



Universidade do Minho
Escola de Engenharia

Mohsen Parsa

**Smart BIM objects for intelligent
modular construction**

Smart BIM objects for intelligence
modular construction

Mohsen Parsa



European Master in
Building Information Modelling



UMinho | 2021

The European Master in Building Information Modelling is a joint initiative of:



Universidade do Minho

Univerza v Ljubljani



POLITECNICO
MILANO 1863



Co-funded by the
Erasmus+ Programme
of the European Union

October 2021



Universidade do Minho
Escola de Engenharia

Mohsen Parsa

Smart BIM objects for intelligent modular construction



European Master in
Building Information Modelling

Master Dissertation
European Master in Building Information Modelling

Work conducted under supervision of:
Maria Isabel Brito Valente



Co-funded by the
Erasmus+ Programme
of the European Union

October, 2021

AUTHORSHIP RIGHTS AND CONDITIONS OF USE OF THE WORK BY THIRD PARTIES

This is an academic work that can be used by third parties, as long as internationally accepted rules and good practices are respected, particularly in what concerns to author rights and related matters.

Therefore, the present work may be used according to the terms of the license shown below.

If the user needs permission to make use of this work in conditions that are not part of the licensing mentioned below, he/she should contact the author through the RepositóriUM platform of the University of Minho.

License granted to the users of this work



Attribution

CC BY

<https://creativecommons.org/licenses/by/4.0/>

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisor Professor Maria Isabel Brito Valente, for her valuable support and guidance during the development of this research work.

I would also like to thank the support provided by Construsoft, especially to Vakis Kokoris and Nuno Pires. They contributed with their insights on the first stages of the research and provided access to the BIM tools used for developing this work.

I gratefully acknowledge the support of Electrofer company and Fractus company in Marinha Grande, Portugal, especially Marta Gregorio for her support during this research. I would like to tanks Hugo Sousa and Philipe Monteiro, engineers of Electrofer and Fractus, for their support during my settlement in Marinha Grande.

My thanks to the staff and professors of BIM A+, particularly to Professor Pietro Crespy, Professor Miguel Azenha and Professor Tomo Cerovcek, for their support throughout the master.

I am grateful to my friends Dr Saeid Zarrinmehr and Pouya Parsa for motivating and helping me find the way when I was stocked with Python.

I would like to thank my parents Mohammad Hossein and Homa, my sister Sara and my cousin Victoria Parsa, also my father-in-law Mohammad Reza Farzanehneghad and Mother-in-law Mitra Nezakati Roshan for their unconditional support and encouragement since the beginning of this journey.

Finally, I would appreciate my wife, Roya that supported me, tolerated this far distance for one year and took care of our child, Karen. I cannot express her sacrifice by words.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

A handwritten signature in black ink, enclosed within a large, hand-drawn oval. The signature appears to read "Mohsen Parsa" with a stylized flourish at the end.

RESUMO

A AEC é uma das maiores indústrias mundiais e , em comparação com outras indústrias, como a da manufatura, apresenta uma das menores taxas de produtividade. Uma das principais soluções para aumentar a taxa de produtividade é mudar os processos de produção . Cada edifício é um produto único e a abordagem usual de manufatura - Produção em massa - não pode ser adaptada diretamente para a indústria de AEC. Graças às novas tecnologias, surgiu um novo método: o ETO ou Engineered-To-Order. Nesta abordagem, os produtos podem ser fabricados correspondendo ao projeto final do produto de construção. Um desafio importante deve ser resolvido na abordagem ETO, as propriedades dos elementos devem ser extraídas dos documentos de projeto. Este problema pode ser resolvido com BIM e com Inteligência Artificial.

O Building Information Modeling (BIM) tira proveito dos poderosos processos digitais e das novas tecnologias de programação, como as linguagens de programação orientada a objetos (OOP). Com o BIM, é possível criar um modelo 3D completo e enriquecido por diversas informações não gráficas do produto final.

Associar o BIM ao método Engineered-To-Order torna os processos mais fáceis. Além disso, existem outras tecnologias embutidas no BIM que tornam os processos totalmente automáticos: Design Paramétrico e Inteligência Artificial que deram origem aos Objetos BIM Inteligentes. Ao aplicar essas tecnologias, o projeto e a modelação de elementos podem ser feitos de forma automática e otimizada.

No contexto atual, como a indústria de AEC aproveita cada vez mais a inteligência digital adaptável representativa de elementos de construção. Este trabalho buscou implementar IA utilizando para desenvolver a aplicação automática de painéis de parede, de pavimento e de cobertura. Esta pesquisa concentra-se em explorar como os recursos BIM e a inteligência artificial podem apoiar os processos de produção na abordagem ETO. Esta dissertação tenta utilizar o ambiente BIM para automatizar o processo de revestimento de um piso ou parede com aberturas.

No método de produção ETO, é difícil encontrar o equilíbrio entre a liberdade do *design* e os limites de produção. Como é que se pode transferir os limites do processo de produção para o projetista? Existem várias abordagens, mas a que é escolhida nesta pesquisa é a Inteligência Artificial. O desafio consiste em projetar painéis para algum tipo de elemento de construção (parede, piso e telhado), considerando as regras e restrições da produção. Esta pesquisa tenta vincular a forma dos elementos ao processo de produção dos painéis, tendo em consideração a visão do projetista e as regras do fabricante. Para tal, foi desenvolvido um algoritmo que permite criar painéis ETO ajustados aos elementos que o projetista definiu para a edificação, incorporando as regras do processo de produção. O algoritmo inclui a forma do elemento, começando por dividi-lo em partes menores e apropriadas ao painel, as modifica e finaliza para o painel. Em seguida, começa a organizar os painéis e define o tamanho dos mesmos de forma otimizada, atendendo às regras e limites de produção.

O algoritmo desenvolvido usa a interface de programação Python do Grasshopper para as entradas e saídas e usa o Tekla Structures como um software de autoria BIM. Este processo automatizado reduz o tempo de trabalho e os custos resultantes, e também produz resultados mais precisos e otimizados. O algoritmo é capaz de incorporar regras que são definidas pelo utilizador e calcular o resultado que corresponde a essas regras.

Palavras-chave: BIM, Smart BIM Objects, Engineered To Order (ETO), Painel Pré-fabricado, Python, Grasshopper, Tekla Structures.

ABSTRACT

From a general perspective, AEC is one of the largest industries and surprisingly, when compared to other industries such as the automobile and airplane production, it shows one of the lowest productivity rates. One of the main solutions for increasing productivity rate is to shift the production processes from traditional to manufacturing processes. But there is a problem: Each building is a unique product and the usual manufacturing approach – Mass Production – cannot be directly adapted to the AEC industry. Thanks to new technologies, a new method has emerged: ETO or Engineered-To-Order. In this approach, the products can be fabricated corresponding to the design of the final product (building). An important challenge should be solved before: in ETO approach, the properties of the elements should be extracted from design documents. Hopefully, this problem can be solved with BIM and Artificial Intelligence.

Building Information Modelling (BIM) takes advantage of powerful digital processes and new technologies in programming, such as Object-Oriented Programming languages (OOP). With BIM, it is possible to create a complete 3D model enriched by various non-graphic information of the final product. Associating BIM to Engineered-To-Order method makes the overall processes smoother and easier. In addition, there are other technologies embedded in BIM that make processes completely automatic: Parametric Design and Artificial Intelligence that led to Smart BIM Objects. By applying these technologies, designing and modelling elements can be done automatically in an optimized way.

Within the current context, as the AEC industry increasingly takes advantage of intelligence adaptable digital representative of building elements. This work tries to implement AI in the work of panelling floors automatically. The research focuses on exploring how BIM capabilities and Artificial intelligence can support the production processes, in the ETO approach. To be more precise, this dissertation tries to utilize the BIM environment to make the process of panelling a floor or a wall with openings automatic.

In ETO production method, it is very difficult to find a balance between the design freedom and the production limits. How is it possible to integrate the limits of production process into the design process? There are various approaches but the one used in this research, is Artificial Intelligence. The challenge is to design panels to be applied in building elements (wall, floor, and roof), considering the production rules and some restrictions. This research tries to associate the shape of elements with the production process of panels while considering the designer's desires and the fabricator's rules. Therefore, an algorithm was developed to create ETO panels fitted to construction elements that the designer created for the building, considering the various rules of production process. The algorithm collects the shape of element, starts to divide it to some smaller parts appropriate for panelling, modifies and finalize them for panelling. Then starts to arrange the panels and defines the size of them in an optimised way responding to production rules and limits.

The algorithm developed uses the Python programming interface of Grasshopper and for the inputs and outputs, uses Tekla Structures as a BIM authoring software. This automated process not only reduces the time of work and resulting costs, but also produces more precise and optimized results. This algorithm is capable to incorporate rules that are defined by the user and calculates the result that correspond to those rules.

Keywords: BIM, Smart BIM Objects, Engineered To Order (ETO), Pre-fabricated Panel, Python, Grasshopper, Tekla Structures.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	III
RESUMO	V
ABSTRACT	VI
TABLE OF CONTENTS	VII
LIST OF FIGURES	IX
LIST OF TABLES	XI
1. INTRODUCTION.....	12
2. LITERATURE REVIEW.....	14
2.1. INTRODUCTION.....	14
2.2. MODULAR CONSTRUCTION.....	15
2.2.1. DEFINITION	15
2.3. PREFABRICATION DOMAINS	17
2.4. MODULAR CONSTRUCTION.....	19
2.4.1. MODULAR CONSTRUCTION PROBLEMS.....	20
2.4.2. SIGNIFICANCE IN THE CONSTRUCTION PRACTICE.....	20
2.5. PREFABRICATED PANEL CONSTRUCTION.....	22
2.5.1. INTRODUCTION.....	22
2.5.2. DEFINITION AND HISTORY	23
2.5.3. PREFABRICATED PANELS	23
2.6. MASS CUSTOMIZATION AND PLATFORM DESIGN.....	24
2.7. SMART BIM OBJECT	24
2.7.1. INTRODUCTION.....	24
2.7.2. DIFFERENT SOLUTION FOR CREATING SMART BIM OBJECT IN BIM PLATFORMS:	25
3. METHODOLOGY.....	26
3.1. RESEARCH SCOPE AND LIMITATIONS	26
3.2. MAIN PHASES	26
3.3. KEY ASSUMPTIONS.....	27
3.4. IDENTIFICATION OF THE PROBLEM	28
3.4.1. THE ARRANGEMENT OF PANELS ON A WALL, FLOOR OR A ROOF:.....	28
3.4.2. DESIGNING THE DETAILS OF PANELS.....	28
3.4.3. ISSUES RELATED TO THE DESIGN PROCESS:	28
3.4.4. ISSUES RELATED TO PRODUCTION PROCESS:.....	29
3.4.5. WHICH CHALLENGES WERE SELECTED TO DEAL IN THE RESEARCH AND WHY? 29	
3.5. STEPS OF THE SOLUTION	30
3.6. ANALYSING THE CHALLENGE:.....	30

3.6.1.	ASSUMPTIONS	31
3.6.2.	RULES AND RESTRICTIONS.....	31
3.6.3.	OPTIMIZATION GOALS	31
3.7.	DEFINE AND EVALUATE THE TOOLS AVAILABLE:.....	31
3.7.1.	MAIN STAGES	31
3.7.2.	SELECTING SOFTWARE.....	32
3.8.	OPTIMIZATION	33
3.9.	BLIND OPTIMIZATION	33
3.9.1.	CONS AND PROS:.....	38
3.10.	RATIONAL OPTIMIZATION	38
3.10.1.	CONS AND PROS:.....	40
3.11.	CONCLUSION	41
4.	RATIONAL OPTIMISATION SOLUTION	42
4.1.	MAIN PARTS:.....	42
4.1.1.	INPUT ALGORITHM:	43
4.1.2.	DIVISION ALGORITHM:	43
4.1.3.	FROM HUMAN METHOD TO COMPUTER METHOD.....	45
4.1.4.	STRUCTURE OF THE ALGORITHM	46
4.2.	JOINING ALGORITHM:	53
4.2.1.	MAIN CONCEPT	53
4.2.2.	STRUCTURE OF JOINING ALGORITHM:.....	54
4.2.3.	ETAILED WORK-FLOW	56
4.3.	PANELISING ALGORITHM.....	63
4.3.1.	MAIN CONCEPT	63
4.3.2.	STRUCTURE OF THE PANELLING ALGORITHM.....	64
4.3.3.	DETAILED WORK-FLOW.....	65
5.	CONCLUSION AND FUTURE DEVELOPMENTS	74
	REFERENCES	76
	LIST OF ACRONYMS AND ABBREVIATIONS	79
	APPENDICES.....	81
	APPENDIX 1: PANELLING CODES IN PYTHON	81
	APPENDIX 2: HIGH QUALITY FIGURES.....	105

LIST OF FIGURES

Figure 1 – Index of labour productivity for construction and nonfarm industries, 1964-2009	14
Figure 2 – Structure of literature review	15
Figure 3 – Degree of Prefabrication (Smith, 2010)	16
Figure 4 – Labour productivity indices of manufacturing, on-site, and off-site construction in period 1967 – 2015 in the American market	22
Figure 5 – Proposed framework	27
Figure 6 – General work-flow of analysing the problem	31
Figure 7 – Domain of solution and related tools	32
Figure 8 – Three stages of solution and selected tools and software	33
Figure 9 – Moving arrangement in X and Y directions by length and breadth of a panel to generate new arrangements	34
Figure 10 – An overview of blind optimization in Rhino and Grasshopper	34
Figure 11 – General work-flow of Blind optimization	35
Figure 12 – Octopus plug in and generative design	36
Figure 13 – Grasshopper script, Analysing part	37
Figure 14 – Grasshopper script, Panelling part	37
Figure 15 – Grasshopper script, Optimization part	37
Figure 16 – A sample result of Blind Optimisation by grasshopper and Octopus	38
Figure 17 – General work-flow of Rational Optimization solution	39
Figure 18 – A sample result of Rational Optimisation by Python in Grasshopper	40
Figure 19 – Main stages of Rational optimisation solution	42
Figure 20 – Alternatives of solution carried out by human	43

