely random. When two random genomes are bred, their offspring in the next generation is located somewhere in the intermediate landscape. The new offspring in population of genomes is now located in the pristine landscape, thus uncovering more landscape, and has already clustered (figure 60) around the high ground, the fitness peaks.\(^{090}\) It is also possible for genomes to survive into new population without breeding.\(^{091}\)

This concludes the phases of the solver. Phases from two to four are repeated until a defined number of generations has passed, no progress has been made during a given number of generations or a certain fitness value is reached.\(^{092}\) More detailed description of processes, such as the breeding processes in the fourth phase, are described in the following titled somewhat according to the original source.\(^{093}\)

Despite the fact that the user can define properties of Galapagos it is acknowledged that some underlying structures remain out of reach for user manipulation. So, there exist some random behaviour in the structures of Galapagos.\(^{094}\)

\(^{087}\) Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
\(^{088}\) Rutten, ‘References about Galapagos?’
\(^{089}\) Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
\(^{090}\) Ibid.
\(^{091}\) Rutten, ‘References about Galapagos?’
\(^{092}\) Ibid.
\(^{093}\) Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
\(^{094}\) Rutten, ‘References about Galapagos?’
Figure 59: Best genomes are selected to breed.

Figure 60: First non-random generation.

---
095 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
096 Ibid.
196
Fitness Function

Fitness function is constructed by the user as any other definition in Grasshopper® to produce a single number, the fitness value. In general, no generation of geometry should happen in the fitness function, only the extraction of the fitness value.097

The manner of which the fitness value is calculated is crucial for the whole process of finding a solution. In most cases, a fitness value consists of several variables. It may be that if one variable improves, it will further affect the improvement of the others. However, it can also be that some variables will improve with the cost of others worsening. Yet variables can also have no effect at all to each other. As an example, a fitness value (f) could consist of the following variables:098

\[ f = a - b + c - d \]

The signs in front of each variable tells whether it is minimized (−) or maximized (+). Yet the simplicity tells nothing about the complexity of the variable relationships in the actual fitness function. Albeit the freedom to construct the fitness function in a desired way, there are some general rules about how it can be optimized. Three instances, the equally treated variables, linear functions and fitness pressure are described in the following.099

First, to continue the example, variable a consists of values ranging from 0 to 100 000 and variable b from values ranging from 0.0 to 1.0. This enormous difference in the domain ranges means that changes in the smaller one will have little or no effect on the overall fitness value, while even the slightest change in the larger one will drastically alter the fitness. To balance out the discrepancy between the equally treated variables, the fitness value should be calculated as:100

\[ f = 0.00001a - b + c - d \]

097 Davey, ‘Grasshopper Galapagos Tutorial.’
098 Rutten, ‘Define “Fitness”.’
099 Ibid.
100 Ibid.
By dragging the variable within the same range using a *fudge factor* (in this case 0.00001) enables reciprocity between variables.

Second, if the function responsible for a variable is linear, i.e. when the increase of input values results in a linearly increased variable value, the fitness function easily results in extreme unwanted outcomes. These extremes can be excluded by assigning *penalty clauses* that instantaneously affect negatively by subtracting or adding a significantly large value on the fitness value of a variable when a certain limit has been passed. Other option is to incrementally affect the variable meaning that the relationship between fitness and input values in a function producing the variable should be non-linear. Thus, when input values are small, fitness will increase significantly, and when values become large they do not affect the fitness as much. In this manner, the variable values remain within certain constraints and no extremes are reached. Continuing the example, say, c has to be of non-linear relation.\(^\text{101}\)

\[ f = 0.00001a - b + \sqrt{c - d} \]

Lastly, fitness pressure can be described using a different example. The force by which the solver is advancing in the fitness landscape is described as fitness pressure. This simply means that if the pressure is high, the solver traverses fast and if the pressure is low, the solver traverses slow. Yet, too high of a pressure, and the diversity of the gene-pool is reduced, since all genomes are mapped on a narrow path from the basin to the summit because of the determined solver that hastily interprets the terrain. Also, too low of a pressure, and the population is spread randomly and there is no clear path to the summit. Even worse, if the pressure is zero, the solver does not know in what direction to advance. The fitness pressure becomes palpable when comparing example results of two fitness functions that are written in different manner for the same problem.\(^\text{102}\)

Rutten describes an area divided into three parts (a, b and c). It does not matter whether the areas are geometrically equal rather their areal values should be equal. Yet, there are a several ways

\(^{101}\) Rutten, ‘Define “Fitness”.’

\(^{102}\) Rutten, ‘Fitness Pressure.’
of calculating the fitness value. For our purposes here, one is enough to elucidate the fitness pressure issue.\textsuperscript{103}

\[
f = |a - A| + |b - A| + |c - A|
\]

In the above function fitness is calculated by comparing the difference of an individual area to the ideal area (\(A\), which is calculated by dividing the original area by three). Say, that in a given situation the values for \(a = 2.0\), \(b = 2.0\) and \(c = 5.0\), thus \(A\) is 9.0. Then the fitness value is 18. Then the process goes on and in a later given situation the values for \(a = 2.5\), \(b = 1.5\) and \(c = 5.0\). The fitness value, since the process has advanced a given amount (\(a\) is now closer to 3 than before), should be better, that is, smaller, as the difference between areas should be close to zero. Yet, the fitness value, again, is 18. Even though the solver has advanced, the fitness has not. In this situation the fitness pressure is zero. To bypass this issue the function needs to be amplified to produce more drastic results when input values change. There are several ways, such as the penalty clauses described earlier, however a general rule would be to square the components. In this manner, the larger the deviation, the larger the decrease on variable fitness. Thus, the original function should be written as:\textsuperscript{104}

\[
f = |a - A|^2 + |b - A|^2 + |c - A|^2
\]

If the example situations are now repeated, fitness is first 114.0 and later 114.5. Thus the ability of the function to assess fitness has developed and there is now fitness pressure.\textsuperscript{105}

The above illustrates the general reason of why values should sometimes be amplified. However there is also the possibility of \textit{square rooting} a fitness variables. If one is interested in emphasizing the small change the square root of a variable should be taken to the fitness function. The figure 61 shows a graph which illustrates how small variable change can be emphasized using a square root.

\textsuperscript{103} Rutten, ‘Fitness Pressure.’
\textsuperscript{104} Ibid.
\textsuperscript{105} Ibid.
Figure 61: Large variable changes affect less on the fitness variable than small changes.

If one on the other hand needs to affect a general property and thus is interest in emphasising large deviations in fitness variables, one should square the fitness variables before inserting them into fitness function. This is because it makes the variable act so to speak in opposite manner as if it were applied a square root: Large deviation are emphasized and small detailed changes do not matter. Figure 62 below illustrates this relationship.

Figure 62: Small variable changes affect less on the fitness variable than large changes.

That concludes the construction of a fitness function, It should be noted though, that Galapagos also provides a fitness graph which can be used to navigate through different generations. In the graph the user can observe the process of the solver. Generations
\( G_n \) are plotted on the x-axis. The red line in the graph represents fitness of the whole population, orange is average fitness, or the standard deviation that contains most of the individuals, and yellow areas represent maximum and minimum fitness values.\(^{106}\)

**Selection Mechanism**

As stated earlier, there are countless different methods to construct metaheuristic algorithms. In terms of structures described as *selection algorithms*, Galapagos includes three basic ones. The mechanisms for parent selection are: *isotropic, exclusive* and *biased* selections.\(^{107}\)

Isotropic selection acts as there would be no prerequisite for selection, or that there is no algorithm set to do that. Thus, in isotropic selection every genome, found anywhere in the landscape, has the same chances to breed. This in turn hinder the population from reaching high ground quickly. It acts as a precaution against premature clusterization of genome around peaks, since there is not enough information whether the peaks are of local or global optimum.\(^{108}\)

The second mechanism is the exclusive selection that allows only the (user defined) top percentage of the population to breed. In this manner a genome in top percentage has better chances to breed when a certain percent is already excluded in the beginning.

In the last mechanism, biased selection, genomes’ chances to breed increase as their fitness increases. Thus, most of the genomes are able to breed, however the most fit ones are able to breed more often.\(^{109}\)

**Coupling Algorithm**

After the selection algorithm has selected genomes qualified for breeding, the coupling algorithm then selects specific genomes to

---

106 Rutten, ‘2010-05-11 RUTTEN, David.’ 0.26.30
107 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
108 Ibid.
109 Ibid.
be bred. Only one coupling algorithm is provided in Galapagos: the selection by genomic distance. In order to understand how this selection proceeds, the genome map needs to be explained. 110

A genome map represents all the genomes as dots in a given population on a grid. In Galapagos every genome has the same amount of genes. Since a genome can consist of a number (N) of genes, as mentioned earlier one axis represents single genes variables, the distance between two genomes can be an N-dimensional value, and thus it is impossible to illustrate an N-dimensional point cloud on a 2-dimensional grid. The axis in a genome map thus lose their importance as well. This is why distances between genomes in a genome map are poorly analogous, however the information the genome map successfully represents is the similarity of certain genomes. This is illustrated as the distance between given genomes. The ones close to each other consist of similar genes while the ones far apart have differing consistencies. 111

In general, there exists two thresholds in coupling that are of concern and they can be described using the genome map. Figure 63 illustrates these. First, if genomes in the immediate vicinity are bred, the resulting offspring are much like the parents and no novel genome is produced. In computing, this kind of in-breeding will result in decreased variability within a population, which in turn leads to decreased chances of finding solution basins. It also means that there is greater chance of getting stuck in a local optima. 112

![Figure 63](image.png)

Figure 63 113: The two thresholds of coupling.

110 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
111 Ibid.
112 Ibid.
113 Ibid.
Second, in order to avoid the loss of diversity, only genomes far away, thus of greatly differing gene consistency, could be selected for coupling. However, this out-breeding comes with equally hazardous drawbacks. If genomes that are bred come from two different groups rather than from one large population, chances are that their offspring will fall in between these two groups. If the two groups have been clustering around different peaks, the in-between landscape where the offspring is located, is most likely a valley and thus the offspring has a decreased fitness.\textsuperscript{115} Figure 64 illustrates such a scenario.

It suggests then that coupling should be done between genomes that are not too close but not too far away either. This is area can be defined as a domain that is represented as between -100\% \& and +100\%. Galapagos does not take into account the fitness of the genome falling within this area.\textsuperscript{116} As a general rule, 50–75\% is suggested.\textsuperscript{117}

In a more detailed view the genome map shows the parents in black and dead ends, or individuals who did not take part in breeding, in red.\textsuperscript{118}

\textsuperscript{114} Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
\textsuperscript{115} Ibid.
\textsuperscript{116} Ibid.
\textsuperscript{117} Rutten, ‘References about Galapagos?’
\textsuperscript{118} Rutten, ‘2010-05-11 RUTTEN, David.’ 0.27.11
Coalescence Algorithm

Galapagos includes two different mechanisms to assign values from the genome of parents to the genome of the offspring. These are crossover coalescence and blend coalescence. To describe them, an example of two genomes, each consisting of four genes, is used.

In crossover the offspring receives a randomly assigned set of genes from one parent and the rest from the other. In this manner, the values of individual genes are not altered. Figure 65.1 illustrates this.