Figure 21 – General work flow of Rational optimisation solution	44
Figure 22 – A geometrical sample of different steps of Rational optimization solution	44
Figure 23 – digital work-flow of rational optimisation extracted from Human work-flow	46
Figure 24 – Detailed process-flow of Division Algorithm (Part 1)	50
Figure 25 – Detailed process-flow of Division Algorithm (Part 2)	51
Figure 26 – Detailed process-flow of Division Algorithm (Part 3)	52
Figure 27 – Regular and irregular Rectangles resulted from Division Algorithm	53
Figure 28 – Detailed work-flow of joining Algorithm	54
Figure 29 – Detailed process-flow of Joining Algorithm (Part 1)	60
Figure 30 – Detailed process-flow of Joining Algorithm (Part 2)	61
Figure 31 – Detailed process-flow of Joining Algorithm (Part 3)	62
Figure 32 – Rule of dividing breadth of panels, regular panels and irregular panels	63
Figure 33 – production rules of breadth and length of panels	64
Figure 34 – Work-flow of Panelling Algorithm	65
Figure 35 – Arrangement of panels in a same rectangle in both direction X and Y	66
Figure 36 – Arrangement of panels in both direction X and Y and the results: 10 panels in X direction and 9 panels in Y direction.	67
Figure 37 – Detailed process-flow of Panelling Algorithm (Part 1)	71
Figure 38 – Detailed process-flow of Panelling Algorithm (Part 2)	72
Figure 39 – A sample result of Algorithms calculation	73

LIST OF TABLES

Table 1 - Prefabrication domains. Adapted from (Lopes et al, 2018).	19
Table 2 – evaluating possibility of solving the challenges	29
Table 3 – comparison two solution (Blind optimization and Rational optimization)	41
Table 4 – data types and function of Division Algorithm	49
Table 5 – Evaluation of direction of junctions	55
Table 6 – Evaluation of the number of corners of junctions	55
Table 7 – Evaluation of joining to the larger or smaller neighbour.	56
Table 8 – data types and function of Joining Algorithm	59
Table 9 – data types and function of Panelling Algorithm	70

1. INTRODUCTION

In Building Information Modelling (BIM) technology, a building can be represented by a set of objects that carry detailed information about how they are constructed and capture the relationship with other objects in the building model. Each building model has typical building rules and relations that can be predicted and defined by a few parameters and constraints. Nowadays, there is a general trend in AEC industry that demands cheaper, faster, safer and more productive methods for design and fabrication of construction components. Mass production and “Engineered To Order” (ETO) products can help the industry to achieve these goals.

This dissertation addresses the combined use of new and powerful methodologies, such as Smart BIM objects, Modular construction, and Artificial Intelligence, in the design and production of building components. More specifically, the work focuses the development of BIM objects for modular panels used for walls, floors or roofs, that can be shared, adapted, and reused across different modular projects within any stage in the life cycle. It is intended to use artificial intelligence in two stages: First that each of the BIM tools and smart objects encapsulate the domain knowledge that allows it to be inserted in a building, with high quality and in an efficient manner, and second using AI to implement a defined modularization pattern on a building wall, floor or roof considering the panel elements properties.

Domain-specific BIM tools, combined with parametric rules, enable better informed design decisions. It is intended to develop procedures related to the use of smart BIM objects on modular construction, so that they are capable of transforming their properties automatically and of identifying requirements for object evolution throughout the building life cycle.

The main objectives of the work are listed below:

- To develop a workflow that enables practitioners to generate Smart BIM Objects;
- To define how rules, specifications, properties, size and relations of modular building elements can be adopted in Smart BIM Objects and how these Smart BIM Objects can automatically respond to their position in model;
- To make a modularization workflow using AI for specific parts of an assumed building, and use it as a base for locating smart BIM object in the model of a modular building;
- To optimize the geometrical properties of smart building objects in a modular building due to rules, defined requirements, relationships between elements, and so on;

The work development is divided in the following activities:

- Reviewing the concepts of prefabrication in construction;
- Identification and characterization of different prefabricated panel solutions that are already used in buildings, including the manufacturing processes associated;
- Identification of inbuilt capabilities of BIM authoring software for creating and maintaining smart BIM objects;

- Exploration of visual programming applications and AI, such as, Grasshopper and Galapagos, in modulation process of a simple building;
- Exploration of capabilities and potentials of programming languages like Python in Smart BIM object creation;
- Development of a workflow for design, analysis, and optimization of a floor for building applications;
- Embedment of required rules, specifications, properties, size and relations in modular building elements in examples of Smart BIM Objects that automatically respond to their position in the model;
- Application of the developed Smart BIM objects in a case study

2. LITERATURE REVIEW

2.1. Introduction

The complexity of Architecture, Engineering, and Construction (AEC) industry is growing every day. In addition, the need for new construction in any field such as commercial, residential, manufactures, and so on, grows continuously in a competitive manner. Therefore, improving construction quality, save money, and shorten construction schedules are vital issue now. Beyond these, there are variety of challenges that AEC industry has to deal with. New technologies and experiences in general aspects like Automatic Manufacturing, Robots, Artificial Intelligence, Information technologies, Energy saving and so on, have improved almost all types of industrial activities but it seems that AEC industry has not progressed proportionately.

AEC industry has some special characteristics that cause it to remain in its traditional forms:

- People's desire to have a unique product.
- The final products are unable to move, export, or import to or from another country.
- Dependency on geographic location and climate.
- Complexity and participation of various trades in production processes.

The mentioned factors and many other reasons put AEC industry in a special condition where it is difficult to implement common or new industrial technologies. Here some evidences are discussed.

Eastman *et al.* (2011) have demonstrated low labour efficiency in AEC industry: Extra costs associated with traditional design and construction practices have been documented through various research studies. Figure 1 illustrates the productivity within the U.S. field construction industry in comparison to all nonfarm industries over a period of 45 years, from 1964 up to 2009.

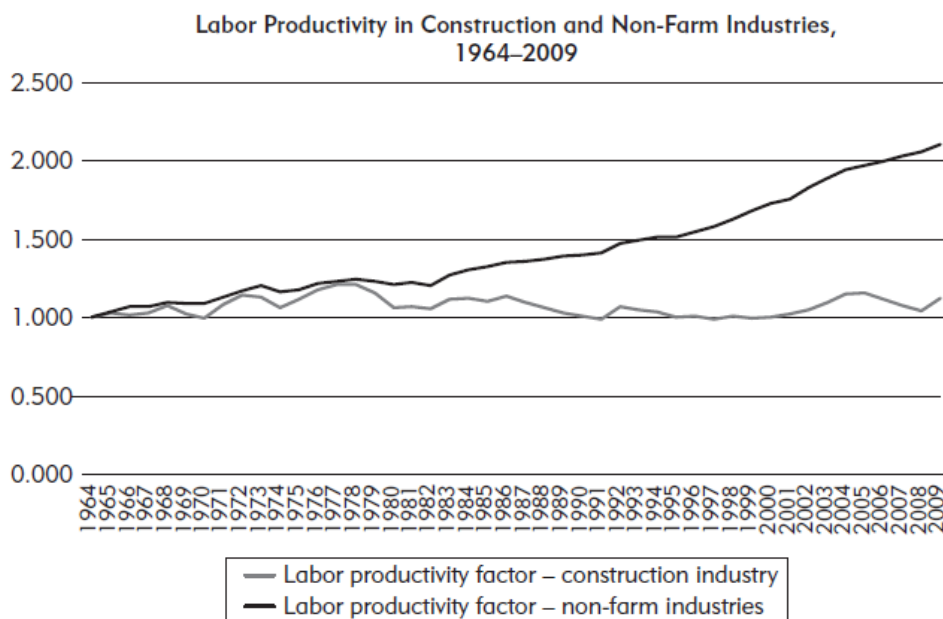


Figure 1 – Index of labour productivity for construction and nonfarm industries, 1964-2009; BIM handbook, Chuck Eastman et al. 2011

In last decades, several technologies have been introduced in the AEC industry that significantly influenced the productivity. These include:

- Building Information Modelling as a new approach to AEC industry focused on information, data-flow, and visualisation.
- Off-site Construction that has been borrowed from manufacturing
- Modular construction

According to the topic of this research, the focus of the work is on the following main questions:

- How do BIM and Modular construction, combined together, can be explored in AEC construction?
- How can they be narrowed down and practical in AEC industry?

According to the main objectives of this dissertation, the suggested diagram for literature review is presented in figure 2.

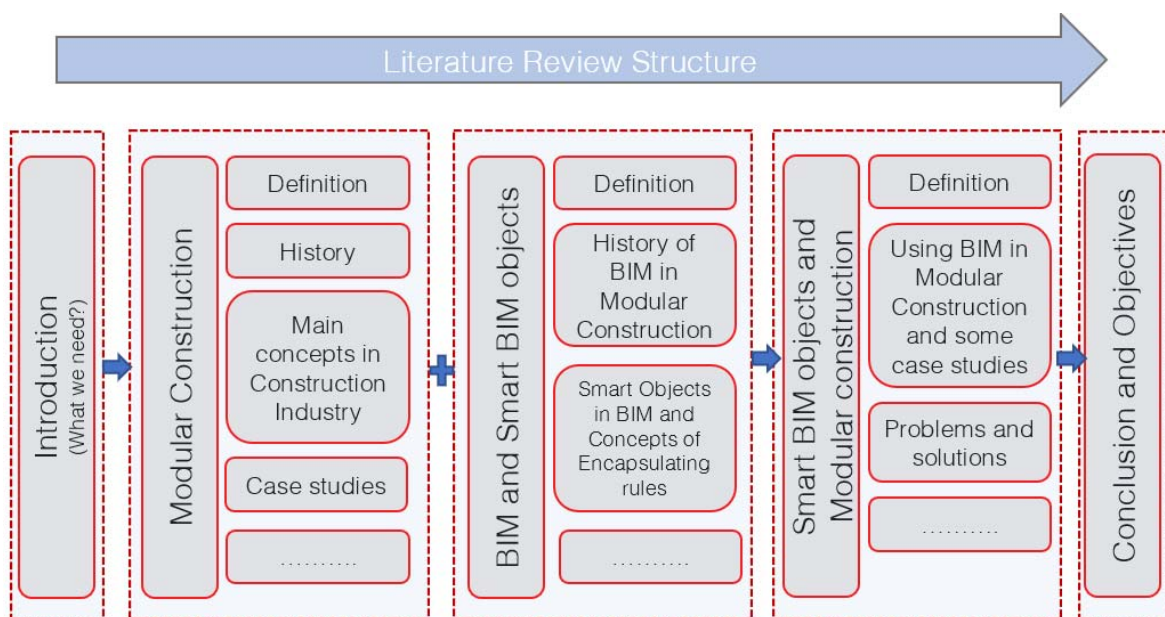


Figure 2 – Structure of literature review

2.2. Modular construction

2.2.1. Definition

Prefabrication can be categorized in types of materials or degree of prefabrication, type of manufacturing, technologies deployed or other approaches.

For example, Sacks et al. (2018) categorizes prefabricated construction in three main groups, considering a component-based approach: “

- *Made-to-stock components: These are mass-produced components such as plumbing fixtures, or drywall panels.*
- *Made-to-order components: These are components produced after a client’s order based on certain shapes and measurements defined in catalogues, such as windows and doors.*
- *Engineered-to-order components (ETO): These components need to be custom designed and engineered prior to production such as precast concrete components and façade panels.* “

From another perspective, prefabricated construction can be divided by their material types: that means any common materials that are used to build the prefabrication components, such as timber, concrete, steel or sometimes the combination of different materials, in the factory.

Third, degree of prefabrication is a construction process through which prefabricated elements are assembled on site, from the small piece to the big piece, for example, prefabricated column, panels, tilt-up, and modular methods (Smith, 2010).

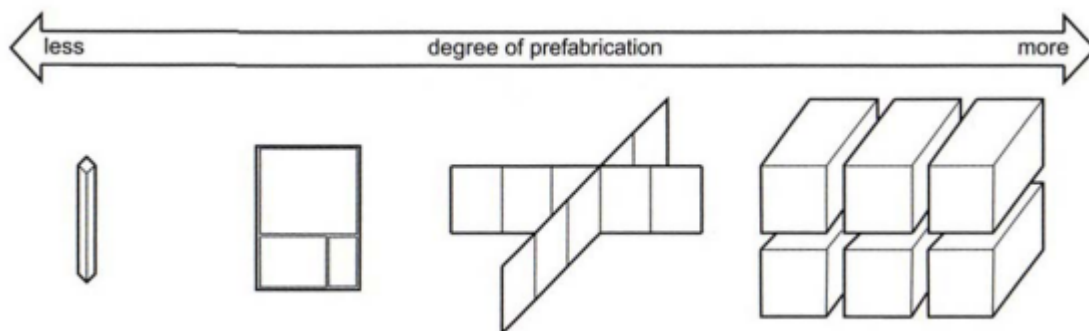


Figure 3 – Degree of Prefabrication (Smith, 2010)

Three principles of standardization, prefabrication, and systems building are the roots strengthening the industrialization in construction. Building components standardization was a prerequisite for the production in factory conditions which goes together with dimensional coordination that permit the growth of systems building. (Nawari,, 2012.)

So, from another view, offsite pre-assembly can be subdivided into following four levels (Taylor and HSE, 2012),based on increasing amounts of preassembling process and standardization involved.

- Component manufacture & sub-assembly
- Non-volumetric pre-assembly
- Volumetric pre-assembly
- Whole buildings

Prefabrication is the practice of assembling components of a structure in a factory or other manufacturing site, and transporting complete assemblies or sub-assemblies to

the construction site where the structure is to be located. The term is used to distinguish this process from the more conventional construction practice of transporting the basic materials to the construction site where all assembly is carried out.

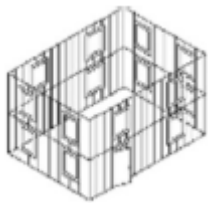
The theory behind the method is that time and cost are saved if similar construction tasks can be grouped, and assembly line techniques can be employed in prefabrication at a location where skilled labour is available, while congestion at the assembly site, which wastes time, can be reduced. The method finds application particularly where the structure is composed of repeating units or forms, or where multiple copies of the same basic structure are being constructed. Prefabrication avoids the need to transport so many skilled workers to the construction site, and other restricting conditions such as a lack of power, lack of water, exposure to harsh weather or a hazardous environment are avoided. Against these advantages must be weighed the cost of transporting prefabricated sections and lifting them into position as they will usually be larger, more fragile and more difficult to handle than the materials and components of which they are made. (<https://en.wikipedia.org/wiki/Prefabrication>)


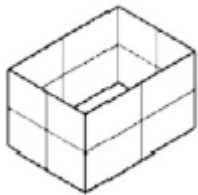
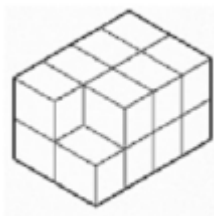
Prefabrication in AEC industry is an expanded and general term applied to variety of approaches in AEC industry that building components are fabricated off-site, shipped to the field and erected and installed to complete the building. These approaches include two main types of prefabrication, namely volumetric (often referred to as 'modular') and panellised.

2.3. Prefabrication domains

Currently, prefabrication is applied in several areas and components. As part of the construction of buildings, the prefabrication domains can be divided according to the degree of complexity of the system and consequent requirement of work on site. In a first phase, the prefabrication domains can be divided into volumetric or non-volumetric systems. That said, in Table 1 the different existing prefabrication levels are illustrated, as well as examples of their application (Lopes et al, 2018).

Table 1 - Prefabrication domains. Adapted from Lopes *et al* (2018).

Levels and Categories		Subcategories	Definition	Example	Scheme
Non-volumetric systems	1 Components and sub-elements	Prefabricated components (elements simple)	These elements, although predominantly associated with traditional methods, are	Doors, windows and stuffing elements	

		<p>produced in the factory, in order to optimize their application on site. These do not allow the total construction of the buildings.</p>	<p>Precast concrete elements (foundations, stairs, pillars, beams, etc.) and truss systems</p>	
<p>2 Panel-based construction (2D)</p>	<p>Panels of coating</p>	<p>Factory-produced panels that can be pre-finished before being transported to the site, where they are later assembled to a pre-existing structure, creating / sharing spaces</p>	<p>Facade panels (ventilated wall system)</p>	
	<p>Vertical panels</p>		<p>Prefabricated interior walls and walls</p>	
	<p>Horizontal panels</p>		<p>usable or simple coating.</p>	<p>Prefabricated slabs and roofs</p>
<p>Volumetric systems</p>	<p>3 Complete volumetric construction (3D)</p>	<p>Small three-dimensional units fully realized and finished at the factory, including water and electricity. These modules are directly installed on the building slab.</p>	<p>Modular bathrooms and kitchens</p>	

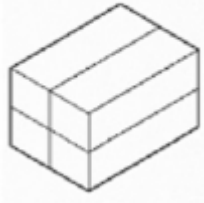
Modular systems	<p>Large three-dimensional units, fully adjusted before being transported, so that they are assembled on the foundations or other modules, thus realizing the structural shape of the building.</p>	<p>Highly standardized schemes: hotels, prisons, medium size and dorms</p>	
-----------------	---	--	---

Table 1 - Prefabrication domains. Adapted from Lopes *et al* (2018).

It should be noted that some authors consider the existence of a fourth category identified in the literature as a hybrid construction system, corresponding to a semi-volumetric construction. This solution translates into the connection between volumetric (modular) and non-volumetric systems, representing the possibility of using different solutions in the same project, increasing constructive flexibility. As an example, this solution can be used in modules for kitchens and bathrooms, in parallel with panel systems for the remaining elements (Lopes et al, 2018). In the course of this work, the theme of non-volumetric systems will be developed, more specifically, vertical panels.

2.4. Modular construction

There is a precise and comprehensive definition for Modular Construction in Wikipedia: “*Modular construction is a form of offsite production of building structural units which are then assembled onsite to complete the construction of a building: the modular units form the structure of the building as well as enclosing usable space. Modular construction is particularly popular for hotels, educational facilities such as classrooms and student residences, and healthcare facilities. This is due to the economies of scale available from many similar sized modules and the particular benefit of reduced on-site construction time.*” (https://en.wikipedia.org/wiki/Modular_construction)

A modular building is a prefabricated building that consists of repeated sections called modules. (Lacey *et al*, 2018). Modularity involves constructing sections away from the building site, then delivering them to the intended site. Installation of the prefabricated sections is completed on site. Prefabricated sections are sometimes placed using a crane. The modules can be placed side-by-side, end-to-end, or stacked, allowing for a variety of configurations and styles. After placement, the modules are joined together using inter-module connections, also known as inter-connections. The inter-connections tie the individual modules together to form the overall building structure.

2.4.1. Modular Construction Problems

- Different from traditional construction

Despite the expected increase of the modular construction market, a recent modular construction study developed by Nick Bertram et al (2019) highlighted that outdated decision-making models are hindering construction organizations from breaking out of the antiquated and traditional stick-built construction processes. In fact, modular construction and traditional stick-built differ in many aspects including design, engineering, transportation requirements, logistics, collaboration requirements, and others (Mohamad Abdul Nabi,2020).

- Decision making process

Modular construction includes different project aspects and phases where the stakeholders of the projects are required to make appropriate decisions. However, the old decision-making processes and planning techniques do not necessarily reflect the complex and unique requirements of modularization in construction projects. According to Smith (2011), some of the decision-making processes in modular construction projects include those related to: (1) the assessment of modularization feasibility and its impact on different project objectives; (2) establishment of the modules' configuration of the structure for off-site manufacturing, transporting, assembling, and disassembling; (3) project development where the percentage of work is proportioned between on-site and off-site factories; and (4) project planning where module inter- faces, off-site work scheduling, and logistics are addressed to minimize costs and lead times.

2.4.2. Significance in the construction practice

A US study looked at the 48-year period between 1967 and 2015 and concluded that the productivity of the traditional construction industry has nearly remained the same, while off-site based construction has enjoyed increases in measured productivity (Figure 4) (Sacks et al., 2018, p.10). This confirms the increased value obtained from off-site construction. The benefits of off-site based construction have been extensively studied in literature, and they include:

- Time savings

Project delays due to unexpected site or weather conditions may incur significant costs due to the failure to meet the project schedule. Prefabrication enables reduced schedule durations and parallel production, assembly, and erection operations, which is often not affected by bad weather conditions (Hardin and McCool, 2015, p.29).

- Increased quality

Factory-produced parts have better quality and tighter tolerances than site-built parts thanks to more robust quality control procedures (Rathnapala, 2009, p.22). Standardized factory products are also more guaranteed to fit-in together (Khalili, 2013, p.31).

- Improved cost-efficiency

Factors to consider include decreased installation cost, lower site occupancy, increased mechanization, reduced construction duration, and reduction of scaffolding (Gibb, 1999, p.38, Khalili, 2013, pp.30-32).

- Better for the environment

Relying on factory-produced components ensures less material wastage, which can happen more often on-site (Rathnapala, 2009, p.20), with reported waste reduction levels up to 52% compared to traditional construction (Jaillon et al., 2009, pp.309-320). Material savings could also be achieved by using more efficient components that can only be factory-built, like hollow-core slabs (instead of solid slabs) (Bachmann and Steinle, 2011, p.5).

- Safer working environment

Factory conditions usually ensure a safer, more controlled environment for workers, where safety and security standards can be more strictly applied.

- Improved standardization and modularization

Building parts can be structured in modules, where similar modules are grouped in one type (Mohamad et al., 2013, pp.289-298). Modularity is key to achieving mass-customization because it helps standardize repetitive building components (Farr et al., 2014, pp.119-125).

- Layout flexibility

Some prefabricated components can offer construction solutions that are otherwise not doable on-site. For example, precast double tee slabs and hollow-core pre-stressed slabs provide longer spans that can open up interior spaces to maximize layout functionality (PCI Industry Handbook Committee, 2010)

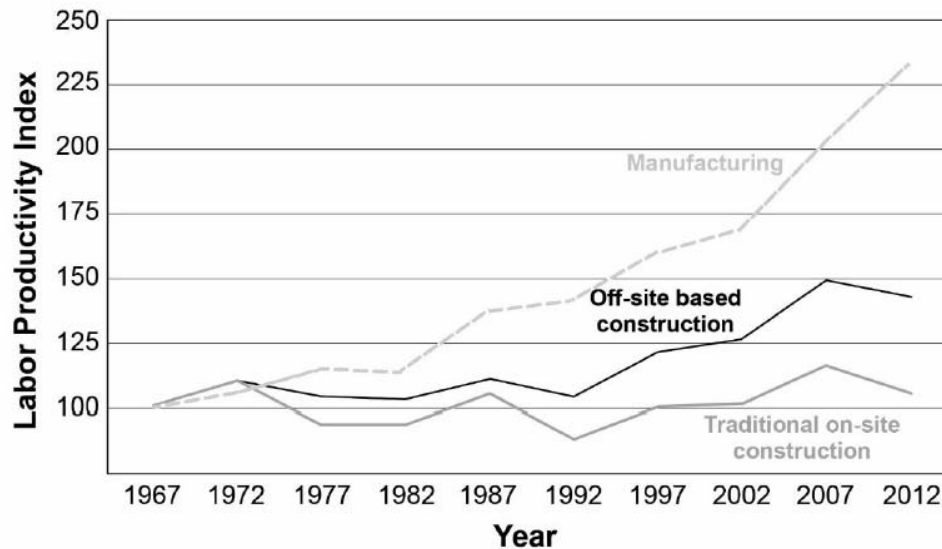


Figure 4 – Labour productivity indices of manufacturing, on-site, and off-site construction in period 1967 – 2015 in the American market, Serdar Durdyev & Syuhaida Ismail (2019)

It is also important to mention some obstacles of pre-fabrication in construction, namely, the rigor in the dimensions and assembly of the elements, the feasibility of transport and storage, and applicability in the work (Pauchet, 2004). Once on site, they are pointed out as barriers the need to use additional connection elements, the rigor and control in the connection of the elements and greater accuracy in the study of the project and details (Couto and Couto, 2007b).

Above all, the main obstacle lies in the production of prefabricated elements, because unlike other industries, one has to wait for an order, a general unitary rule and only after that the production is considered (Resendiz-Vazquez, 2010).

It is also important to mention some obstacles of pre-fabrication in construction, namely, the rigor in the dimensions and assembly of the elements, the feasibility of transport and storage, and applicability in the work (Pauchet, 2004). Once on site, they are pointed out as barriers the need to use additional connection elements, the rigor and control in the connection of the elements and greater accuracy in the study of the project and details (Couto and Couto, 2007b).

Above all, the main obstacle lies in the production of prefabricated elements, because unlike other industries, one has to wait for an order, a general unitary rule and only after that the production is considered (Resendiz-Vazquez, 2010).

2.5. Prefabricated panel construction

2.5.1. Introduction

Panelised systems have been considered by the industry as a viable building prefabricated system due to its flexibility in constructing exterior building façade and interior partitions offsite under varying design requirements. Panelised wall system involves dividing the wall length into panels that are fabricated using either wood or metal studs, with panel widths usually controlled by trucking width that

does not allow special transportation permits (around 10 ft. in the USA). Wall panels can be fabricated for either interior partition or exterior façade, with a height ranging between one to two floors. PWS is classified as a non-volumetric pre-assembly OPP approach, which provides more flexibility in satisfying varying design requirements than the opposite strategic OPP approach, modular building. (Hisham M. Said, Tejaswini Chalasani, Stephanie Logan, 2015)

2.5.2. Definition and history

In recent years, there has been a growing interest in the prefabricated houses industry from both investors and buyers, and it is quickly becoming a new standard in residential building. Prefabricated houses offer a number of attractive advantages compared to the traditional on-site construction method such as substantial reduction of construction time, higher quality control, and potential cost savings. But just as traditional construction has several construction methods, the prefabricated homes category encompasses two main construction methods: panelised and modular. A modular building is a prefabricated building that consists of repeated sections called modules, Lasey et al. (2017). Modularity involves constructing sections away from the building site, then delivering them to the intended site. Installation of the prefabricated sections is completed on site. Prefabricated sections are sometimes placed using a crane. The modules can be placed side-by-side, end-to-end, or stacked, allowing for a variety of configurations and styles. After placement, the modules are joined together using inter-module connections, also known as inter-connections. The inter-connections tie the individual modules together to form the overall building structure. Lasey et al. (2019)

2.5.3. Prefabricated Panels

Prefabricated elements are a particular case of prefabrication, being a form of industrialisation of the walls molded on the site - conventional bricks or masonry wall. Within the wide range of prefabricated panels, it is possible to find elements produced with different materials that respond to the most diverse requirements, such as structural, thermal and acoustic.