In blend the offspring receives genes whose values are somewhat averages of the parents' gene values. Figure 65.2 illustrates this. This averaging can be altered using blending preferences to emphasize values inherited from the more fit parent. Figure 65.3 illustrates this.

Galapagos lists the genomes according to their fitness value within each given generation. The user is allowed to reinstate them if wanted. This is useful if intermediate solutions are of interest, or if the final output model is not pleasing.

Figure 65: 1. Crossover, 2. Blend and 3. Blend with emphasis.

119 Rutten, ‘References about Galapagos?’
120 Davey, ‘Grasshopper Galapagos Tutorial.’ 11.48
121 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
Mutation Factory

All of the mechanisms, that is the selection, coupling and coalescence, mentioned earlier are set to reduce the diversity of genomes in a population. The only mechanism that can increase it is mutation, namely point mutation in Galapagos. In order to understand mutation, a genome graph should first be explained.\textsuperscript{122} Figure 66 illustrates a one.

Again, a simplification has to be made in order to represent multi-dimensional points (genomes) in a two-dimensional graph. Each bar in the graph represents a single dimension. Multi-dimensional points are drawn as series of 2-dimensional lines connecting different values of a single dimension residing on vertical bars. In this manner, points with any number of dimensions, and even points with a different number of dimensions can be displayed in the same graph.\textsuperscript{123}

In figure 66 a genome consisting of 5 genes is illustrated using a genome graph. This genome is thus a point in the 5-dimensional space that delineates this particular species. Genome ($G_0$) is marked at 1/3. This means that the value is 1/3 from the minimum limit and the value 2/3 from the maximum limit. Since lines connecting different values are plotted as series of lines in the graph, it is easy to notice deviating lines, and thus the subspecies and lone individuals in a population. Thus, when mutation is introduced to a genome, there should be a clear deviation shown in the graph. In figure 66 a point mutation, a change in the value of a single gene, is shown.\textsuperscript{124}

\textsuperscript{122} Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
\textsuperscript{123} Ibid.
\textsuperscript{124} Ibid.
Maintain High Fitness

An optional phase regarded as the sixth phase in the loop can be used as a precautionary measure to make sure that fit individuals are never left out. By default 10% of the genomes from $G_n$ are allowed to displace individuals in $G_{n+1}$ if they are more fit. In general, the fit individuals (10%) of $G_n$ are compared to the most unfit genomes (10%) of $G_{n+1}$.

The reader now knows the basics to generate architecture — albeit only floor plans — using a metaheuristic algorithm. The process of evolutionary design generation, as all computational processes, is a highly systematic method of constructing design. As such it is similar to the design methodologies where multiple designs, scale models or drawings are produced and their good parts are combined in the second iteration of design. The clear difference is however the systematicity that is unintuitive and unable to interpret. Therefore the architectural models in computation are not malleable to designers quirks and reasoning.

---

125 Rutten, ‘Evolutionary Principles Applied to Problem Solving.’
126 Rutten, ‘References about Galapagos?’
The next chapter is an example of this kind of process and it is carried to demonstrate the significance of the methodology presented so far.
V. The experiment

“Architectural concepts are expressed as generative rules so that their evolution may be accelerated and tested. The rules are described in a genetic language which produces a code-script of instructions for form-generation.”

- John Frazer

The preceding chapters have all been acquiring knowledge for the construction and design of a model for floor plan generation. In this chapter one will be constructed and scrutinized. The model construction is not pursuing the goal of innovation in its field, but it is built to aggregate a variety of practical knowledge for the research.

In general, one has to avoid confusing the model for a finished work of architectural floor plan that would represent its most important aspects: human occupation and encounter in spatial systems. The model is constructed to resemble an architectural floor plan on a very general and abstract level. Instead of focusing on a design of a detailed self-explanatory floor plan, the model construction is more concerned about the design of relations that could possibly produce such an outcome. I.e. the modelling is concerned of the logic how parts — spatial elements, adjacencies and such — construct the whole — the floor plan. The underlying logic then is to construct a model that is able to produce feedback out of spatial relations during given iterations, and thus allow

001 Frazer and Johnston, An Evolutionary Architecture: Themes VII, 9.
002 Coates, Programming.architecture, 159.
morphogenesis to take place. In this manner, it is approaching a vernacular defined by its inherent algorithmic processes\textsuperscript{003} and distancing itself from the specificity of a certain building typology or aesthetic style.

The constructed model is based on the precedents and the syntactic methodology described in the third chapter. It thus tries to mediate between them, and incorporate their merits into its own constructs. It should also be clarified that the model elaborated here is placing itself in the early phase of design when one is seeking to generate novel approaches, or concepts, and not in the end phase when one is optimizing one specific design in detail.

In conclusion, as a generalized interpretation of architectural floor plan, the model construction is more easily attainable for a wide range of designers. It thus tries to form a general take on architectural computation and avoid being an open-ended research, a research with no predetermined criteria or limit for the design in question.

\textsuperscript{003} Coates, Programming.architecture, 160.
The prototypical building

“Perhaps the massive wave of the cultural studies people has passed somewhat, leaving a few survivors twitching in the sand to plod on in this very English pursuit of philosophy in the tradition of the logical positivists, the structuralists, and an interest in the natural language of architecture.”

- Paul Coates

This thesis research was partly conceived out of the criticism towards the inherent case study bias or the open-endedness infested in digital architectural design. Therefore, it is only reasonable that it itself does not resort to the same exit strategy. This is why a prototypical, generic building is used in the following demonstration. If a generic and fundamental architectural aggregate is studied, a continuum for developments is more likely to be established, as a generic and elementary example can be seminal to different architectural design contexts, not just one specific. Such context as housing design, as an architectural task of seeking spatial quality in relation to small amount of square meters, public buildings, rendering culture into sublime spatiality, not to mention villas, design for relentless individuality, could all equally benefit from an general construction of architecture. It would also be arrogant for a thesis research, which set out to ask how one can generate a floor plan using a computer, to proclaim that it was able to incorporate context specific design theory and implement that into the computational model, let alone

004 Coates, Programming.architecture, 160.
develop that into an innovative construct. The fact that a simple prototypical construct is needed, highlights the notion that a general understanding of computation is still lacking in the field of architecture.

A building can be defined as a manifestation of the social knowledge and unconscious organising principles behind the description of society. By commencing a study that is dealing with floor plan design in a syntactic, or computational, manner, it is also an attempt to limit the effect of cultural prejudice in design.

Despite the above reasoning for the elementary, the research experiment seeks to produce a model of which designs have the capability of being characterised as novel, meaning its results can surprise the designer. (See the advanced methods on page 143). That is, the experiment is seeking to generate emergent layouts that are coherent and syntactically correct (see page 107) yet resemble architectural floor plans.

The overall purpose of the experiment is to provide a contrasting outlook on the methodology of this research. This chapter could be characterized as a diary of trial and error in a learning process; an attempt to extract more knowledge and information that was not possible to derive from literature. With the help of a demonstration, the theory can be reflected in greater detail.

---

005 Hillier and Hanson, The Social Logic of Space, 184.
Experiment construction

“What classes do we want to recognize in conceptualizations of buildings and critical discourse about them? How should we characterize and label members of classes? What properties do we want design elements to inherit by virtue of class membership?”

- William J. Mitchell

In the following, the construction of a floor plan generating model and its operations are explained. The model is constructed using Rhinoceros and Grasshopper software. The utilized metaheuristic is an evolutionary algorithm found in the Galapagos solver, a component of Grasshopper. Elucidation of the generation process consists of three parts. First, the construction logic of each model part is explained i.e. the anatomy of the model is studied. This consists of the geometrical constructs and methods of the variable it produces to inform the fitness function. Second, the functionality of the solver and the evolutionary algorithm in relation to different settings is looked at. Also, the characteristic problems of such a computational framework are scrutinized. Finally, the generated results are analysed and future developments contemplated.

In order to communicate the general idea of how the operation proceeds, the overall development lineage of a given evolutionary development process is shown as illustrations.

from certain generations. **Figure 67** below illustrates the general process of elements locating themselves according to the design criterias implemented into the model. It shows how the overall shape of the floor plan aggregate starts to take shape and highlights the differences between the creation of computer generated geometry and human designed layouts.

As in previous illustrations the thick dotted line represents the program adjacencies. Thin rectangles represent individual spatial elements. The diagonal hatch is indicating of overlap and thus about the unfit part of the design. The thick boundary line is the sought envelope. Outside this envelope the cross marks are illustrating the deviation from the rectangular form. In the following chapter this process is explained in detail.
Figure 67: A sequence of best solutions of generations.
Model anatomy

“In such systems, ‘the designer’ was to be reduced (or elevated) to the level of system designer; and the ultimate ‘design’ was the emergent outcome of the complex interaction taking place under the software’s control on the aggregating system.”

- Paul Coates

The geometry that constitutes the model for floor plan generation could be defined with a myriad of different methods. Here, simplicity is sought in order to carry out a rudimentary test or demonstration of the general capabilities of a floor plan generating model. The model developed in this demonstration is constructed using rectangles that have adjacency constraints, and are packed within a unified envelope. Figure 68 illustrates this goal. In this manner, the syntactic method described in chapter three is followed, and minimal information is used to represent architectural floor plans. It should be noted right away, here, that many details, such as door or window placing, are intentionally left out to allow efficient simulation of the model. In general, the logic of the model construction is twofold.

First, the designer translates the architectural program brief into the model definition. That is, the amount of spatial elements and their corresponding areas are defined. The adjacencies, or the system of construction, of the program elements are interpreted from the brief as well. In the demonstration,
seven spatial elements — 180 sqm, 90 sqm, three 45 sqm and two 22.5 sqm — are used, and two hierarchies are formed.

Second, the designer starts the Galapagos solver, and the evolutionary algorithm starts to process the fitness values produced by the fitness function that is interpreting the geometry of the model.

In the solver, a decreasing fitness value is sought. That is, all the properties desired to emerge in the model are translated into computing as deviations from desired values. This means that the definitions are calculating the distance of a given adjacency or comparing it to zero. As an element is restricted to overlap another element, the length never diminishes to zero, so the shortest distance is the best one. Also, the difference between the desired area and the current area is calculated in a given iteration. Then, the boundary definition seeks to organize the rectangles into orthogonal relations by calculating the degree of angles in the boundary envelope and their deviations from 90 or 180 degrees. Also, the subtraction of total element areas from the area of the bounding minimum rectangular are closing in on zero. As such, the fitness function consists of a sum of variables that are responsible for element areas, element adjacencies, program adjacencies, angles of boundary polygon, and the difference between consumed total area by the elements in relation to the boundary area.

The designs presented in figure 67 represent only a fraction of the processed designs. As it was described in chapter
four, the evolutionary algorithm in Galapagos uses specific strategies to breed certain genes of solutions. The designs in figure 67 are offsprings of the designs in the previous generation that are results of such strategies. This explains why some designs depart greatly from the previous designs, as they were bred with different genomes. Therefore, the development lineage is not simply a gradual transformation.

More detailed description of the model anatomy is presented in the following as a kind of a diary of model construction. The idea is to show the methods that failed, and how the development process then continued. More detailed technical descriptions of the construction, that is the Grasshopper definitions, are included in the appendix starting on page 273.