These are essentially applied in large industrial buildings, although their applicability in hospitals, residential buildings, offices and practically all types of structures is also verified. The diversity of materials that can constitute a panel and the quality in its manufacturing process, combined with the speed of construction, represent attractive characteristics for the use of this type of elements.

The use of these prefabricated elements is mainly conditioned by the economic viability and transport conditions. The construction with prefabricated panels requires a project with high degree of repetition or the possibility of using elements standardized by the manufacturers, since the production of unique and personalized elements is an obstacle to its viability economic. It is important to note that economic viability can be leveraged by the unavailability of manpower and the planning of the work.

Another fundamental aspect is the possibility of transporting from the production site to the construction site, the weight, the dimensions of the elements and the quality of the access roads, which can also be essential factors. Monty Sutrisna, Jack Goulding, (2019). In addition, the numerous advantages underlying pre-fabrication can sometimes be diminished by the existence of pathologies in the connection between panels and by the monolithic of the structure, requiring a adequate dimensioning of

the connections with the structure and with other panels (Jonatan Francisco Fernandes Salgado, L, 2019).

2.6. Mass customization and platform design

Mass customization was investigated intensively in the manufacturing industry that inspired construction engineering and management research to apply its concepts in mainly home and residential construction. Consumer products and automobile manufacturing industries attempted to overcome the commonality-distinctiveness trade off by developing new design approaches following the principals of mass customization (MC). The main goal of MC is to satisfy the unique needs and design requirements of different customers/projects while still being close the efficiencies of mass production. A well-known strategy to implement MC is the development of product family architecture, which involves the design of generic product architectures that can capture commonalities between different products with design features added or changed between products.

2.7. Smart BIM Object

2.7.1. Introduction

Building information modelling (BIM) is one of the most promising recent developments in the architecture, engineering, and construction (AEC) industry. As the various professionals in the AEC industry become more experienced in the use of BIM, they will utilize an increasing number of intelligent design applications. Numerous BIM applications can be classified over several dimensions. One dimension addresses the integration of BIM tools used in the different stages of the building life cycle, such as code checking, building performance simulations, spatial configuration, and so on. Another addresses the development of smart BIM objects that can be shared, adapted, and reused across different projects within any stage in the life cycle. Designers can insert the smart BIM objects directly into a building model, ranging from low-level building products like doors, windows, to high-level design configuration, such as building core, and façade elements. Overall, this is the anticipated transition to design intelligence, where each of these BIM tools and smart objects encapsulates the domain knowledge to put a building together for better quality and greater efficiency, (Chuck Eastman et al., 2011).

This research explores the concept of smart BIM objects for design intelligence. By enabling geometrical programming in heart of BIM environment, a building element can be designed by a set of objects that carry detailed information about how they are constructed and also capture the relationship with other objects in the building model. Each building elements in a BIM model has typical building rules and relations. Some of those are defined by the user (fabricator or designer) and some can be predicted and defined by a few parameters and constraints. While domain-specific BIM tools combined with parametric rules that are defined by users, allow a building design to be better-informed, there is a lack of research to develop the procedure of utilizing smart BIM objects in transforming their properties automatically and in identifying requirements for object evolution throughout the building life cycle.

2.7.2. Different solution for creating Smart BIM Object in BIM platforms:

Nowadays, in AEC industry, there are several BIM platforms that are progressing fast and introduce new abilities every day to market. Parametric modelling as an important and advanced feature, is implemented in various BIM platforms. BIM services developer are aware of importance of parametric modelling capability in their software and try and compete to improve it continuously, (Chuck Eastman et al., 2011).

Ease of Developing Custom Parametric Objects is a complex capability which can be defined at three different levels:

- 1- Existence and ease-of-use of a sketching tool for defining parametric objects; determining the extent of the system's constraint or rule set (a general constraint rule set should include distance, angle including orthogonally, abutting faces and line tangency rules, "if-then" conditions and general algebraic functions)
- 2- ability to interface a new custom parametric object into an existing parametric class or family, so that an existing object class's behaviour and classification can be applied to the new custom object
- 3- ability to support global parametric object control, using 3D grids or other control parameters that can be used to manage object placement, sizing, and surface properties, as required for the design.

Chuck Eastman et al. in BIM handbook 2011 explained the options to create new parametric objects when they are not available in BIM software:

- 1- Creating an object in another system and importing it into your BIM tool as a reference object, without local editing capabilities
- 2- Laying out the object instance manually using solid modelling geometry, assigning attributes manually, and remembering to update the object details manually as needed
- 3- Defining a new parametric object family that incorporates the appropriate external parameters and design rules to support automatic updating behaviours, but the updates are not related to other object classes
- 4- Defining an extension to an existing parametric object family that has modified shape, behaviour, and parameters; the resulting object(s) fully integrate with the existing base and extended objects
- 5- Defining a new object class that fully integrates and responds to its context.

Considering these capabilities regarding to Parametric objects, here are some main methods have been used for Parametric Modelling. These methods are practical and focused on tools that can be used for the purpose:

- 1- Inbuilt parametric objects such as Wall system family in Autodesk Revit
- 2- Family templates that allow users to define their own parameters and formula.
- 3- Macro scripts in almost all BIM software that provides a simple programming language to define complex parametric objects.
- 4- Visual programming software such as Dynamo and Grasshopper.
- 5- SDK (Software Develop Kit) and API (Application Programming Interface) that can be used for programming with different programming languages like C# and Visual Basic for creating a plugin.

3. METHODOLOGY

3.1. Research scope and limitations

This research work focuses in exploring the ways BIM in general and Artificial Intelligence in a narrowed way, can be leveraged to assist the creating Smart BIM objects as representatives of real building elements that can adjust themselves in different positions according defined rules. These digital Smart Objects can distribute AEC industry to automate processes and take advantages of prefabricated elements. As it is obvious, the issue is very expanded and cannot be handled in this research. Therefore, here the scope has been limited to a practical challenge that a building structural elements fabricator – Fractus – located in Portugal, Marinha Grande has faced: Prefabricated Panels for floors and walls of buildings.

3.2. Main Phases

This research work is divided in 4 phases:

Phase 1) An extensive literature review, focusing on the state of the art of the following topics:

1) Modular Construction in AEC industry, 2) BIM and Artificial Intelligence, 3) Building Information Modelling uses, especially with Artificial Intelligence in context of Modular Construction.

During this stage, the different methods of modular construction with emphasize on Engineered to Order have been reviewed and a framework for producing the selected elements defined by the fabricator has established. Selection the BIM Authoring software, programming interface, and proper API was performed.

Phase 2: Formulation of the BIM-based framework for panelising building parts (walls, floors, and roofs) automatically regarding to rules and constraints defined by panel fabricator. This stage also includes development of an application in line with the selected BIM tools and programming language that aims to automate the process of designing and arranging panels in specific parts of building.

Phase 3: Development of a case study to validate the applicability of the proposed framework and test the application operation.

Phase 4: Formulation of results and conclusions derived from the research work.

A particular limitation of this research work is that it focuses solely on the panelising of the floors of building, considering the special kind of panels produced by Fractus Company.

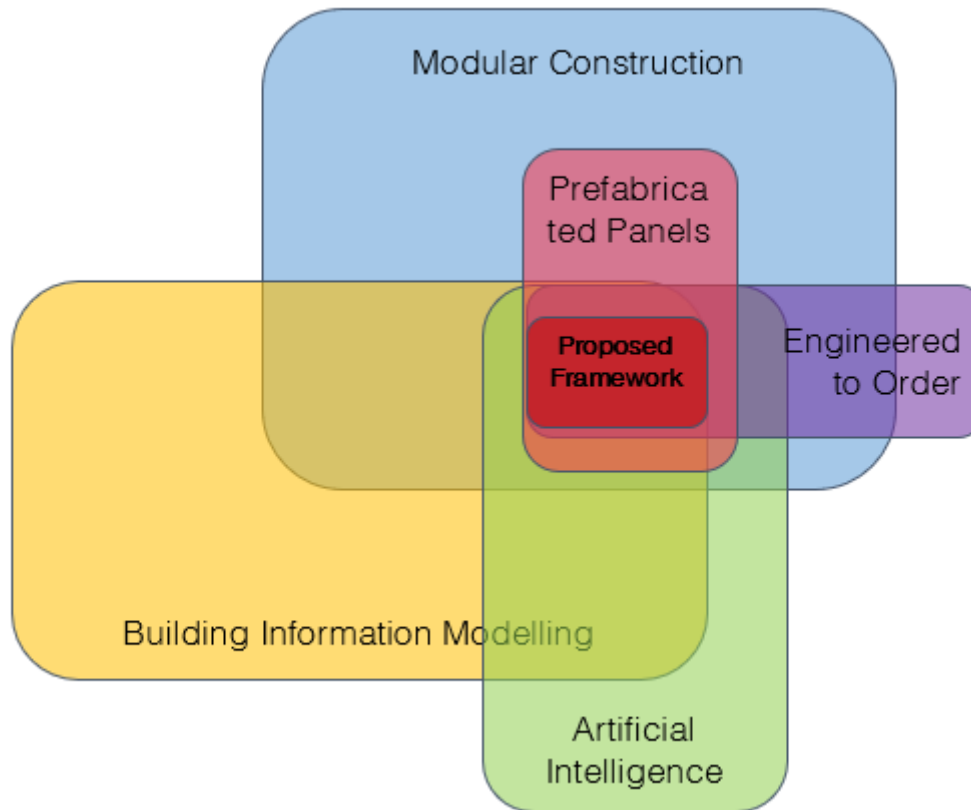


Figure 5 – Proposed framework

3.3. Key assumptions

Before expanding the framework, it is necessary to explain the main assumptions that define the borders of the work area and framework.

- This research has focused on Engineered-To-Order components. It means that prefabricated components should be produced considering the shape of building. It can be a bidirectional relation, both sides (component and final product) influence in each other shape. In ETO process, the manufacturer tries to produce the needed elements completely fitted with design documents of final product. But in some cases, applying some changes in design can increase the productivity and optimisation in ETO elements production processes. In addition, due to some technical limits, it is common that the manufacturer is not able to create a specific element fitted to design.
- As mentioned in the previous chapter, there are two main prefabrication systems in AEC industry: Non-volumetric and Volumetric. This research worked on non-volumetric prefabrication systems. This system has been divided to a couple sub-systems: Component and sub-element, and Panel based elements. As it is obvious, the research focuses on Panel-based component and other systems are excluded.
- The elements are addressed in this research, are limited to floors of buildings. They should have only perpendicular angles. The boundary of elements that have arcs or non-perpendicular angles

are excluded from this research. In fact, complicated shapes need a huge effort and time to handle and the scope of this research is not proper for it.

- The processes are designed for of the special panels produced by the construction company Fractus, located in Portugal. Therefore, the work has been more focused on those panels, although general uses are also considered. It should be mentioned that due to the nature of AEC industry (local elements and local standards) it is hard to develop an algorithm fitted to all kind of panels.

3.4. Identification of the problem

Fractus is a new established company, working on development of a novel kind of prefabricated panels that will be used in walls, roofs and floors of buildings.

There are two main challenges in design and production of these panels:

3.4.1. The arrangement of panels on a wall, floor or a roof:

It is a time-consuming task that never has a unique solution. Each trial can result in a different solution and the designer is never sure whether the solution found is the best or not.

3.4.2. Designing the details of panels

According to Engineered-To-Order approach in the panels production process, each panel can have different properties that respond to its specific situation. Details includes various aspects: shape of frame profile, junctions, thickness of different part of panel, structural form, holes for pipes and conduits, and so on. These details are designed manually and consume a huge amount of time and resources.

A list of issues that have impact on the mentioned challenges is presented in the following.

3.4.3. Issues related to the Design Process:

- There are some rules that should be followed due to some limitation in production process.
- There are rules about the connection between panels - when they are located side by side or when they are close to structural elements. The connections have a large impact on the definition the panels' shape.
- The design process due to production process needs to be enriched with different details. These details include junctions, shape of frame, thickness of metal sheets used on frames, the overall thickness of panels and so on. These details should be designed according to the location of the panels, the loads imposed to them, their neighbourhoods, facilities embedded inside them, and many other factors. These processes are vital and are usually are done manually.
- The panels are considered as structural and also non-structural elements in a building. When they are structural elements such as floors, the loads imposed on them must be calculated and panel should be designed considering that calculations. This process is time consuming and needs to be optimised.

3.4.4. Issues related to Production process:

- The size of panels in one direction was limited between 30 cm to 122 cm, due to limitations in production tools and in the size of polyester sheets were used to cover the panels.
- If the size or length of a panel is more than 350 cm, the structure of the panel must be strengthened and some more metal elements should be added in the middle of the panel.
- Using panels with similar size is preferred because the production time will be reduced and the probability of error will decrease. So, dividing a building element into panels should be done in a way to reduce the variety of panel size as much as possible.

3.4.5. Which challenges were selected to deal in the research and why?

Choosing the practical challenges to be solved in this dissertation are dependent to some main abilities and limitations:

- Scope and domain of author's knowledges:

Some of the mentioned problems, need structural knowledge and experiences and for an architect as the author, it may be unsuccessful.

- The time that is assigned for dissertation:

Due to the dissertation period of 5 months, it should be considered to submit the result in due time.

- The domain that results can be applied to:

Often between industry and education (university) a famous gap appears: General approach or Narrow-downed approach to solve a problem? The industry tends to receive an exact solution for a real and technical problem, while scientific researches want to solve a more general problem. In this research it is important to balance this situation and lead it such that addition solving a general problem, a specific technical problem of Fractus, be solved.

Table 2 presents a list of challenges that are evaluated according to the mentioned criteria and explains the selection of challenges.

Challenge	Needed knowledge	Needed time	General use	Specific solution for Fractus
The arrangement of panels on a wall, floor or a roof	Geometry	4 months	High	Yes
	Architecture			
	Programming			
Designing the details of panels	Geometry	10 months	Low	Yes
	Structure calculation			
	Detailing Programming			

Table 2 – Evaluating the possibility of solving the challenges

The challenge chosen for further development is the process of automatic panel arrangement on a surface of a given building. This case is fitted by the time limitation of the research. in fact the selected challenge

has kind of balance in one side, it is a general solution that other designers and fabricators can adapt it on their process and in another side it can be a practical solution for Fractus as industrial representative.

3.5. Steps of the solution

The development of a process that can automatically arrange the distribution of panels in a floor includes five main sections, presented in the following.

Analysing the challenge: In this part we tried to analyse the problem from a general perspective and to review current existing solutions.

Defining and evaluating the tools available: This section is focused on tools such as BIM authoring software, Visual Programming tools, Programming Languages, and available algorithms for Artificial Intelligence. The purpose of this section is to find the best tools to develop our solution.

Proposing the possible solutions: In this section, some different but possible solutions are explained. The solutions are described briefly through flowcharts, diagrams, and even programming scripts.

Evaluating the solutions and selecting the proper one: It is important to evaluate and select the best solution. The evaluation criteria is defined carefully and the evaluation process relies on it.

Implementing the selected solution in practice: This section is explained in detail, in a separated chapter.

3.6. Analysing the Challenge:

The most usual way to manufacture walls, floors, and roofs, is to construct with prefabricated panels. These elements cannot be prefabricated in one piece due to the limitation in transferring and erecting huge building elements. These parts of buildings in term of size, usually are oversized for transportation. So, the challenge is to cover a floor of a building with prefabricated panels. But these panels are produced as an ETO element not mass-produced element. It means that the designing process is not a linear process. There are two factors to be solved: arrangement of panels and the size of the breadth and length of panels. As a sequence, there are more than a single solution. For each arrangement alternative, there is a set of dimensions for panels. So the main challenge is not about dividing the element by panels but to divide it by panels and to find the size of panels in the most optimized way. In fact, the problem is about *optimization*.

The 2D boundary of a given element will be covered by panels according to rules and restrictions of panels production. But it should be examined from optimization perspective. Can it be improved? Or the solution is the most optimized. Here the challenge has been described by a diagram.

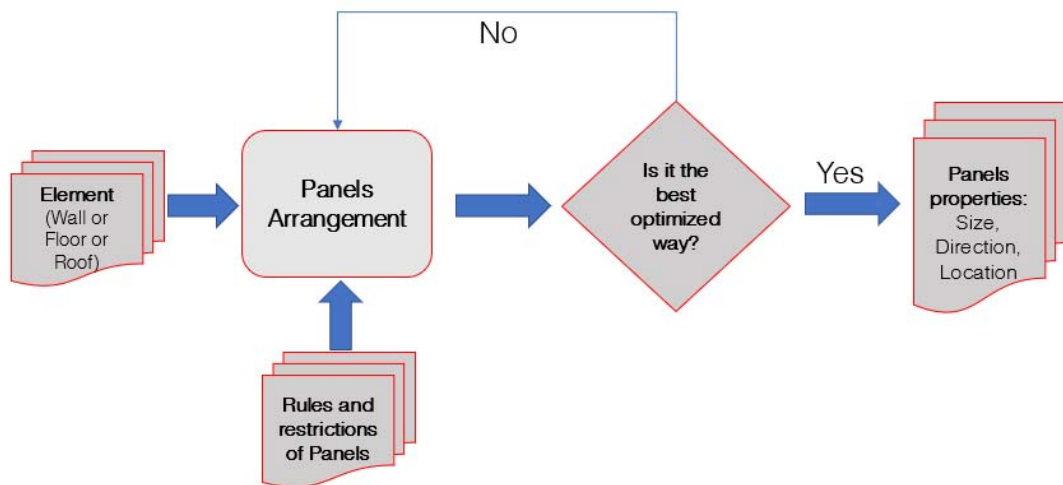


Figure 6 – General work-flow of analysing the problem

3.6.1. Assumptions

To cover a partitioning 2D element of building (wall, floor, and roof) with prefabricated panels Panels are produced through Engineered To Order (ETO) approach.

3.6.2. Rules and restrictions

The width or length of panels cannot be less than 30 cm.
 The length of panels cannot be more than 700 cm.
 The width of panels cannot be larger than 122 cm.

3.6.3. Optimization goals

The panels should have same size as much as possible.
 The direction of locating panels is free and they can be located in any direction fits better.

3.7. Define and evaluate the tools available:

It is necessary to find the tools available for solving the problem. Each stage of solving work-flow needs a specific digital environment to get the needed information, to do a calculation process, and to submit the result. Here, each stage and related digital environment is explained.

3.7.1. Main stages

- Collecting needed data from a given building element;
- In the first step, the geometrical data includes 2D shape dimensions of the element (regardless of its thickness);
- To perform the arrangement process and to optimize the result. First we should get data related to rules and restrictions that are defined by user (panel fabricator). This data is non-graphical and should be adjustable. After that, arrangement and optimization process must be performed.

- To submit the result and to transfer it to the element digital representative. In this stage, the building element should be covered by panels.

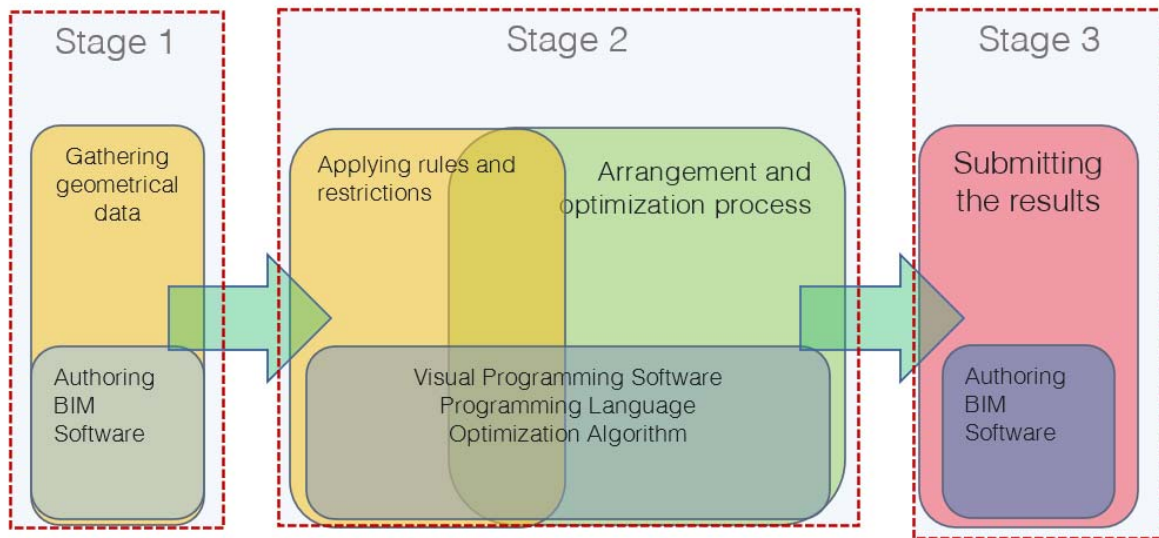


Figure 7 – Domain of solution and related tools

3.7.2. Selecting software

In this research, two alternative BIM authoring software were considered: Tekla Structures and Autodesk Revit. The given building element should be extracted from models, as provided by one of these software.

As it is shown in Figure 8, Autodesk Revit is chosen as the BIM authoring tool for stages 1 and 3. This software transfers geometrical data of given elements to stage 2 and applies the result back to that element. Stage 2 can be done by Python scripts in Dynamo environment. Dynamo is a visual programming application that has recently been embedded in Revit. The work flow between Revit and Dynamo is bidirectional and completely automatic.

An alternative, Tekla structures is also chosen as the BIM authoring software. Tekla does not have a visual programming application but one of the most famous and popular visual programming applications - Grasshopper - can be used by Tekla. There is a plugin for Grasshopper called Tekla Live Link that connects models from Tekla to Grasshopper bidirectionally and returns back the results to it. It must be mentioned that Grasshopper itself is a plugin that must be installed on Rhinoceros.

In fact, the core of calculation is handled by Python or C# or any other proper programming language and the other mentioned applications only are used as a platform, digital environment or Visual application. So, the alternative paths are almost equal. The programming language may use the Application Programming Interface (API), especially in geometrical areas. Each individual software has its own API and specific instruction to use. Fortunately, now a days almost all 3D software uses a couple of globally accepted standard frameworks in geometrical data structure. For this reason, we can almost exchange all geometrical data between all 2D and 3D modelling applications. So in this case, APIs of

different BIM authoring software, in geometrical area, are almost similar or at least have the same structure that do not make a big difference in overall process of finding a solution.

In the end, Tekla Structures was selected as BIM authoring tool because the case study was modelled in Tekla and Fractus company uses that software.

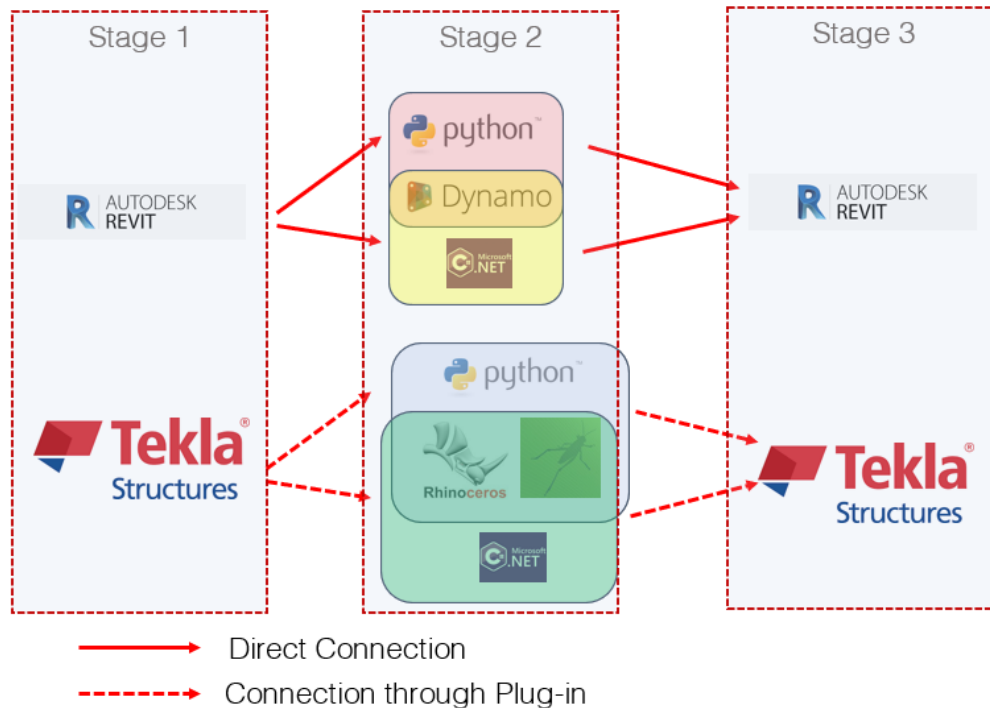


Figure 8 – Three stages of solution and selected tools and software

3.8. Optimization

Some different but possible solutions have been explained. The solutions are described briefly through flowcharts, diagrams, and even programming scripts. Two main concepts for solving the problem were created and developed:

- **Optimization by generative design algorithm:** It is called “**Blind Optimization**” due to its structure.
- **Creating an algorithm by exhaustive search method:** It is called “**Rational Optimization**”, considering the solution structure.

3.9. Blind Optimization

The main idea is to divide the surface of a building element with an arrangement of panels. Then they should be moved in both directions (X and Y) by the size of one panel to generate new alternatives for arrangement. Moving the arrangement in both direction (X and Y) in distance of breadth and length of panel, generates a new arrangement. If the movement goes further length and breadth, the generated arrangement is similar to ones generated before. See Figure 9. after collecting all results, all must be examined and the most optimised one should be selected.