**Element areas**

Regardless of the form a spatial element takes, it has to incorporate information of its desired size into the model. Thus, each element in the model can be given an areal size for the element to reach. This is done by subtracting the area of the element during a given generation from the original, desired value. Then, an absolute value of that subtraction is used, for the value cannot result in a negative value which would confuse the solver. The absolute value is then squared, since it is a sufficient and reliable method to penalize too large and small areas. The resulting variable is then delivered to the fitness function. If the area of the element is equal to the given favourable area, the subtraction results in zero. Otherwise a value derived from the subtraction will represent the distance from a fit. Shorter the distance is, the better the fitness.

In the process of constructing the overall model, it turned out that the element of area definition is by far the most difficult one. The fitness pressure it creates seems to easily override the goals of other definitions, while turning the area definition off does not seem to affect significantly to the appearance of desired areas. This results from at least two issues. First, by allowing the fitness variable to reach zero, one takes a risk of losing fitness pressure, as areas with correct square meter size will have zero as fitness variable, and thus are reluctant to take other shapes in following generations. The
increased fitness in other definitions does not result in progress, as they would most likely make the area fitness variable deviate from zero. Second, the elements, if they are allowed to have a strong area size definition, become emphasized as individuals. This means that as their individual size matches the size described in the brief quite accurately, the organizing method of all elements is confused. This is because the deviation in area size will manifest in the fitness variable too drastically, while loose packing of elements is not. Figure 69 below illustrates a coherent whole with elements of average desired areas, and an incoherent whole with more matching but stretched elements. In conclusion, the area definition is needed, however, its effects should be subjugated to the other definitions. The fitness variable corresponding the area is squared since area definition is not meant for detailing purposes but rather as an approximate guideline.

![Figure 69: Coherent and incoherent wholes using rectangles, however the similar phenomena could be illustrated using other primitives as well.](image)

**Element shape**

As stated earlier, the experiment strives for simplicity, thus only rectangular shapes for elements are produced in the experiment, as they require the least amount of vertices to generate habitable space. Also, they do not allow the formation of concave shapes which are demonstrated in figure 70. Thus, the variance of the element geometry can only take place as varying the ratios of the proportions of rectangles that are constructed using x and y input variables. These variables are also the variables, that along with the topological location of the rectangle, constitute the genome of the model, that is, they are the variables that the solver is directly addressing.
Figure 70: Convex and concave shapes

In the experiment the preferred ratio can be set as a fraction. Unlike with areas, no ratio is given the status of optimal, which was the case with the value that equalled the given area size, and subsequently the value submitted to the fitness function was zero, the best possible outcome. Rather, in the case of rectangular proportions, certain ratios falling between certain domains are sought, and the ones not within this range are penalized. The implementation of this kind of a penalty system, namely penalty clauses, makes the situation more complex. There are two ways to approach the situation at hand. First, the solutions failing to meet the domain are punished as outright unacceptable, and thus given a large fitness value that is larger in multiple scales. In this manner, unwanted ratios will not appear at all, since the penalty is severe. The other more agile approach is to penalize the values failing to meet the wanted value non-linearly, that is, values close to the wanted value are penalized but not as severely as the ones further away from the desired value. However, as the simulation is done within one coordinate system, and since the proportions of the elements are constructed using x and y values, there can be two identical elements that are equally fit and have same proportions however their x and y values are flipped. Rectangles b and d in figure 71 below illustrates this. The problem is not that there can be multiple fit solutions, but rather, that the in between shapes need to coherently guide the solver towards the fit solution. Since they

008 Hillier and Hanson, The Social Logic of Space, 182.
are polarized, the situation is troublesome. Also, the midpoint of the domain equals a square, which is most likely an unfit outcome, unless the equal ratio of the edges is the desired one. Therefore, a definition that coherently guides the solver away from solutions resembling a, towards b, accepts c but does not merit it significance, towards d and again away from e. During this it should sustain a certain fitness pressure to do this efficiently.

Figure 71: Same elements with different values.

An easy solution for the above mentioned problem is to use the first method described, and allow the elements to adjust mostly according to other criteria in the overall model. The only needed criteria is the penalty clause for overtly unacceptable shapes. However, this does not solve the general issue of ratios, so a more advanced method needs to be used.

The method for solving the issue is to define the given ratio and a value where the indicator and the denominator are swapped i.e., a rectangle always has two ratios: x/y and y/x. Then the fitness variable produced by the ratio function is always the smaller one out of these two values. The value is thus calculated by subtracting the ratio of a given solution from the above mentioned ratios, and the smaller subtraction is then chosen. In this manner, square and stretched shapes are avoided. However, if squares are accepted, then the values between the two ratios constitute zero. Additional switch for making change more disruptive is to implement a 90 degree rotation ability. In this manner, the midland of the square form might be avoided, however the rotation can only be evoked by a mutation since the switch has
two states: turned on or off. That is, the whole evolution of the form might have developed with a genome where the switch is off meaning, that the on option is completely missing.

Like the area definition, the ratio definition is likely to make the overall model stagnated as it seeks to create fit individuals with the cost of a fit overall layout. It turned out that the scaling of its effect on the fitness function to a bare minimum allowed the model to develop more coherently.

**Element adjacencies**

The element adjacencies define which elements need to be close to each other. There are several ways to construct the adjacency definition, and the most general ones contemplated during the research are described in the following. However, a definition that would dictate that certain rooms should be as far apart as possible is not implemented into the model. It is argued that such a definition is irrelevant for the model functionality, since from the actual inhabitant’s perspective, two adjacent spaces can be perceived as distant only because there is no direct connection between them. 009

The simplest way to control the adjacency is to calculate the distances between center points of given elements. Thus, the distance is sought to be minimized between those elements that are desired to be next to each other. However, if the minimized distance is translated into the fitness variable as such, it will never reach zero since elements are not allowed to overlap. Therefore, there is constant fitness pressure in the model.

Another method considered was to create a line from the element center point to the nearest point on the outline of an adjacent element. The same line would be constructed for the adjacent element also. These lines would have been connected from the outline points. Then the only the distance left between the adjacent elements would have been calculated. **Figure 72** illustrates this adjacency connection. Since the connecting line would reach zero only when elements are in orthogonal relation, this would then encourage the whole model to be orthogonal as

---

009 Koenig and Schneider, ‘Hierarchical Structuring of Layout Problems in an Interactive Evolutionary Layout System.’ 135.
well. However, similarly as with the area and ratio definitions, by allowing a variable to reach zero has detrimental effects on the model agility. This kind of an adjacency connection was therefore dismissed.

A more elaborate way to control adjacency is to use two centrally located points for an element. If an element is a square, it will have a single center point as in previous case, however if it is another type of a rectangle, it will have two points along the central axis which are formed by the midpoints of its shorter edges. This axis is further divided into four parts, so the two points used for adjacency definition are at the second and third division points. Figure 73 illustrates this. When two points are used, elements align more easily in an orthogonal manner, as adjacencies between elements are calculated using the closer point out of the two, and the elements are allowed to stretch, so other elements can join in on multiple sides. However, this makes the model more complex and results in greater variability and a more rough fitness landscape. There is also a great likelihood that two center points of an element are equally close to an adjacent reference point and a double distance will be calculated, which although can be easily removed.
The simplest version, where adjacency is defined between center points, was chosen for the model demonstration. It proved to be the most efficient definition, as it was light and provided more fitness pressure compared to the other methods. The adjacency definition, however, can be twofold. On one level, the adjacency of all elements in the model is calculated to allow them to pack densely. On another level, the adjacency according to the architectural program is also calculated. Both of the fitness variables from adjacencies are squared, since it suffices that elements end up touching each other, and no straight lines connecting them are needed. This means that only the excessively long distanced needed to be penalised. The program adjacency definition was accompanied by a planarity check, that is, if the connections of the program intersected each other, a penalty clause was afflicted on the fitness variable. However in the final model construction the adjacency of all elements was dismissed as it is unnecessary and obfuscating function in the end. All elements do not need to be packed, that is a stretch rectangular envelope is as fit as a rectangle.

**Overlap of elements**

Unlike with previous definitions, the issue of overlap does not necessarily manifest itself in a model. That is, it is an anomaly of some but not all model constructs, however it is a common
one, and as such it is present in the model construct used in the demonstration. The overlap conditions found during the model construction are discussed in the following.

Models where elements are completely free to move and their relations are defined solely by the adjacency definition can have overlaps of many kind. This kind of a model construct is also prone to produce solutions which are not planar, and thus incompatible with the design criteria. Elements can overlap elements that they were supposed to be adjacent to, they can overlap other elements that they were not supposed to be adjacent to, and multiple elements can overlap each other. Concerning the overlap between elements, there is essentially three different conditions in which the elements can produce an overlap in a model where they are free to move unconstrained.

Figure 74 illustrates these three conditions.

First one is that they do not touch at all. There is a chance that an element is inside another element and the solver cannot distinguish between nested and loose elements. Therefore, a definition checking if an element is completely within another element is needed.

Second, the rectangles can intersect and thus create overlap. At first it might seem that it is comparable to the earlier situation, that the distance measuring the overlap should equally affect the fitness value negatively. However, since we are dealing with floor plans, this kind of a situation makes no more sense than the first situation of nesting. The adjacency definitions are already pulling rectangles to each other, thus loose elements do not need to be penalised as drastically.

Third, the rectangles touch but do not intersect each other. This is essentially the best condition. Although, it should be noted that it is not because of the short distance but because the borders cannot be further apart. That is, they are not outside nor inside the neighboring rectangle. Although, if the first and second situations were penalized as drastically, the solver would become confused as only possible good fitness value would be produced by the third situation. Chances of arriving in this situation are small, and once even a single contact is reached, the solver would not leave that state and would most likely get stuck.
Figure 74: The three stages related to overlap.

An effective way to encourage a non-overlapping situation is to calculate the overlapping area and affect the fitness value with that. In this manner, the penalty is more definite and pressurized in comparison to a penalty clause affected if any overlap occurs. However, unlike a penalty clause, the overlap definition will result in decreasing element area sizes and, thus in failure of the area definition to live up to its purpose. This is because smaller areas result in smaller overlaps. A certain definition calculating the total area of elements and subtracting that from boundary is needed to balance the situation. This will be described later in detail, however it suffices to note now, that this definition counterbalances the anomaly of decreasing elements and allows the overlap to produce fitness variables that guide the search.

The above mentioned structural problems in models where elements can move freely can be bypassed by constructing the model in a different manner. These kind of constructs would be definite future developments. Instead of freely moving individual elements as in the demonstration model used here, elements would be constructed into a hierarchy of parents and childs. A child is an element constructed from the outline of another elements, namely the parent’s outline. In this manner, the adjacency is constructed into the model and there is no concern for overlap between a parent element and a child element, as the direction of the construction of the child element is always away from the center point of the parent element, a child is always an offset of a parent. Thus, the only possible overlap that can take place is when other elements get in between the parent and a child. This kind of a construct can create further hierarchy if adjacencies for a child elements are
defined through the previous method of centerpoint or the distance of the closest outline point. These cases act as earlier stated, thus there can be elements in between and overlap can manifest as before. However, if the other connection is also constructed from points on corresponding elements, there can be no overlap between these elements, only overlap of unconcerned elements. Figure 75 illustrates this.