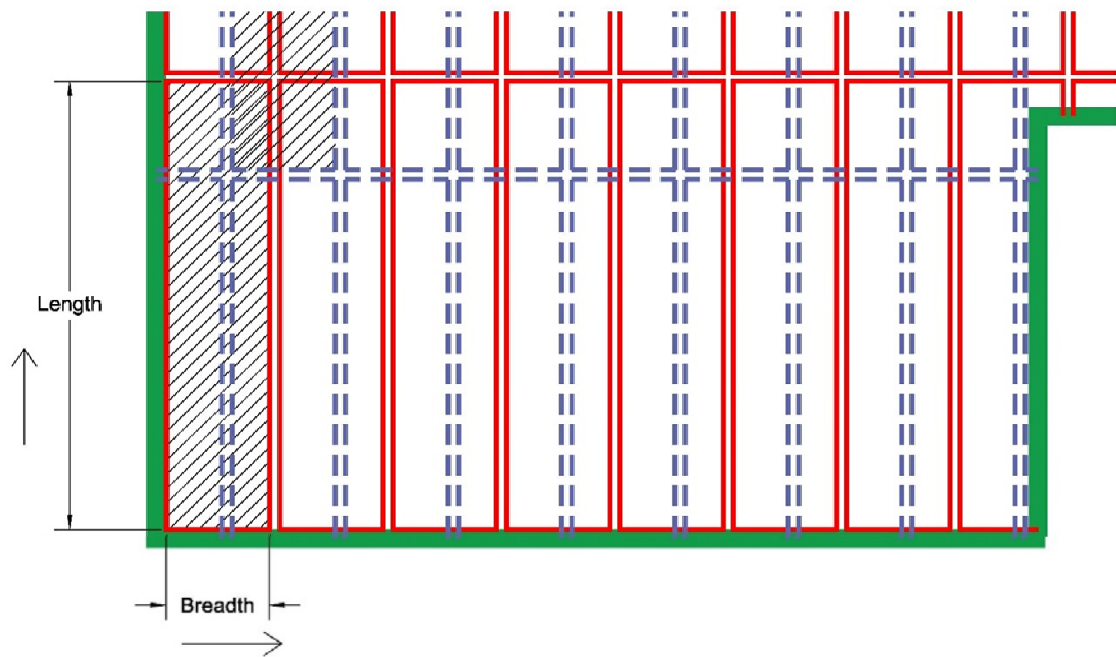


Figure 9 – Moving arrangement in X and Y directions by length and breadth of a panel to generate new arrangements

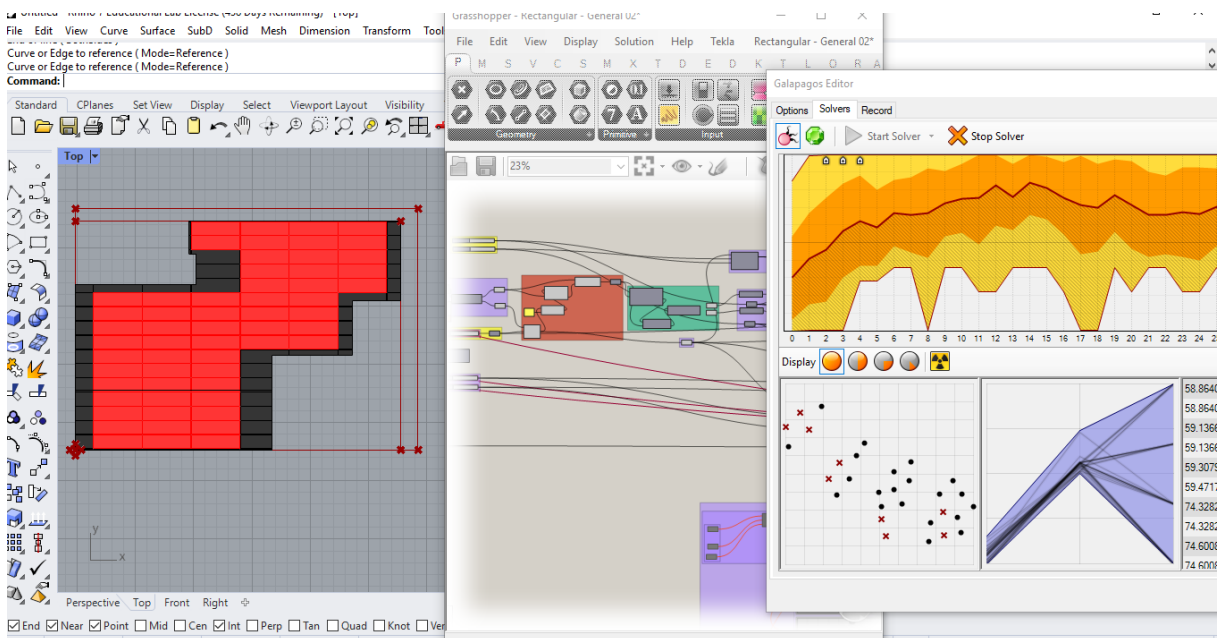


Figure 10 – An overview of blind optimization in Rhino and Grasshopper

The work flow of "Blind optimization" is described in Figure 11. The main steps include:

- Getting Data - geometry and rules
- Performing the panel arrangement
- Creating different genes of arrangement
- Evaluating results against the goals
- Selecting the optimal result

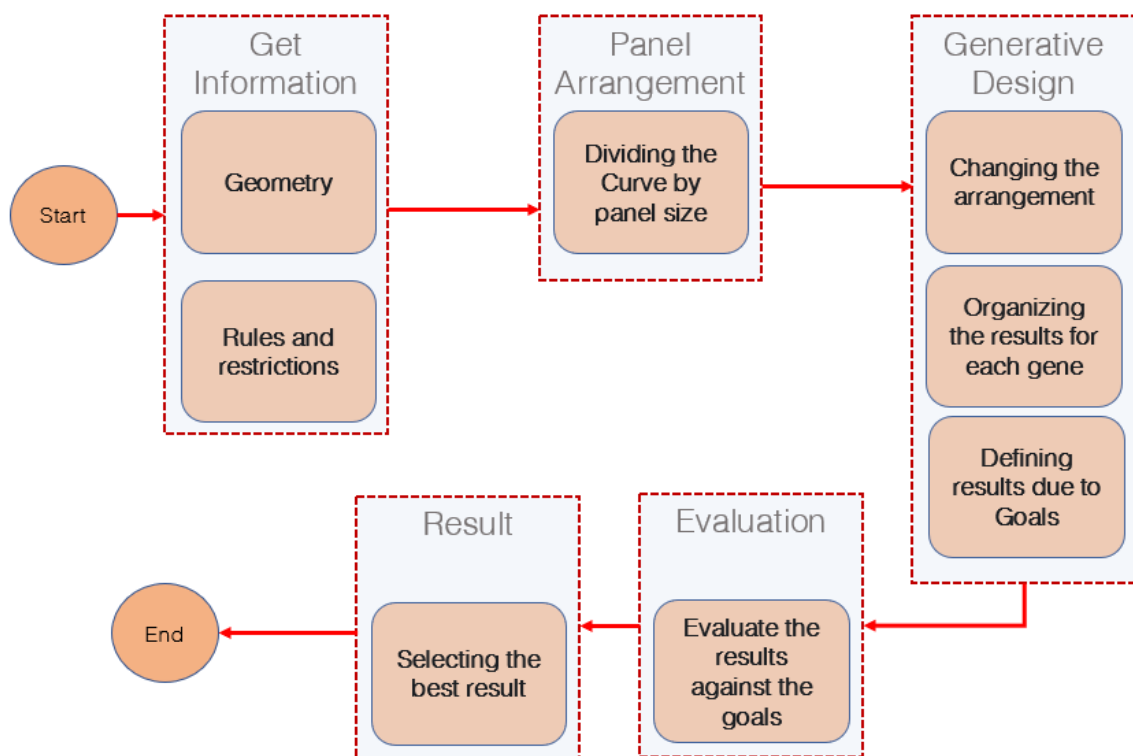


Figure 11 – General work-flow of Blind optimization

The structure of this solution was developed in Grasshopper. As shown in Figure 12, all the process is done by Grasshopper nodes.

Steps in a brief explanation include:

- Getting 2D boundary of element (floor) as an input.
- Surrounding the boundary by a bounding box. Bounding box is the smallest possible rectangle that can surround the boundary.
- Finding the longest and shortest edges and the corners of bounding box.
- Calculating the number of panels in each direction according the given panel size.

- Creating the arrangement and calculating the number of regular and irregular panels that are placed in the original curve.
- Defining and extracting the properties of arrangement includes:
 - The rate between the number of regular panels and the number of irregular panels.
 - The rate between the overall area of regular panels and the area of irregular panels.
 - The total number of panels (regular and irregular).
- Creating the possibility of moving arrangement in two directions by size of one panel.
- Making possible to start arrangement from each corner of bounding box.
- Using the optimization node and define the adjustable parameters.
- Creating a proper number of genes according to the parameters and gathering the results.
- Finding the best result:
 - Maximum rate between number of regular panels and number of irregular panels or,
 - Maximum rate between total area of regular panels and total area of irregular panels, or
 - Minimum number of total panels.

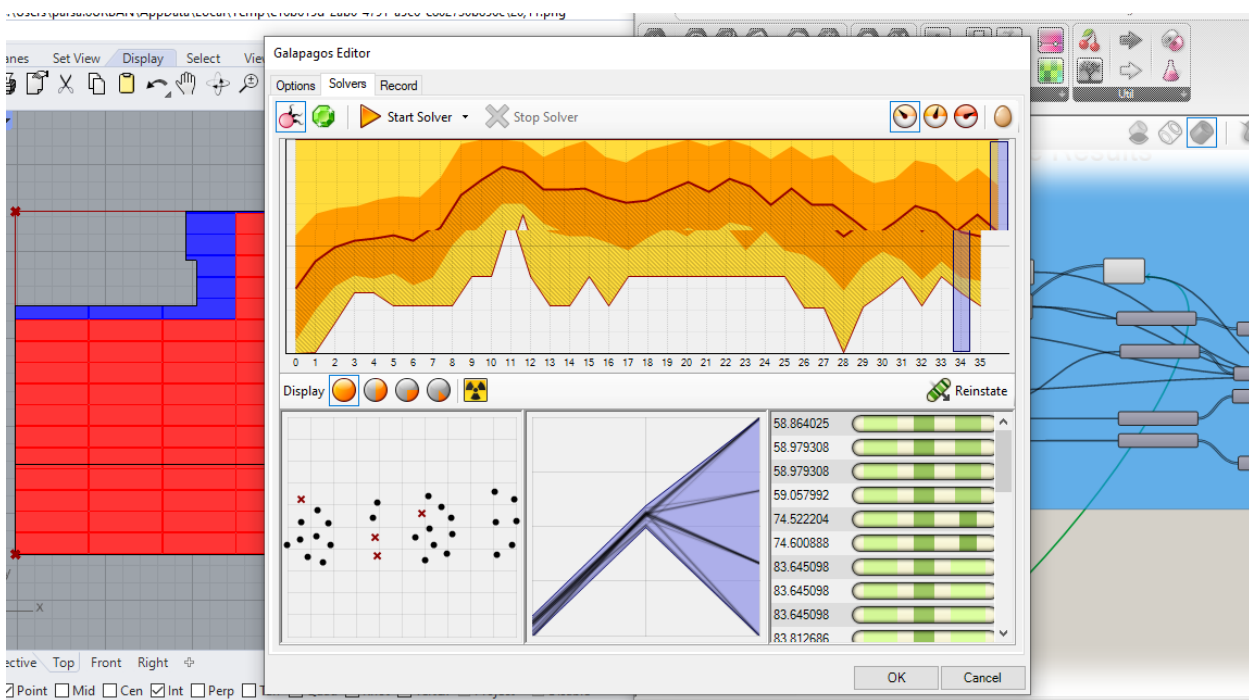


Figure 12 – Galapagos plug in and generative design

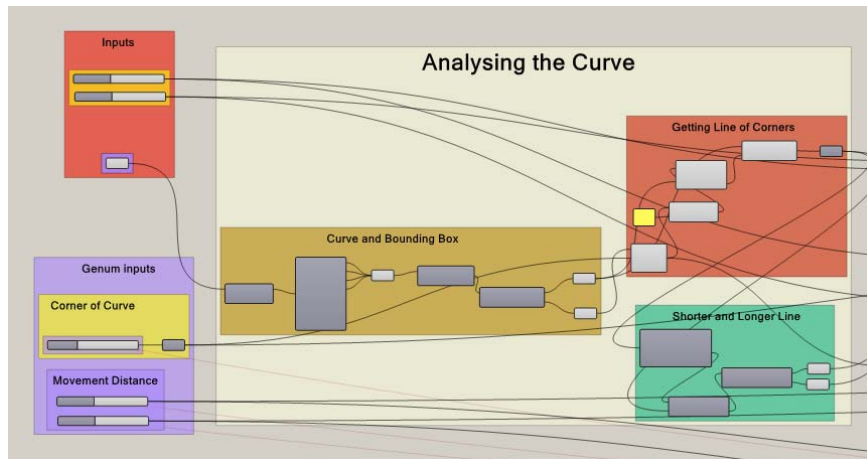


Figure 13 – Grasshopper script, Analysing part (for detailed image, refer to appendix 2)

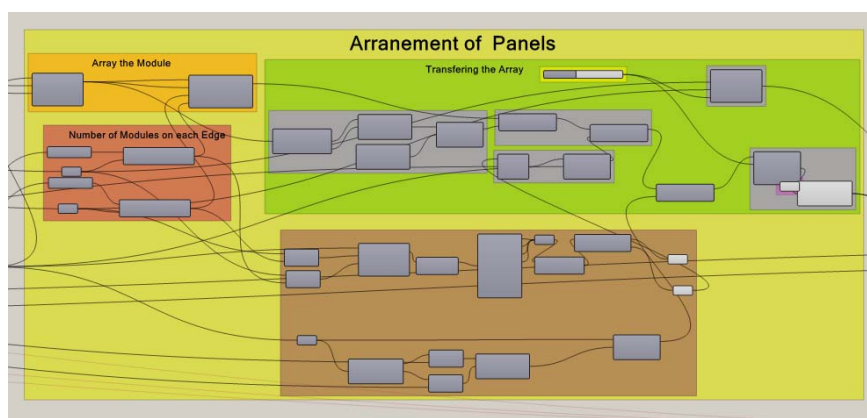


Figure 14 – Grasshopper script, Panelling part (for detailed image, refer to appendix 2)