Finally, the most rigid model where no overlap can take place is the one where all elements are constrained in a grid. The elements themselves do not traverse the space as in previous cases, but rather their information or program, including the area and ratio, does move between grid squares. Once a prominent cell is found, the information inhabiting it will start to swell the cell according to its information. If the situation changes and another cell might allow better opportunities for development, the information might traverse there. The problem with this approach is however, that once some cell swell large enough, other cell adjacent to them become too stretched, and thus incompatible for architectural purposes. Such a model would result in designs where empty cells would appear. Thus, a subdivision definition or a modular grid superseding the element accuracy should be implemented into this kind of a model construction. The subdivision construction would

Figure 75: Model where the possibility of an adjacent element overlap is removed. There is still chance of overlap between elements with secondary adjacency definition.
however result in a significantly more complex model. Therefore it was dismissed, as the demonstration in the study was sought to be introductory and preliminary. Figure 75 illustrates the approach of the above mentioned model. It has resemblance to the approach developed by Janssen.

Figure 75

![Figure 75: Illustration of the approach of the model developed by Janssen.]

Figure 76: Grid model with empty spaces.

Again, the most simple definition was used to allow fast and agile model development. Since the model already incorporated freely moving elements, the decision to use the first mentioned definition was because the parent child relation in the second model did not coherently remove the issue of overlap.

So, the overlap definition allows detailed information of the overlap produced as it calculates the area. The square root of the overlap area represents the fitness variable, since it is of interest to reward even the slightest increase in fitness. Also, a secondary clause was implemented which calculated the distance of the centerpoints of overlapping elements. This was done as a precautionary measure to avoid too aggressive overlaps. If the distance became too short, a penalty clause was afflicted to the fitness variable.

Bounding perimeter

The bounding perimeter refers to the unified boundary defined by the outer elements in the model. A definition is needed in order for it to appear in the models where elements can move freely, and where elements are constructed using parent and child constructs. An example of such a definition for the bounding perimeter is presented in the following. A bounding perimeter is shown in the figure 76.
Purpose of the boundary in these models is to pack elements and to smoothen the outline into as continuous envelope as possible in order to be reminiscent of an actual building floor plan. Without the bounding perimeter definition, elements would pack only according to the adjacencies, and the envelope would be left undefined and discontinuous.

A simple definition for the bounding perimeter is to create a two dimensional bounding rectangle defined by the elements. Then the method is to calculate the combined desired area of all elements, and to compare it to the combined area of the given solution. The squared absolute value of the subtraction is delivered to the fitness function, as a fitness variable describing the fitness of the boundary perimeter should emphasise the overall correctness of the area size. The danger with this approach is that if only the area is calculated, there is an increased risk for situations where elements are not allowed to escape certain areas since their transition would result in larger bounding area, even though the gradual development from the new location would eventually surpass the old fitness. Figure 77 illustrates this situation. Although this is a common problem in all boundary definitions, it is why the boundary perimeter should not be given too strong penalizing capabilities in the first place. Smaller elements become favored, despite their area definitions, as they result in a smaller bounding perimeter and they cause less overlap between each other. Therefore, the sum of all element areas should be equal to the area of the bounding perimeter.010

In this manner, the definition working for a continuous facade is not hindering the development of the interior structure. The perimeter definition should be thus viewed merely as a secondary definition, much like the area definition of individual elements, a detailing definition to support the adjacency definition.

However, as the purpose of the bounding perimeter is to smoothen out the envelope, it does not have to interfere with the model through total area calculations. Rather, it should mostly affect the connecting of the corners of adjacent elements, that are on the outer perimeter of the aggregate, to the overall boundary. Therefore,

010 Koenig and Schneider, ‘Hierarchical Structuring of Layout Problems in an Interactive Evolutionary Layout System.’ 134.
the corners of the boundary polygon are calculated. Corners which are closing in on 0, 90, 180 or 360 are considered fit. In this manner, the overall boundary of elements will take a more rectangular shape. This is a more elaborate method of creating a unified boundary condition for the model that also allows the model to develop more freely.

Figure 77: The boundary perimeters angles and subtraction areas.

However, a certain overall area calculating definition was used in the model demonstration. Subtracting the total area consumed by the elements from the area of the bounding perimeter box allows more dense packing to occur. Reason for this is that as the individual elements enlarge they consume more from the bounding box. This counterbalances the diminishing size of the elements resulting from dense packing. Since the subtraction is closing on zero as the total element area is becoming as large as the bounding box, the orthogonality trait of an architectural floor plan becomes more visible.

The square root of both the angle and area fitness variables were delivered to the fitness function, since the boundary definition is concerned with the detailing of the aggregate, namely the orthogonal shape.

This concludes the model anatomy. Now that the model parts are explained, it is logical to define the solver settings suitable for such a model.
Solver settings

“Evolutionary algorithms suffer from the same problem as all such optimisation procedures, that of having to define a fitness function – the evaluation test that determines who survives each generation.”

- Paul Coates

The information about the general logic of how the solver settings are operated was detailed earlier in chapter four on page 201. The specific settings that were used in the demonstration model are presented here.

Fitness function

The fitness function in this model consists of variable values that are added together and this addition then sought to be minimized. The variable values are derived from the above described definition accordingly and the fitness function for the demonstration model is:

\[
750 \sum a + 250 \sum \sqrt{b} + \sum c^2 + 50 \sum d^2 + 10000 \sum e \\
+ 100 \sum \sqrt{f} + \sum \sqrt{g} + 3000 \sum \sqrt{h}
\]

where fitness variable a is the result of a semi-penalty clause for
nesting, **b** the variable for element overlap, **c** adjacency of all elements (optional), **d** adjacency of program elements, **e** penalty clause for program planarity, **f** the angles of boundary polygons, **g** the subtraction of total element area from the bounding box and **h** the variable for individual element area. Each sum of fitness variables is also multiplied with a factor in order to have a meaningful effect on the model development. This is a vulnerable spot in the design process, as each factor is defined by empirical study of what kind of results different factors provide. This is essentially a process of trial and error. Basically, emphasis is a multiplier that increases the fitness of a fitness variable into slightly higher order in order to emphasise the particular property the fitness variable is responsible for. If a model starts to gain complexity and there are multiple inputs, emphasising multiplications can be used to further scale variables within smaller range. In this manner, it is repairing or tweaking the model, that all properties, rather than fitness variables, are in balance and thus more coherent solutions are produced. However, the extensive use of emphasising multiplications is a symptom of a problematic model construct. Usually, the excessive tweaking of emphasising multiplication variables is a process of trial and error, and is ultimately induced by discrepancy of fitness variable scales. That is, some variables are close to zero, and even allowed to reach it, while some are not.

Galapagos provides a small set of settings for the solver that the user can alter. Some are programmed into the solver as default settings and they cannot be altered. In this sense, Galapagos is resembling a computer aided design tool rather than a computing method. The abilities of settings were described earlier in the previous chapter. The following is contemplation on how different settings affected the floor plan generation.

**Population**

By default the population is set to 50 individuals per generation, and the first initial random generation is multiplied by two in order to encompass a variety of solutions. The amount is somewhat decent at first, however when the model gets more complex, a larger population is needed. This need became evident when
fitness variables for nesting, element overlap and adjacency of all elements in fitness function were accompanied by program adjacency and planarity variables. Increasing the population size to 100 slows the evolution process, but more coherent designs start to emerge. Also, a multiplier for the initial generation was increased from 2 to 4–6. Now, the initial random generation has better chances of being populated by more fit individuals, and the starting individuals for evolution are better. Thus, the evolutionary lineage of solutions is more clear and coherent. There is also no such a large risk of arriving in specific solutions dominating the progress, as there is with a smaller population. With the above described population, there exists at least two rivalling populations of design for quite a long time.

The setting for how many stagnant populations should there be before the solver stops, was not greatly experimented with. However, the default 50 generations was increased to 100 generations. This was done merely as a failsafe, so that there would not be any disrupted lineages while allowing the solver to stop when there was clearly no change.

**Breeding**

The domain of breeding method is between -100% and 100%, where the -100% represents total zoophily and 100% total inbreeding. There is also a setting for the amount of solutions allowed to survive from previous generation to the next. I.e. the next generation is not formed only by the offspring but some parents become part of it as well. The breeding methods were described in detail in chapter four. By default, the solver is set to 75% meaning it is 75% biased for inbreeding. Also by default, 5% of solutions from an earlier generation are allowed to survive into the new.

By allowing fit parents to be included in the subsequent generation, good model geometries will not be lost, as they can only be replaced by better ones. In this manner, there are also no abrupt changes in the population. In contrast, if no parents were allowed in the subsequent generation, there would be no guarantee that the fitness is preserved as the children might be located in the valley in the first place. Despite this fact, generations that do
not include parents, do not get stuck as easily in local optima and are able to traverse the landscape more freely. For example, a common local optima problem manifests, when a single element stuck on the outer area of the element aggregate cannot escape its location as it is adjacent to a large element. **Figure 78** illustrates this situation. Small incremental changes are helpless in getting the element to the other side, as the fitness valley in between is flat and thus provides no information of where to go. More complex mutation properties or abilities to construct larger scale deviations in the topology definitions would be needed. Currently, the only viable way to deal with situations like this is by diminishing the probability of it appearing in the first place. This can be somewhat done by increasing the population size and emphasising the program adjacency fitness variables in the fitness function. In this manner, long distances in the genome become a minority and there is greater chance for them to emerge. However, it should be noted that this is a great flaw in the model.

![Figure 78: Element stuck on the other side of a big element while there is an obvious better location for it if the whole model is concerned.](image)

All in all, the default settings for breeding provide for a good development lineage to emerge. However, a slight decrease of the inbreeding factor helps to allow more variability in the design populations. I.e there are multiple parallel lineages of ways in which elements accumulate. This also helps to avoid the above mentioned local optima trap. However, the development lineage of

---

012 Koenig and Schneider, ‘Hierarchical Structuring of Layout Problems in an Interactive Evolutionary Layout System.’ 133.
the overall model is slower and somewhat obfuscated, as also more unfit solutions are greater in numbers throughout the development lineage. This is because offspring of the genomes that were far away in the genome map can be in the fitness valley.

However, to speed up a development lineage while maintaining the existence of varying solutions, a higher rate of surviving solutions can be set. In this manner, there is a larger amount of more fit solutions and a chance that novel offspring might emerge. Compared to the earlier mentioned strategy of allowing a smaller set of parents to survive and narrow down the breeding perimeter, this strategy remains too undetermined and consumes excessive amounts of time without producing enough fit solutions.
Technical demonstration conclusions

“Seeing a model struggle to resolve a situation allows the designer to identify with the algorithm, and this emphatic coupling provides the basis for a dialogue between model and designer, and designers and stakeholders. A narrative is constructed through visible behaviours leading to a quasi-conversation between the designer and the computational model.”

- Christian Derix & Åsmund Izaki013

The demonstrated floor plan generation is intentionally simplistic, and there is much to be developed if it would be applied to the processes of design. The most significant properties to develop and the conclusions of the whole process in general are discussed in the following.

Setup interface

The process of running the Galapagos framework is a tedious task if one has to create the whole model from scratch. However, if readymade components can be used or better yet, if a template definition would be used with the right scaling of fitness variables set up, the process would be near practical. The above is also said with the expectation that certain definitions in the model would be enhanced.

013 Derix and Izaki, ‘Spatial Computing for the New Organic.’, 44.
At the moment the model developed within this framework remains quite unreliable. It is interesting to notice how different developmental lineages can differ so greatly in their speed and ability to provide quality solutions. The initial random generation of design might, by pure luck, devise a fit solution and the development lineage will have a kick start, while in most cases there is a long path before decent solutions start to emerge. It would be efficient to have a fitness check for the first generations to see if the whole process is worth continuing.

The construction of a model translates into the construction of orders of scales and hierarchies. At one level, one has to have general definitions that take care of the overall construction. Example of this could be the adjacency definitions packing all elements together. It is not, however, concerned of packing the elements in too great of a detail. Such a definition is then high in the hierarchy as it dominates the overall development of the model. Then, one also needs definitions that do not affect on such a general level but have an effect on details. To counterbalance the adjacency function for example, the boundary definition is needed to finetune the angles of the boundary. Large angles result in bad fitness, but their deviations do not cause as large increase or decrease in fitness as deviations in small angles. As such, it is subject to the general definitions as adjacency. Finding the right kind of definition relations and defining their status in the hierarchy allows the model to be developed in an unobfuscated way.

Another unreliability of the model is that a large part of its construction consists of finding the right scaling factors for the variables in the fitness function in order for the model to run coherently. As described earlier, this is a task of trial and error. A more dynamic interface is of great need for speeding up the process of finding out the right balance of fitness variables to achieve a working model.

This ambiguity in setting up the scale factors also begs the question of whether a metaheuristic framework is an efficient implementation in such a model. As in Arvin and Houses’ model, the method of solving the generation was by implementing forces and physical properties. In such a case, the equilibrium of the model will emerge by itself and simultaneously the solution as well.
As the model where the elements are free to move and the only constraint is the overlap definition then it hints of a fundamental structural problem in the model. I.e. the grammar in the model is not well defined. Instead of producing solutions that are inaccurate in relation to the design brief, the model produces solutions that are erroneous in relation to the brief. Despite fact that the error is great in the first generations, it remains present in latter generations as well. The model should not have to develope by getting rid of errors rather it should developed by enhancing error free solutions. A more rigid system where overlap is restricted by the means of the model construction, not repairing overlap definition, would accompany a metaheuristic framework better.

Another way to increase the model efficiency is to narrow down the domain of input variables. This is an effective way to increase the model development, however, one should be careful not to narrow the domain down too much, and in the process of doing that eliminate the possibility of some fit solutions to appear. Yet, it is safe to narrow down the domain of input variables that define the topological location of elements, as the rough area needed for their traverse can be estimated from the total program area. Equal amounts of program total areas should not be used since there needs to be some excess space for the element aggregate to move, so that some elements are not pushed against the sides of the design space. Also, narrowing down the domain of vectors defining each element according to the corresponding element area size is effective. Again, one should be careful not to narrow it too much, as this will affect the elasticity of individual elements to fill unused space in the aggregate.

In the demonstration model all the elements were perceived to represent programmed spaces. However, they could represent merely areal boundary, that is, empty space. As the model was considered as a preliminary model development the following definitions were left out. Other such element representations include the light shafts and corridors.

In the current model there is no definition guiding the elements to locate themselves close to a source of light. There is then a chance that an element is surrounded by other elements from all sides. Right now the only possibility to receive light is
to locate on the aggregate boundary. Therefore, there should be a category of elements that are considered as light shafts instead of programmed space. These elements would be defined to locate themselves adjacent to programmed elements that can not reach the aggregate boundary. In this manner, the aggregate would resemble a floor plan to a greater respect. A light shaft however, can also be on the aggregate boundary. If located there it would of course not be a shaft, but rather it would be a concave shape in the envelope. 

**Figure 79** below exemplifies the usage of “a light shaft”.

![Figure 79](image)

*Figure 79*: The light shaft can be an actual shaft or courtyard. It can also be used to create envelope subtraction.

The corridor would also be implemented as an element and it would be a part of the adjacency definition of all elements. The corridor element would, however, be defined using only a minimum width, and it would have multiple center points. This would be similar to the alternative element adjacency definition explained earlier in model construction, in which the distances were calculated using multiple points. As such, the corridor would be allowed to stretch indefinitely to reach all elements. However at the moment, it is hard to say how the definition of corridor turns should be approached. In a small model such as the one on this research turns in the corridor would seem unnecessary since a straight corridor would be short in relation to other elements anyway. In a larger model the turning definition should be of concern. **Figure 80** illustrates the corridor element in relation to other elements in a model.

---

During the design of the demonstration model, three different alternative methods to construct a model came across. These would have however required a complete reset of the model design, so developments were not taken upon. This is why they are described here as paths for future developments.

First, the development path that resembles the demonstration model the most, would incorporate a level of element adjacency that would be one scale more detailed, into the structure of the model. In such a model an element, as it is defined in this research, would consist of smaller constructs. Therefore, the status of an element would be transferred into a spatial component and a collection of them would represent a programmed space, a light shaft or a corridor. The adjacency would also be calculated for an aggregate of such collection. Figure 81 illustrates such a difference in model construction.

In such a model the spaces could take other shapes, such as the L-shape, rather than just rectangles. There would be, however, a need to define the boundary condition for individual elements as well, and thus the model would be more complex.

Second, a fundamental problem with the models where elements are free to move, besides overlap, is that rectangular elements are not capable of swiftly passing a corner of an adjacent element. They tend to get stuck on one side as the adjacency line would become longer. Such a problem would not manifest, if the elements would be represented as circles. Figure 82 below illustrates such a case.
Figure 81: Change of element accuracy and its development of packing.

Figure 82: Circles can glide more easily to better locations. As there would be no corner to temporarily stretch the adjacency distance.

Therefore, a model where adjacencies would be calculated in one phase using circular elements and only later the actual shapes, more or less rectangular shapes, would be introduced into the model. This, however, would require a more elaborate framework than Galapagos. Although, a model which structure would be twofold could produce interesting results. It is described in the following.

The element adjacencies and preliminary element areas would be calculated using circles. In this manner, they would find better locations more easily. At the same time, at another level there would be constructs that would generate the final geometrical outcome. To form a dual graph, a voronoi diagram would be constructed using the circles. The voronoi would represent the actual walls of the floor plan. In order for the voronoi to represent a
more traditional floor plan, its cell angles, much like the boundary
definition in the demonstration model, would be calculated as a
fitness variable. These angles too would then be qualified as how
close they are to 0, 90, 180 or 360 degrees. Also, the cell areas would
be used to achieve matching relative areal sizes. Figure 83 illustrates
the pseudocode of such a model.

Figure 83: Double layered (dual graph) model construction taking advantage
of circle packing and voronoi diagram.
Third, a predefined grid where cells would swell to create a floor plan seems like a strong method. This was essentially the one developed by Janssen, illustrated in figure 38 on page 151 and further contemplated during the model construction (see figure 73 on page 231). This model construct would however need a subdivision definitions, such as the ones described earlier in the model construction.
Demonstration reflections

“Complex systems are full of interdependencies — hard to detect — and nonlinear responses. ... In such environment, simple causal associations are misplaced; it is hard to see how things work by looking at single parts.”

- Nassim Nicholas Taleb\textsuperscript{015}

The problems that arise from working with an algorithmic solver reveal its true nature. As a technocratic product, it translates geometrical elements through fitness variables into values that its fitness function ranks, and finally makes decisions on good designs based on one overall value, the fitness value, that is composed of the fitness variables. Yet, the computer slips from the responsibility of the design initiation as it requires the human designer to first subjectively decide for it how these variables are produced, i.e. what kind of functions are responsible for the values and how are they comparable and if they should be in the first place.

Yet, the most significant problem is how one can define the geometrical properties that qualify as good and those that qualify as bad? That is, given the range of fitness from 0 to 10, what kind of spatial form is best described as 2.5 and what 9.8? What is the geometrical property of the solution that is rewarded 0.0 instead of 0.1.? The formation of the hierarchy reveals the true designer who has been significantly involved in the

\textsuperscript{015} Taleb, \textit{Antifragile: How to Live in a World We Don’t Understand}, 7.
process taking place within the machine: A human being.

The fitness values’ slow increase or decrease is problematic. The development or evolution of the model takes place incrementally and therefore there is great risk that a solution will be isolated, as slightly departing solutions have decreased fitness by default. If differing genomes would be allowed to survive longer in parallel with the main development, the solver could be quicker in developing new solutions. This is basically what the human designer in the traditional sense does. A human designer does not gradually change the layout of a floor plan incrementally by slowly moving an element towards the other side of the plan, while during each move drawing the entire plan. Instead, the room will be moved with determination and the old plan will adjust to it accordingly in a one sketch, that is, in one generation.

Even though this is partially a flaw inherent to the simplicity of Galapagos, it is also ingrained in the design approach of an evolutionary algorithm. It might very well be that this problem first of all reveals another kind of subjectivity in the formation of algorithms described as definitive, but also the reason why the field of metaheuristic algorithms has such a varied scope.

The practical use of a given metaheuristic requires time and multiple iterations of trial and error. It needs tuning up and tweaking in order for the model to exhibit agility when it is traversing the fitness landscape. A given problem with a given solver requires given settings. Although, through logical and formal construction of a model, setting up the domain of input values and its fitness function, this phase of tweaking can, and should, be minimized.

Some methods of construction and weighing will get stuck, as penalty clauses make the model cautious since only a small set of solutions are maintaining the fitness rather than increasing it, while the rest simply destroys it. This is why nonlinear penalties allow for a more flexible and resilient model which then can move more agile. Also, one method to improve the model would be to allow phases when certain constraints loose their effect and the model is allowed to relax within certain limits. This would allow the elements to find more suitable locations. However, this kind of phasing is impossible with Galapagos.

In a way, the above could also be interpreted as a need
for a different kind of metaheuristic to guide the search process. Simulated annealing, for example, is characterised as one that first searches with a general scope and only later focuses on a small sample. It also allows fitness to decrease occasionally, which allows it to escape local optima.⁰¹⁶

What is also greatly necessary, is the ability of the user to tweak the mutation methods. This is currently unavailable in Galapagos. In relation to floor plan generation or the problems concerning packing of elements, there would be a need to specify mutations that would allow elements to escape in certain situations from their location without decreasing the fitness significantly. Although as always, this is not clearly a problem of mutation as the construction of the geometry can help to solve the issue. Such example is implementing the ability to allow element program to change elements, thus the escape is then made possible for the program.

**Novelty**

Another significant issue arises when the model is constrained in a manner that it is quick to produce somewhat expected solutions. For example the model developed as a demonstration in this research could be criticized for being such. However if one imagines a more well organized computational structure and a more complex program brief the resulting solutions of the demonstration would not be as expected as they were anymore.

A great part of the reason why computational models are constructed is that they could in theory produce novel unforeseeable solutions to a problem, as they are deriving solutions bottom-up from the parameters of individual elements opposite to the logic of human designers.⁰¹⁷ Yet, if the model is too drastically constrained could it then be unable to produce emergent and novel solutions? Isn’t it merely producing average solutions while the heavy constraints eliminate the outliers from the genepool?