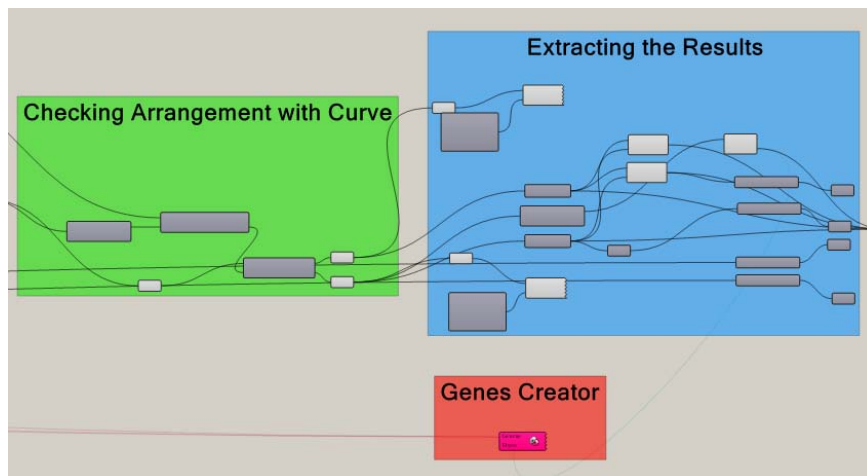


Figure 15 – Grasshopper script, Optimization part (for detailed image, refer to appendix 2)

3.9.1. Cons and pros:

The “Blind optimization” presents some vital disadvantages:

- It is a time-consuming approach: for an average number of genes creation, with a normal Core i7 processor, it needs more than 30 minutes to produce results.
- It is not proper for Engineered To Order panels because it assumes that all panels have the same size.
- All the panels are oriented in one direction.

The “Blind optimization” also presents advantages:

- The process is easy and fast.
- It can analyse all kind of 2D shapes.

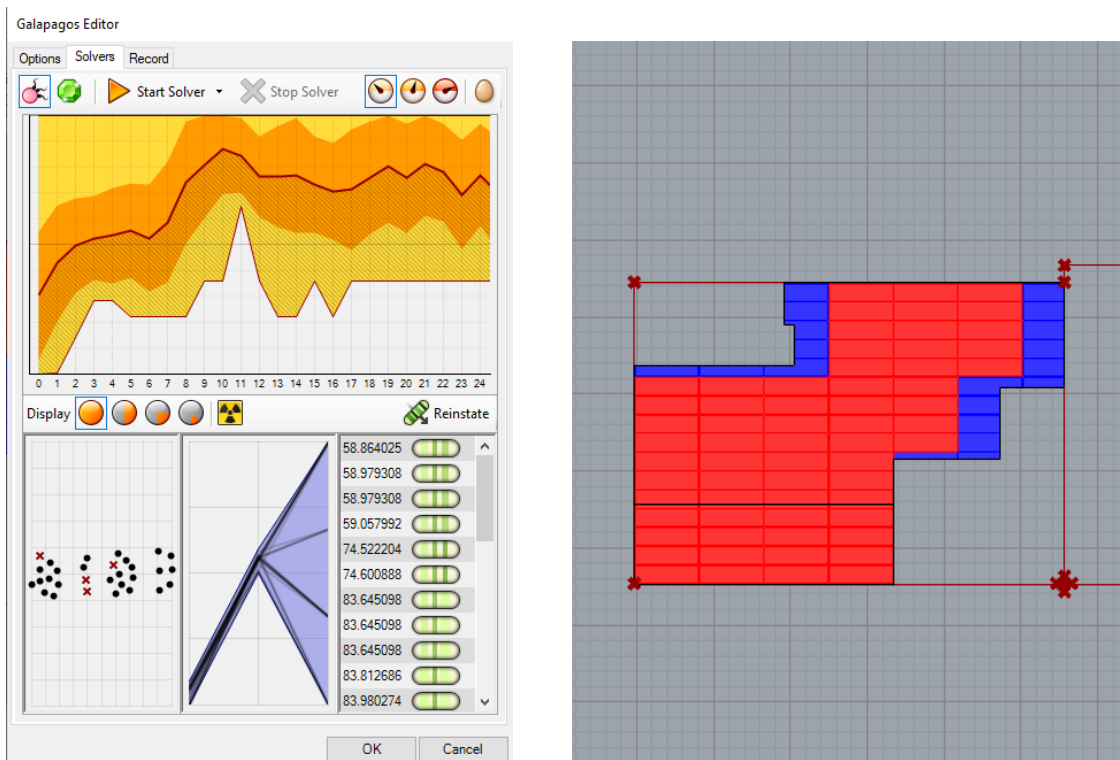


Figure 16 – A sample result of Blind Optimisation by grasshopper and Galapagos

3.10. Rational Optimization

The main concept is to perform the arrangement process in a linear way. It means that the solution is created based on a human logic of solving the problem. As it was explained in the first alternative, that solution ignores some key factors: a) the possibility to locate the panels in different directions and b) the possibility to use panels with different sizes when it needed. It is not claimed that this solution returns the best result. In fact it cannot be proved. This solution as it is shown following, has been extracted

from human method. A human starts to simplify a boundary by dividing it to some rectangles and tries to arrange panels in those rectangles.

The first step is to recognize the boundary and openings located in it and to find the biggest rectangle that is surrounded in its boundary and does not have intersection with voids. Then, it necessary to find the next biggest rectangle and repeat this function until the original boundary is completely divided into different rectangles.

The work flow of "Rational optimization" is described in Figure 17. The main steps include:

- Getting Data
- Organizing the boundary and openings
- Finding the biggest rectangle surrounded in boundary that does not intersect with voids
- Subtracting the rectangle from boundary and defining the rest of the area as a new boundary
- Finding the next biggest rectangle and repeat this cycle
- Collecting the rectangles found and arranging panels in each one.

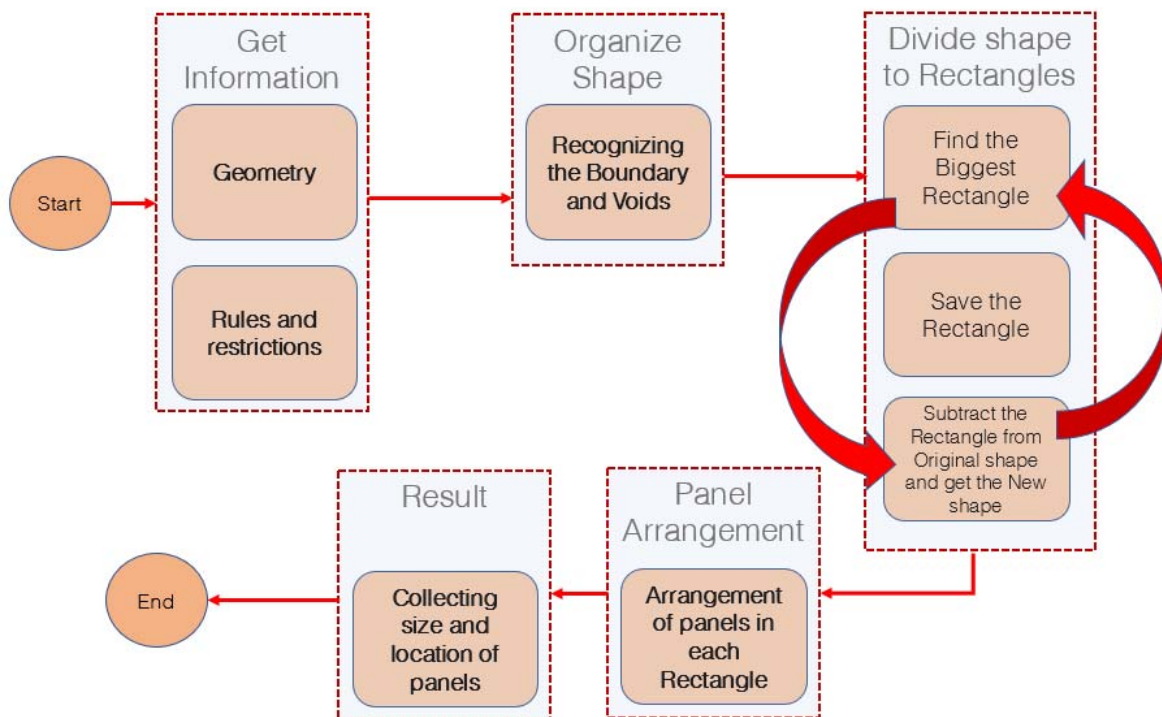


Figure 17 – General work-flow of Rational Optimization solution

The structure of this solution was developed in Python and using Grasshopper API. Implementing this solution by Grasshopper alone is not easy because the algorithm needs to repeat itself frequently. As it is shown in Figure 17, all processes were done in Python, within the Grasshopper environment.

Steps in a brief explanation include:

- Extracting the geometrical data from BIM elements in Tekla structures
- Getting the rules defined by user:
 - The minimum size for opening to be considered as void
 - The minimum and maximum possible size for panels
- Finding and organizing the boundary curve and voids and small holes (columns)
- Finding the biggest rectangle that:
 - Is surrounded in the boundary curve
 - Does not intersect with voids
- Subtracting the rectangle from boundary and getting the resulting shape
- Repeating the cycle of finding the biggest rectangle until the rest shape is a rectangle itself.
- Dividing each rectangle by the most fitted panel size and the most proper direction
- Exporting the shape of panels to Tekla Structures for creating the panel on model element

3.10.1. Cons and pros:

The “Rational optimization” also presents advantages:

- The process is fast and takes between 1 to 4 minutes;
- It is fitted with Engineered-To-Order manufacturing processes;
- Changing the direction of panels and size of them are considered,

and also disadvantages:

- Changing and improving the algorithm needs programming skills.

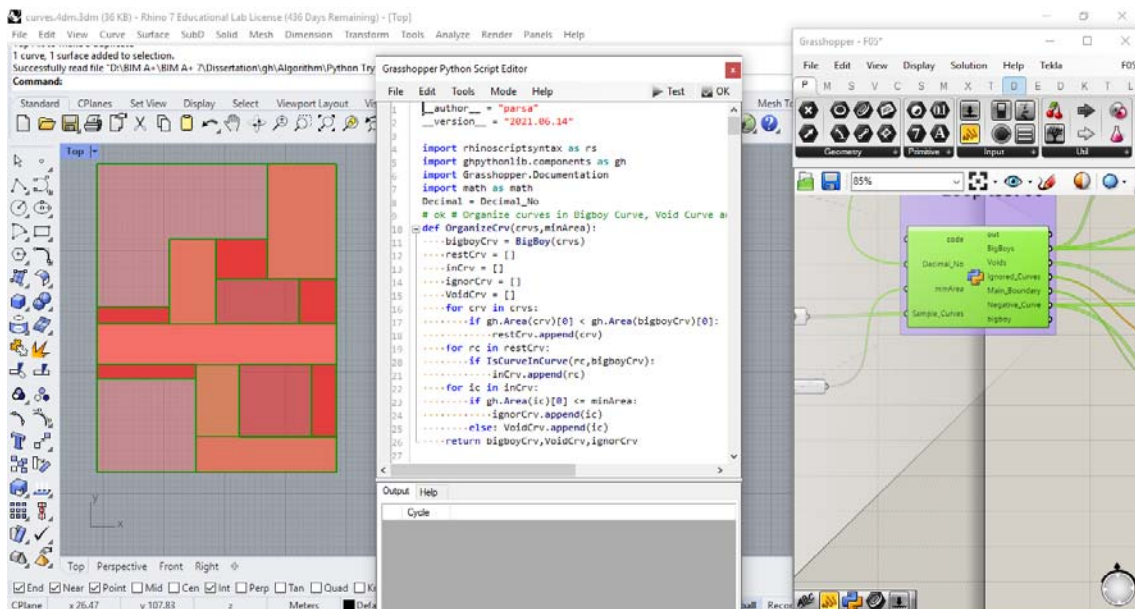


Figure 18 – A sample result of Rational Optimisation by Python in Grasshopper

(for detailed image, refer to appendix 2)

3.11. Conclusion

Usually, there are several solutions for a problem and each solution has its own specifications. Here we tried to compare both of solutions in 9 criteria. It must be mentioned that each solution has its own attributes and characteristics. These attributes and characteristics may be advantages or disadvantages, depends our goals. Here, for comparing the two alternatives, the criteria are extracted from our goals, and each solution has gained a score against each criteria. At the end, we have selected the solution that has gained the highest total score.