⁰¹⁶ Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 10.
In many cases, one is confronted by the need to create a module construct, whose purpose is to act as a precautionary method for eliminating solutions which are visibly unfit for the human designer. However, without the elimination definition these unfit solutions would be qualified by the algorithm construct as fit or at least fit enough to survive in the lineage of solution. What guarantee does one have that these precautionary methods do not eliminate the possibility of a novel solution to appear if the whole definition of emergence is that the order of the whole is not visible in its parts?

Suddenly one is confronted with large and significant questions of unforeseeability and prediction. In architecture what is considered novel is usually considered also as, like in many other disciplines concerned with complexity, an unforeseeable event. These events can be positive or negative surprises. Yet, they both are clear entities, i.e. they are not compilations of noise or some random aggregates as they represent an emergent order that has an effect on their environment. The systems that are responsible for their creation only rarely succeed in it. How is it then that when in

---

018 Ibid, 280.
architecture novelty is desired — unlike in other fields the whole pursuit is its elimination — can we then elaborate this ability to create them? Yet, as no one can predict future, no one can enhance the system producing it. We can only create our environment so that it is able to perceive these unforeseeable entities. However, when concerned with design or scientific analysis, one is able to define what something is not instead of what something is. This point is further contemplated in the following chapter of the overall contemplation. If we are then able to say at least which solutions are not bad, we could extract some coherence that starts to resemble the grammatical properties described throughout the study. Thus, it would seem that through a grammar all solutions are not unfit and out of those solutions novel solutions can rise, since we are not excluding the design for fit solutions only. In this manner solutions are diluted with noise, however they are not overly constrained.019

Overall, the process reveals the need for another aspect of architecture to provide constraint for the layout generation to be purposeful. Solving the floor plan issue alone results in designs that are unaware of other constraints such as lighting, site adjacencies as such. These aspects cannot be compartmentalized, as the feedback they provide project the generated floor plan with completely different fitness. This kind of a floor plan design would reduce architecture in this sense into something else, floor plans would resemble circuit board design or cargo packing or such.

The comparison between circuit board design and architecture provides an effective example of the fundamental differences between the engineering approach with clear objectives and unambiguous architectural design. The problem of the computational model boils down to the relationship of circulation and programmed elements attached to it. If the whole program is connected to the circulation, the circulation will end up in the center of the mass and there will be no entrance whatsoever.

This kind of an approach has sufficed for design of circuit board layouts and they are a standard practice in that field. Architecture requires a much more varied set of aspects in the construction of the layout. Such aspects that need to affect the logic of floor plan formation are the mode of

019 Coates, Programming.architecture, 166.
construction, types of spaces, types of occupation, natural light, inhabitant access and public private relation.\textsuperscript{020} This is why for the floor plan generation model to be purposeful it begs constraints such as the envelope form or conceptual redefinitions, since the program itself is able to no matter what bend into various different fit combinations. That is, it requires case (urban, rural, lighting etc.) or type (public building) specific detailing into the computational tool so that it could be purposeful and efficient in practical use. This might be the general misconception concerning the efficiency of computing to generate floor plans. The model developed here can in no way compete with designs made by humans as it remains a simplistic tryout. But implementing case specific detailing into it now would not accumulate significant knowledge concerning the overall process of floor plan generation. That was the whole purpose of refraining from case specific studies.

Is floor plan generation then a totally barren field of research as it is only capable of concentrating on such single issues? First of all, to conduct a formal study one has to separate issues to fully analyse a single one. This is what the researchers of floor plan generation have also done in order to have a clear lineage of developments, each building up on the previous. More elaborate and detailed takes have became specific tools and are undisclosed as business secrets. As such, information from them is unreliable, however in general they are widely used and worthy of the computation.

If phenomenon that is contradictory to a body of theory is found, the theory has to change, since the mere existence of the phenomenon proves the theory wrong. It does not matter if the appearance of the phenomenon is rare or highly unlikely. History is full of developments that were first considered impossible. All a computational design methodology has to do, is to produce one design that could live up to the standards of the designs by human architects, a certain Turing test. However, this seems to be still far away in the future. However, if it happened the industry would most likely go through a drastic change regarding such aspects as the methodology of design, employment, and education. By denying the

\textsuperscript{020} Coates, \textit{Programming.architecture}, 163.
possibility of the appearance of an advanced computational design method architectural industry is taking a risk. It is true that similar utopias envisioned in the 1960s and 1970s did not take place and why should they take place now? Back then the computing powers were insignificant compared to what exists today. The technology has also been accompanied by the developed knowledge of interface and usage. After the 1960s and 1970s the architectural CAD systems have drastically developed in their capabilities. iPhone, when it launched, was ridiculed for being a reminiscent of the touchscreen phones made by Nokia years before it. Yet Apple went to redefine the mobile phone industry and Nokia was obliterated.

In 2006, a design which was “fully automatically generated”\textsuperscript{021} through computational design, received the fourth place in an architectural competition.\textsuperscript{022} What this achievement means, is that artificial intelligence, in architectural design computation beat the majority of the human designers. It shows that there is great potential and it seems more likely a matter of time now, before something similar happens. Thus, a single occurrence proves that the critic’s theory of the insignificance of computational design is faulty.

\textsuperscript{021} Hovestadt and Danaher, Jenseits Des Rasters -- Architektur Und Informationstechnologie/Beyond the Grid -- Architecture and Information Technology, 236.

\textsuperscript{022} Ibid, 237.
VI. Conclusions

“We were trying things out, and in the beginning, at least from my point of view, I remember thinking that we could do it all much faster without the computer.”

- Joe Tanney\textsuperscript{001}

The previous chapter already analysed the floor plan generation process and its properties. This chapter will focus on its immediate effects on architectural design, essentially the design of program and floor plans, methodology, and the near future consequences that might surface regarding the design process.

During the recent years the cinema and game industries have significantly taken advantage of computational software when designing virtual spaces.\textsuperscript{002} There exists a great need for generating interior spaces of buildings in the gaming industry. Interactive graphics applications are increasingly situated within building interiors that can be explored. Therefore, “data-driven techniques for automated generation of visually plausible building layouts” have been researched outside the architectural discourse in digital gaming industry.\textsuperscript{003}

This, it might be asserted, is again another anomaly of architectural industry refraining from acknowledging computation as a part of the design process, and not relinquishing from the

\textsuperscript{001} Lynn, Archaeology of the Digital, 75.
\textsuperscript{002} Burry, Scripting Cultures: Architectural Design and Programming, 62.
\textsuperscript{003} Merrell, Schkuza, and Koltun, ‘Computer-Generated Residential Building Layouts.’ 1.
old conceptions of creativity and authorship. This is at least partly the case. The other part can be credited to the ineffective design of practical tools by the industry of architectural design software.004

Simultaneously, the architectural profession has been criticised for designing only a minor portion of all the constructed buildings. If the gaming industry is already taking advantage of computation to create virtual architecture is it far fetched to expect that from actual architectural design? It is of course true that one can more easily create artificial realities since there are no laws of physics. Virtual architecture can therefore incorporate unrealistic properties such as non solid objects. Yet, one can not stop wondering why computation can not, even partly, be applied to speed up and enhance the design workflow of architectural design and manufacturing?

Even architects are also producing multiple different projects throughout their career, such as housing designs, which have significantly similar design constraints, budgets, design goals and such. Computer, and namely algorithms, are exactly created for automation of certain processes. Certain parts of creative design fall into the category of repeatable tasks. These include such task as checking the scale of the sketch, calculating areas that the match the program to name a few.

In this sense, architectural discourse should open up to embrace the digital and start to teach it as a part of its curriculum. A deeper understanding of the digital in general is needed, so that when a digital tool is encountered, it can more readily be implemented into the processes of design. There is no need to wait for fully developed ready made digital tools for architects to emerge, as this would result in excessive work on already solved issues.005

Rather, architects should have a general knowledge of the digital, much like architects have a general knowledge of the arts, their tools, methods and history. If architecture still wants to be defined as residing at the intersection of arts and science, it is truly misrepresenting science.

005Burry, Scripting Cultures: Architectural Design and Programming, 64.
Methodology from computation

“Ironically the accelerating multiplication of the idiosyncratic converts it progressively into a new generic: the spectacular is absorbed back into monotonous normality by way of incessant replication. Such works bear little projective power relative to a specific situation or context precisely because of their increasing exchangeability and literal superficiality.”

- Michael Hensel

The design of computational models and their operation has peculiar similarities to the general methodology of architectural design. It is often said that one learns the subject at hand only after one tries teaching it. In the process of creating models for generating designs, one has to, in a way, teach the computer to do certain things. That is, one teaches the computer to design. This is then achieved through algorithms: specific step by step rules for the computer to follow. The operations within the algorithm are the teachings. Yet, almost always, after a given iteration of editing an algorithm, the computer ends up creating something that was not of intention, and the model has to be updated to include more information and detailing. It is thus constant re-education of the computer, but of oneself as well, to truly understand the design question at hand.

Yet this kind of a dialog included in computing offers a lot to the designer. That is, you are explicitly and instantly told what your design criterias produce, as the computer plots the designs.

006 Hensek, Performance oriented architecture, 32.
Compared to the traditional intuitive dialog between designer and his or hers design, one can easily deviate from the established design criterias because of an personal bias towards, for example, some geometrical forms one is interested of. In general terms, the computer is incapable of this kind of a humane interpretation or bending of the rules. It follows the rules no matter where they lead. Although it should be noted that different kinds of biases come into play when dealing with computers. Just as with any analog set of tools, pen and paper for example, the tools tend to guide the design process to an extent. The fundamental problem, in computational design and therefore with computers, is how the designer can prescribe and formulate the design issue at hand for the computation since the computer can only work with numbers. This conversion of a complex design issue into numerical form will contain subjective decision made by the human designer. Therefore the computer is unbiased to the extent of what is being provided for it. If the information in itself is corrupted the computer can not realize that.

The following is contemplation of different aspects of architectural design process that computation at least allows to be reassessed. I.e. it is reflecting the computational design methodologies to traditional design methodologies in architecture and it bases itself on the experiment demonstration and literature researched in this study.

**Negative generative approach**

The orthodox method for proceeding with a design process in architecture has been focusing on one design and rigorously developing it. This is the truly intuitive method of design, where one is influenced by certain aspects of the given design context — the site, the program, the budget, etc. — and the whole process of design is induced by them. These kind of designs are then qualified as good or bad based on their explicit merits that are easily readable for professionals, the designer colleague working along the project but to the jury of the competition as well. This kind of scrutiny is however highly biased, as every individual has preferences of their own and the design itself is then merely marketed and sold for these individuals. Humans are more inclined to assess the content
of information rather than the reliability of that information, that is, how it was produced in the first place. Therefore, the realm of the design task is seen to be more simple and the sole design seems more coherent than what it actually is since it is easier to make decisions based on our imagination and conceptions than on the actual real world phenomena. 007

A more progressive procedure in architectural design process, however, is to create multiple designs and then compare those alternatives in order to clearly see the properties of each design. As such, it is less intuitive and more analytical method in a sense, since it is through comparison that the positive and negative characteristics of a given design becomes truly visible and attainable for the designer. This line of thinking bases itself on the idea that it is safer to say what a design cannot be than what it actually can be. As described in chapter four, architectural problems fall into the category of NP-Hard where the correct solution and the test for that correctness are both unknown. 008 Therefore, the only method to proceed with is to compare designs and choose the more correct one, meaning it is the indirect instead of the direct expression.

Here the bias that affect is cultural and thus may be impossible to be removed since we all come from a certain culture, of design or society. Certain aspects of design parameters can be interpreted as essential while in different environments they could go unnoticed. These are certain unwritten rules handed down in professional circles from teachers to students. Such examples could be in housing design the adjacency of vestibule and toilet, in public buildings the separation of service and public. Therefore these conceptions might mean nothing in some cultures and in some they make the difference. Despite these shortcomings, the above is a viable method for exposing the profession related biases that might go undetected.

Through the tradition of critique architects, designers and scientists in general, are more equipped to define what is wrong rather than what is right. This kind of negative knowledge is far more solid than the positive knowledge that is prone to errors. Contemporary

007 Kahneman, Thinking, Fast and Slow, 118.
008 Rutten, ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ 2.
knowledge of what is right might turn out to be false, when what we know to be false is much harder to prove to be right. However, this kind of a design process requires an excessive amount of different designs in order to produce enough variability from which to start the process for design synthesis. In this sense the practise of architectural design competitions are a plausible concept. Computation could be of significant help in this kind of a design process. Not only to save time when countless processes of manually performed design construction could be replaced by algorithms, but to make the mapping of variant designs more efficient. In manual design generation one is more prone to leave the parts that constitute the whole design undefined, that is, one is seeking a design where parts are more or less made to fit the purposes of the whole. It is only at the stage of comparison between whole designs, that is complete and potentially finished designs, that the properties of parts are scrutinized. It is the design in which the parts are also of quality that the overall design is considered better. With a computer model one has to explicitly define the parts constituting the whole, i.e. one has to define the input variables and the construction of the parts. This reduces the amount of bad quality parts found in resulting designs, which in turn reduces the amount of resulted, whole designs that would have been deemed bad as well. It is thus following the same logic of negative knowledge of knowing what a part cannot be — by defining the domain of its input values and the algorithm constructing the form — and therefore reducing the amount of possible design and thus making the design process more efficient.

**Change in design initiation**

Typically, an architectural design process starts from the large scale design ethos and progresses towards the small scale detailing. Quite on the contrary, algorithms and thus digital models start to develop according to local rules defining their parts. It is only after the parts have been developed that the overall aggregate starts to exhibit some kind of an order. In this manner, computing proceeds in the
opposite direction than the human designer.\textsuperscript{010} As discussed earlier, the design rules implemented into algorithms cannot necessarily reveal the generated higher order structures of the overall aggregate model.\textsuperscript{011} This is why there exists a fundamental potential for generating novelty that could otherwise be invisible to the human designer. Thus, computing provides a completely different perspective into the design process by inverting it.

The design then starts from the definition of the parts, such as ceilings, doors, walls, ramps, corridors, floors, roofs, windows, stairs and facades, and their relations, much like in the building codes and regulations, what kind of thresholds are there which prompt certain adjustments and how freely the parts can move. This is exactly what was done in the model experiment chapter. In theory, once these definitions are set, the computer would generate a design based on these input values.

**Genocide of darlings**

The subject of objectivity in assessing designs was slightly touched upon in the previous section when a design process that constituted the development of one sole design was described. In general, designers tend to attach to their designs, which is why they developed such designs in the first place. The problem, however, rises when a design is not serving the overall goals of the task but only the interest of the designer, whatever these interests might be. In these kinds of situations the unwritten informal rule to *kill your darlings* is often prompted in order to have a sound and logical design process.

However it could be argued, that this attachment to one’s design, is part of a more general psychological bias or illusion. On a general level, once an individual has accepted a certain theory or conjecture and uses it for reflecting decisions on a regular bases, it is difficult for that person to notice it flaws. In design, the theory could be seen as the intuitive reaction based on the context, and decisions as what one allows in one’s design scene. Finally the flaws would be represented as the state when the design process is stuck and the whole


is suffering from the problematic parts. Disbelieving is a tedious task and humans are vulnerable to a theory-induced blindness.\textsuperscript{012} However, once the individual can no longer reconstruct the reasons leading to a failure, one has made a theoretical advancement and has thus escaped from the adherent mindset.\textsuperscript{013}

Computing could offer a design perspective where the designer is less exposed to the situations of adherence. Since the computational design process can be characterized as inverted, one is designing rules and processes instead of singular, fixed outcomes. The production procedure does not have to start from zero even if a major change in design conjecture is made, since the tediousness of the appliance or operation of the producing the design remains roughly the same. The outcome, however, can be drastically changed.

**Peer reviewed architecture**

Already in 1962, Christopher Alexander called for the construction of logical design structure to disclose the assumptions that they were based upon.\textsuperscript{014} In the following, certain aspects of the validity or explicitness of architectural design are reflected on the computational methodologies. As described earlier, in traditional design process one design is elaborated and developed throughout the whole design process. Generative methods have provided tools for escaping the bounds of a single design with the ability of comparison. Generative methods also include the property of explicitness, as one has to define the states or parameter settings which define the design. Yet, as stated in the conclusion of the experiment demonstration, in chapter five, it is hard if not impossible to be certain about the relations and their outcomes. However, in traditional design the intermediate decisions are often left unclear. The process is harder to scrutinize since the only thing exposed to criticism is the final design. In computing and when creating algorithms one has to be unambiguous and explicit in design operations in order to generate an outcome.

\textsuperscript{012} Kahneman, *Thinking, Fast and Slow*, 277.
\textsuperscript{013} Ibid, 279.
\textsuperscript{014} Alexander, Notes on the Synthesis of Form, 8.
In any given field of profession, the validity of a process or performance cannot be graded based merely on its results or outcomes. A more coherent way to carry out the review is to analyse all the costs of possible other outcomes. The value or novelty generated by a process where possible other outcomes could have been barren or impoverished should not be regarded as good compared to a process which steadily gives decent outcomes.

Analogy to the design process then is that the design process where the intermediate states of design operations are concealed and only outcomes valued, tells nothing of the prestige of the design methodology. Yet, this is how architectural projects including competitions and academic projects, in simplicity are judged. The following argument is not that computation is more noble methodology than the traditional because of it unambiguousness. However, one has to be unambiguous when creating digital models and their operations. It is possible to derive the process for a certain design and show how a change somewhere could have resulted in an alternative design. This kind of explicitness is usually merited to parametric design, however on a more general level, it applies to all computational processes as discussed earlier. The argument is that a computational design methodology has the ability to enhance design processes in general towards more openness and transparency where states and design decisions can be more accurately analyzed and judged. Even though a process would result in an unprosperous result, its legit process could be identified and the point of degradation or wrong design decision thus acknowledged. This could enforce the traditions of architectural design as a process, not an outcome.

Another psychological anomaly that is of interest in relation to design is that humans tend to react stronger “to an outcome that is produced by action than to the same outcome when it is produced by inaction.” For example, a design decision to elaborate on some specific addition or permutation, which one senses to be induced by the context, is an easier target for criticism than a design decision to refrain from elaborating.

---

015 Taleb, Fooled by Randomness: The Hidden Role of Chance in Life and in the Markets, 22.

016 Kahneman, Thinking, Fast and Slow, 348.
anything. Again, going back to the traditional processes of design where design process is somewhat unambiguous, design decisions concerning whether to act or refrain from acting can be more easily hidden in these kind of processes than in computing.

A final note on the explicitness of design deals with the marketing or recognition of the design question. A contemporary trend in architecture is to describe the design process of a project with diagrams of a building mass that starts from generic and progresses through intermediate stages into a unique design. Bjärke Ingels, REX and the likes are the prime examples of these kind of project descriptions. What these images are advertising, is the process of how the design brief was capable of inducing only one design — the design described through the images — and how this design then naturally emerged from the constraints of the brief.

It could be argued that the complexity of a design brief is substituted with a different, more simple question that purports the issue at hand differently. The sequence of diagrams merely finds a related question and develops an easier answer to it. This kind of advertising pitch of substituting a question with another one is part of a general tendency in human cognition.\footnote{Kahneman, Thinking, Fast and Slow, 97.}

In natural computing, designers were influenced by the analogy of parallel computations taking place in nature. They wanted to harness those computations into their own design processes by constructing generative scale models and designing with natural materials, taking advantage of their material properties to generate certain forms and shapes. In general, they took the essence of a complex process and used only that. Computing was later seen to enhance this kind of a design methodology as the digital realm could now simulate the same process. Similarly, here the advertisement sequences of projects could be liberated from their advertising duties into the actual design tools for verifying certain processes and to design with them.
Metaheuristic influences

The metaphor derived from metaheuristics allows one to regard architectural design as a search process to find the correct design which matches the requirements of the context. So, all the possible designs, in a mathematical sense, are merely waiting in the shape space to be found.\textsuperscript{018} All one has to do is to compare those in order to choose the most fit one. Needless to say, this process is not this straightforward and easy.

Yet anomalies in the realm of metaheuristics, not just the metaphor of search, allows one to analyze the architectural design tasks through them, even when not using computation. So, the drafting with pen and paper or mere contemplating of things prompts notions also present in computing.

A traditional way of drafting floor plans is to assemble rough areas in adjacent relations according to the program and then proceed to fixing the abstract areas into an actual floor plan. However, one has to draw and estimate area sizes and relations manually. This kind of professional knowhow develops through time, however, it is necessary to swap spaces and try different relations and combinations until a working design emerges. This kind of a design merits a lot when one is conscious of the vastness of design possibilities and traps of local optimas, which are in a way design methodologies exported from computing.

\textsuperscript{018} Dawkins, The Blind Watchmaker, 66.
Future implementation

“Only when the computerized version of design is the master
document from which all auxiliary information is derive,
preferably with computer assistance, will a complete computer-
aided design system have been created.”

- Ivan Sutherland

The following is a collection of educated guesses about the near future of floor plan generation and design. Despite the long history of development, and especially the recent breakthroughs in computing interfaces, namely visual algorithm editors, along with the computing power, the near future implementations of computing vary from the vision of rendering floor plan designs like images to workflows where the computing remains as it is today.

Near future tools

Architects are used to working with graphical interfaces by manipulating geometrical objects with geometrical transformations. It has not really mattered whether the tools have been digital or traditional. The visual orientation is only reasonable since the final output of architectural design is geometrical. Therefore, it would seem that hybrid tools of computation and visualization, that is

the drawing of documentation for presenting designs, to emerge in greater numbers. A software framework where textual input is intertwined with a graphical one has been out there throughout history. Despite the fact that they allow greater control of the processes within the software because of the terminal it does not make it necessarily a more intuitive design tool. This is proven by the rapid rise of graphical interfaced algorithm editors. It is a noted concern, that if the development of computing would result in the replacement of drawing with text, there would have not been any progress. Overall, the next phase of computing would seem to require communication developments. However, it remains unclear who should devote more time into learning another language, the computer or the architect? Computer programming as a subject is entering the schools at the moment. It would not be a surprise if the following generations of architects were used to writing programs as easily as we are used to drawing. Then again, it might be that since the computing interfaces have only recently started to develop the future might introduce more predictive and interpretative programming languages. The Processing environment with its libraries and Grasshopper with its graphical interface are merely early demonstrations of such software. Although there is a risk if only the latter mentioned development will take place that designers can not distinguish the actual processes behind the interface and make sense of the true relations of variables. This is already happening with graphical interfaced algorithm editors as the visual representation by nature obfuscated and represents them in a certain way. This anomaly is however not inherent to graphical interfaces but textual ones have their own representational problems.

Another angle to look at the future implementation of computing into architectural design is to consider it as a phase. As the precedent analysis of floor plan generation models and the experiment chapter itself showed, there is still no efficient process which would turn input variables into final detailed plans. There is still a great need for manual work, or work that the human intelligence is much more efficient to process. However, certain steps in design are repetitive and generic. For example, the process

---

020 Burry, Scripting Cultures: Architectural Design and Programming, 64.
of turning a sketch into a more readable formal drawing including all necessary details is a tedious and time consuming task. There is also room for greater accuracy in this translation. Thus computing, instead of a heavy framework one is forced to work with, could be intertwined into the intuitive design processes. It could then manifest itself in the form of a physical scanner, a smart pen or a ruler system. Early examples of such tools, it could be argued, are the Adobe Ink, Moleskine MSK 2 or Wacom Inkling. These physical tools allow more intuitive approach into design while enhancing it by harnessing computation. It is easy to envision pens that are aware of the scale and can also access the building program information to better guide the sketching of areas and floor plans. Geometric forms could be better grouped or snapped together through the abilities of digital rulers to form more accurate and clear images. Relaxation of elements into more coherent wholes could be done through a digital scanner, and thus speed up the process of allowing new versions of design to emerge.

**Leapfrogging technology**

In the contemporary world, technology is increasingly destroying more jobs than it is creating. 3D printing and robots are the prime examples of the technology that is taking up the labour previously carried out by workers. If the trend in manual labor and assembly line work is that less humans are needed in it, then what about more advanced jobs? There is a certain degree of repetition and strict instruction to following in every job description. Some are even speculating that 47% of jobs in the US are under a threat to be taken up by technology. Where does this leave the profession of an architect?

Technology in actual computing could also be on the brink of a new revolution in technology by the means of inception of a totally new generation of computers. The alleged new generation of computer are taking advantage of the quantum mechanics. As such, they could perform computations that would currently require

---

021 Thompson, ‘We’ve Reached a Tipping Point Where Technology Is Now Destroying More Jobs than It Creates, Researcher Warns.’

022 Frey and Osborne, ‘THE FUTURE OF EMPLOYMENT: HOW SUSCEPTIBLE ARE JOBS TO COMPUTERISATION?’ 1.
millions of years, in seconds.\textsuperscript{023} This would mean computational capacity that could instantaneously solve any encrypted password, intercept private conversations and break into banking records. Current estimations of when all this will be possible state that it will take from five to 20 years.\textsuperscript{024} An exhaustive search of the phase space would then be carried out in miniscule amount of time.

The kind of speculations these news elaborate are wild. Could floor plan design then truly be a task of setting up properties and watching the computer render the plan? Could the sequence of diagrams described by BIG and their likes become alive, as millions of parallel computations would be possible? Could the development of computing entail great drops in the amount of trained architects or just create efficient and formal workflows for design? No matter how interesting these speculations are, they do not accumulate any formal knowledge that could be used as a legit prediction. Only one thing is sure. There are no general signs of computation in retreat.

\textsuperscript{023} Wadhwa, ‘Quantum Computing Is about to Overturn Cybersecurity Balance of Power.’

\textsuperscript{024} Ibid.

268
Postscript

“The poverty of reprisal from all involved
And the scathing trajectory from the past
Markovian process lead us not in vain”

- Greg Graffin

The term iatrogenics, that literally translates as “caused by the healer”, is used to describe the problem when some procedure is producing more harm than it is producing benefits. A fundamental example is the turning point in medical history, as hospital treatment before the invention of penicillin rather than decreased, actually increased one’s chances of death. Thus iatrogenics is usually connected to medical treatment.

For our purposes here, the term will be applied to digital technologies or procedures in architecture. The history of computation in architecture is full of experiments. Few designers focus on the computation and proceeds with it as a design methodology addressing architecture. However, they often describe it as tedious or otherwise problematic. Yet, their enthusiasm and conviction has kept them returning to this process. Thus, they acknowledge the iatrogenics in their work, but still feel that the

025 Graffin and Bad Religion, ‘Markovian Process.’
026 Taleb, Antifragile: How to Live in a World We Don’t Understand, 111.
027 Lynn, Archaeology of the Digital, 12.
overall goal of development is worth the trouble. Currently the overall perception of all studio work, university lecturing and teaching seems to be emphasizing introductions to specific programs. Despite the long history of computation in architecture, it is still missing a pedagogical agenda for digital design and computation.\textsuperscript{028} The theoretical framework for floor plan generation, being one of the first accounts of computer usage, is no exception and knowledge attached to it still dispersed within in the academy without generally acknowledged body of theory. This hinders significantly the complete formalization of the agenda of floor plan generation.\textsuperscript{029} The technical and quantifiable factors can be used however the political agenda to alter the built environment has such a drastic affect on what is built that the technical, computational approach can not apply its whole potential.

The parametric insurgencies have brought the software related to it, namely the graphical algorithm editors, close to a point — similarly as the invention of penicillin change the balance of harm and benefits of medicine — of a tipping point. That is, up to a point when they start to serve design process in a more efficient way\textsuperscript{030} and the use of computation might just become part of the design on a regular basis.

Hospitals and medicine today live up to totally different standards than what they used to. In general, as medicine is, in contemporary world, increasing the quality of life and thus the benefits have surpassed harm. Even though medical procedures are still producing more deaths than any cancer, and in the United States between three to ten times more people die from medical errors than from car accidents, their significance is remarkable. The question merely comes down to another problem concerning the righteousness of the agency or the person carrying out the procedures, that whether or not, the

\textsuperscript{028} Kotnik, ‘Digital Architectural Design as Exploration of Computable Functions.’ 2

\textsuperscript{029} Merrell, Schkufza, and Koltun, ‘Computer-Generated Residential Building Layouts.’, 1.

\textsuperscript{030} Burry, Scripting Cultures: Architectural Design and Programming, 62.
procedures are benefitting the agent perceiving the procedure more than the procedure bolstering the needs and interest of the person or organization administering the procedure.031

The 1960s and 1970s saw the rise of digital computing in architecture. It gained significant follow up and variety of design theories. The field was seen promissinary and fertile, however it lost its momentum, when the computer aided software packages became more easy to use and the majority shifted from tedious programming to easily attainable software tools. It even took time for these readymade softwares to reach professional level of hand drawing. When they did in the 1980s, they took hold of design, and computing in architecture became synonymous to these tools and aids. The methodology of computing faded away.

The 1980s and 1990s were described as the time when technologies took hold of the design goals and made the use of digital tools an intrinsic value instead of the resulting design. Thus, the timeframe can be an example of the great iatrogenics of digital technologies. Today, as technology is becoming more attainable by many, coding is even taught in schools, the digital native generation is maturing, there seems to be great optimism for digital tools’ more purposeful implementation into design. It seems that we are still living the times where digital natives are a minority and thus digital technologies are only superficially applied into our everyday life and thus everyday design. It is true that we are already seeing great leaps in the attainability of computing in architecture, as visual algorithm editors are becoming more and more intuitive. Similar to medicine, based on current outlooks it seems evitable that the appliances of digital technologies and computing in architecture is approaching a point where its start to provide significantly more benefits than harm. Computation is deeply embedded in the structures of the future architectural discourse as a fundamental part of the design process and methodology.

There are no signs of revolution and no signs of retreat either. The digital sphere is becoming a more and more integral

031 Taleb, Antifragile: How to Live in a World We Don’t Understand, 111.
part of the society. It would be thus contrary to the logic that architecture, as a discipline, would remain outside of it.
Acknowledgements

I would like to thank my thesis supervisor Pirjo Sanaksenaho and thesis advisor Otso Helenius for their assistance and professional counselling. I have benefitted greatly from the remarks and suggestions of Maija Parviainen throughout the development of the manuscript.

I would like to thank Toni Kotnik and Teemu Seppänen for thought-provoking discussions on architectural computation. Also Grete Soosalu and Raul Kalvo for their generosity to provide me with information and point of view to map the field of computation in architectural design.

In the beginning of the research I had the advantage of being guided by conversations with Mikko Heikkinen, Eero Lundén, Markus Wikar and Ashish Mohite. Therefore I would also like to thank them.

Finally I would like to thank the people at studio N:S for the thesis debates. Especially Thomas Miyauchi and Johannes Kaira for ideas and criticism. Those to whom I cannot do justice will understand.
Appendix

In the following the technical illustrations of the Grasshopper definitions used in the experiment demonstration are shown. The actual file from which these illustrations are taken from, is provided to the Aalto eAge service.
Individual elements components and their connection points to the overall construct.
Planarity definition.

Fitness variable scaling definition.
Adjacency of all elements definition.
Overlap definition.
Element area definition.

Program adjacency definition.
Boundary related definitions.
Boundary visualization definitions.
Boundary related definitions.
Bounding box subtraction definitions.
Adams, Tim. ‘The Eisenman-Deleuze FOLD.’ Edited by The University of Auckland, 1993.


Carpo, Mario. ‘Ten Years of Folding.’ Folding in Architecture, n.d.


http://www.danieldavis.com/a-history-of-parametric/.

Davis, Daniel. 'Modelled on Software Engineering: Flexible Parametric Models in the Practice of Architecture.' School of Architecture and Design College of Design and Social context RMIT University, 2013.


Rutten, David. ‘NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING.’ *NAVIGATING MULTI-DIMENSIONAL LANDSCAPES IN FOGGY WEATHER AS AN ANALOGY FOR GENERIC PROBLEM SOLVING*, 2014.


Thompson, Cadie. ‘We’ve Reached a Tipping Point Where Technology Is Now Destroying More Jobs than It Creates, Researcher Warns.’ *Business Insider*. 

292


The supposed algorithmic architecture has recently emerged as a distinct field within architecture. It is infested with ambiguous research while no rudimentary study of how to practise computational design in general exists. This thesis thus studies the process of promptly harnessing a computational design methodology and applies it to floor plan design.

The research methodology consists mostly of literature however a demonstration computation is carried out to support it. The research is divided into six parts, elucidating the process of constructing a computational model for generating floor plans.

There exist a lot of misconceptions about design computation in architecture. This is mostly because methodology is acquired from computer sciences through informal studies.

Curvilinear geometry and generative methodologies are not intrinsic to computational architecture. The digital computer however provided significant processing capacity that elevated computational design into a field of its own.

Much of the innovations in computational design took place in 1960s and 1970s. Only the recent developments in technology and software, especially the graphical algorithm editors, have provided new momentum and wider acceptance for computational design.

There exists formal computational precedent models capable of generating floor plans. The scope of quality vary from designs barely recognizable as a floor plan to designs that have succeeded in architectural competitions. There is a lineage of models that incorporate higher-level algorithmic constructs: Metaheuristics. The subject is introduced and evolutionary metaheuristic are explained in general since they are most widely used in architectural computations.

A specific evolutionary algorithm, found in Galapagos solver, is analysed as it used in the model demonstration, which in turn showed that the floor plan generation has many drawbacks and it is a tedious task. However it is possible to generate schematic designs of architectural program with reasonably primitive setup.

The computational methodology has explicit similarities with architectural design methodology. A coherent understanding of them would enhance the whole architectural profession.