Column	Subject		Evaluation Grade		Requirements
	Domain	Evaluation criteria	Low = 0 Medium = 1 High = 2		
			Alternative 1: Blind Optimization	Alternative 2: Rational Optimization	
1	Hardware and resources	Processor usage	0	2	Less processor use
2		Time consuming	0	2	Short time
3	Manufacturing approach	Fitted to Mass production	2	1	Engineered to Order production process
4		Fitted to ETO production	1	2	
5	Geometry covering	Shape with arc or non-perpendicular angles	1	0	Shape with perpendicular angles
6		Shape with perpendicular angles	2	2	
7	Development	Possibility of developing the solution	1	2	Improve the ability of algorithm is needed
8		Possibility to implement in other BIM platforms	1	2	Implementing in different BIM platform is required
9		Maintenance and modification	1	2	Easy Maintaining and modifying
10	Total Grade		9	15	

Table 3 – comparison two solution (Blind optimization and Rational optimization)

As it is obvious in Table 3, the “Rational optimization” approach has an upper grade. Therefore, it was selected for continuing the research.

4. RATIONAL OPTIMISATION SOLUTION

Development of selected solution

The selected solution has been performed by using programming language, with software API in a visual programming software.

4.1. Main Parts:

The process of solution has been divided to five main parts that include:

- The Input Algorithm: collects needed data from BIM model.
- The Division algorithm: divides the given area into rectangles.
- The Joining algorithm: finds the rectangles that are not able to be covered with panels (considering the production rules of the factory) and joins them with neighbour rectangles.
- The Panelising algorithm: arranges the panels in each rectangle.
- The Output algorithm: sends the panel arrangement to the BIM model.

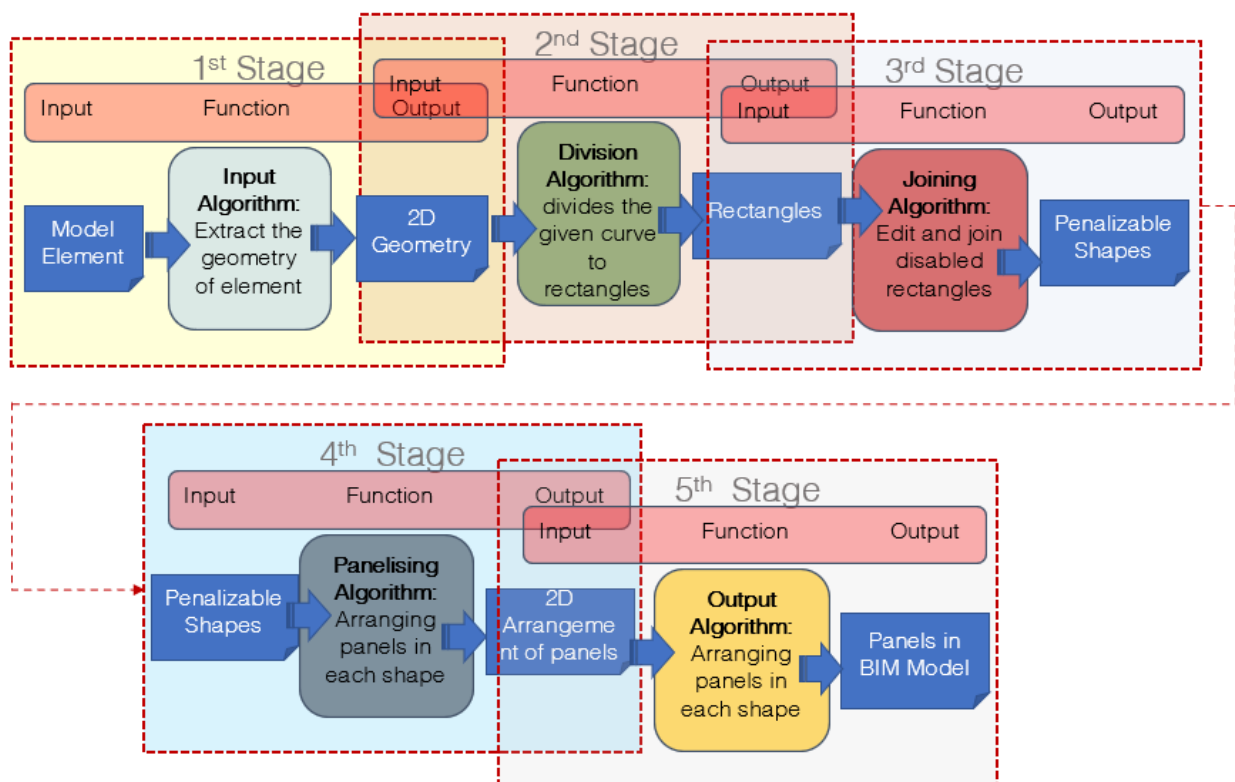


Figure 19 – Main stages of Rational optimisation solution

4.1.1. Input Algorithm:

This algorithm has been developed to collect, refine and extract geometrical data from the element modelled in Tekla Structures.

4.1.2. Division Algorithm:

Main concept: the Division algorithm is the most creative part of the solution. This algorithm is inspired by the way human think to solve the problem. The method used by engineers in Fractus company to divide a floor, roof or wall and panelising it, is implemented in the algorithm. As can be observed in Figure 20, a human can find at least four alternatives for dividing the floor area. So, which division is better? Often, the best alternative is the largest rectangle. Here, we evaluate the largest rectangle by its area.

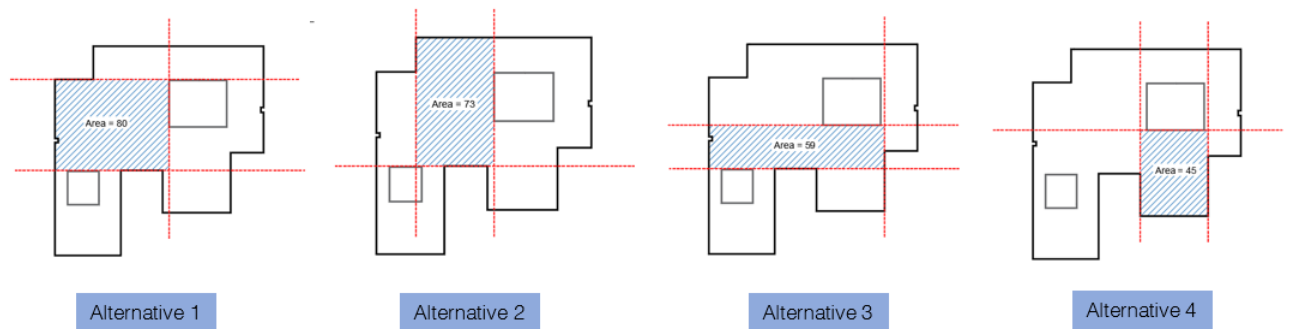


Figure 20 – Alternatives of solution carried out by human

The solution with the rectangle that presents the largest area (alternative 1) will be selected. Based on this alternative, the search process will be done repeatedly. The work-flow of a human addressing this problem is mapped in Figure 21.

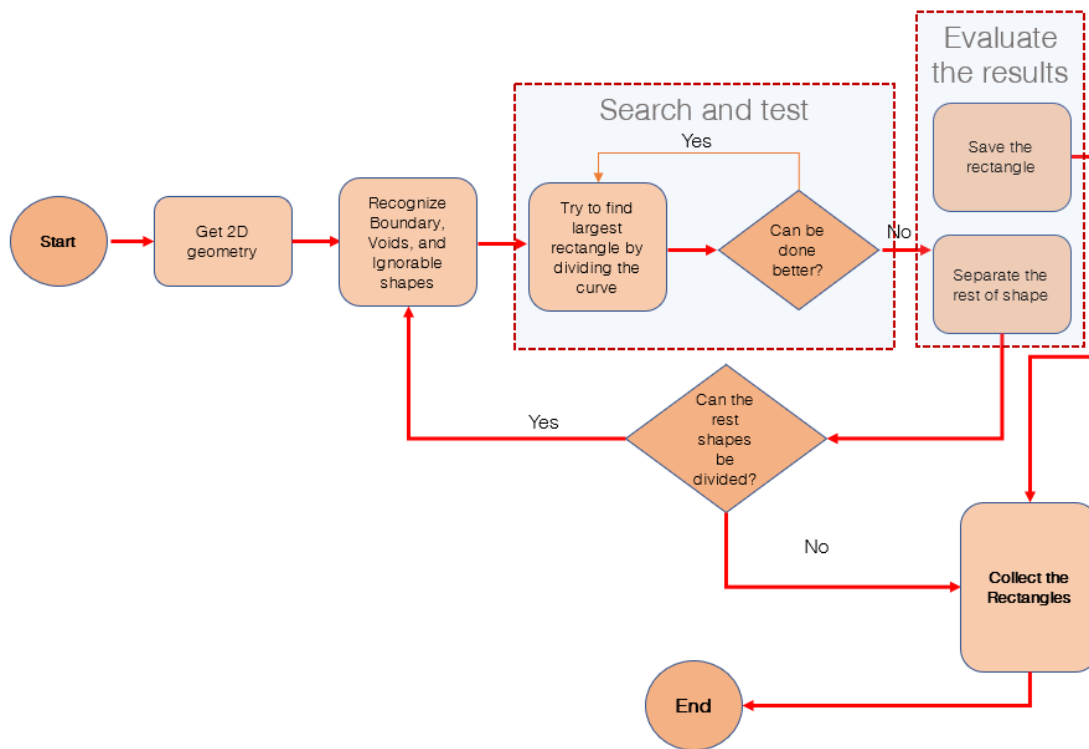


Figure 21 – General work flow for the rational optimisation solution

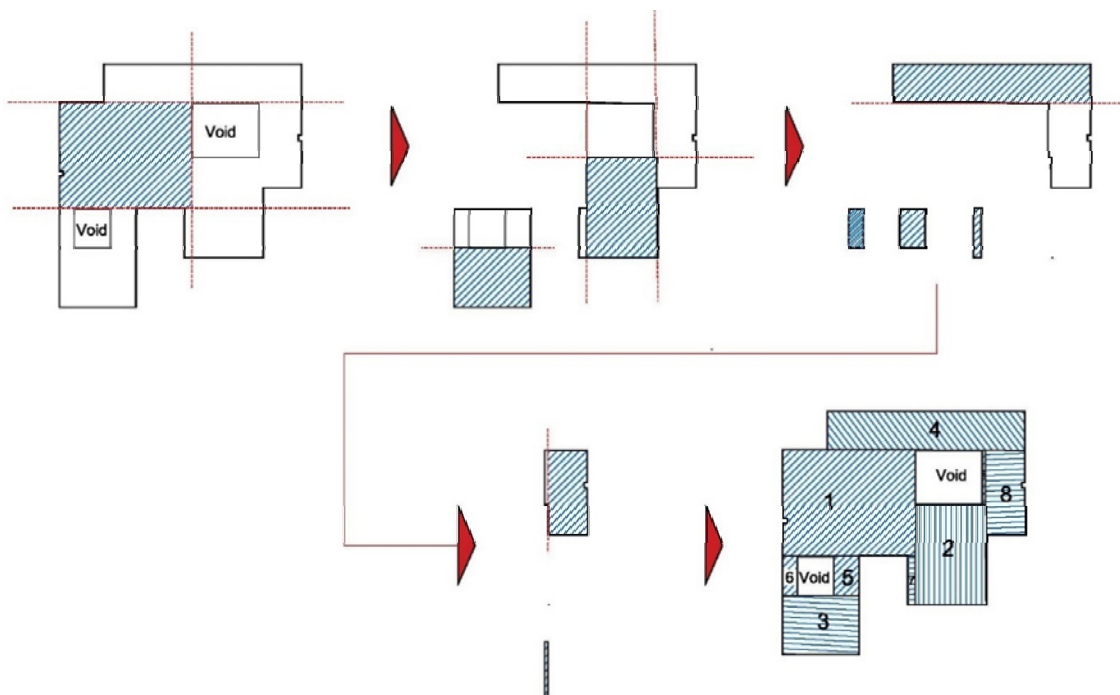


Figure 22 – A geometrical sample of different steps of Rational optimization solution

4.1.3. From human method to computer method

Translating the method of solving this problem by a human into a digital method which can be performed by a computer processor, presents several difficulties. Computer science concepts, as Heuristic technic and Brute-force search are used to solve this problem.

4.1.3.1. Heuristic technic

“In mathematical optimization and computer science, heuristic (from Greek εὕρισκω "I find, discover") is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This is achieved by trading optimality, completeness, accuracy, or precision for speed. In a way, it can be considered a shortcut.” ([https://en.wikipedia.org/wiki/Heuristic_\(computer_science\)](https://en.wikipedia.org/wiki/Heuristic_(computer_science)))

Why in our human method, the largest rectangle should be found? Is there another way more efficient? We do not know!

Dividing a given shape into some rectangles is a shortcut that seems logical but we cannot say if it is the most efficient way. The objective of a heuristic approach is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand. This solution may not be the best of all the possible solutions to this problem, or it may simply be an approximation to the exact solution. However, it is still a valuable solution, because finding it does not require a prohibitively long time.

4.1.3.2. Brute-force search

As it is shown in Figure 21, there is a challenging part in the algorithm: finding the largest rectangle in the boundary. A human, due to his/her limitation in considering and calculating various alternative, might choose a wrong option or even not collect the best answer in the candidate pool. How The human brain solves such challenges is not completely clear. To adapt this process to a digital algorithm, we used Exhaustive search method. In computer science, Exhaustive search or Brute-force search is a very general problem-solving technique and algorithmic paradigm that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.” (Mark Burnett, 2007)

Brute-force search technic is here used to find all possible rectangles that fit to the statements: 1) surrounded in the boundary and 2) does not intersect with voids. This method is not appropriate for human’s brain calculation because it may take hours our days to survey all possible candidates. This method also has some limitation for computer processors.

“While a brute-force search is simple to implement and will always find a solution, if it exists, implementation costs are proportional to the number of candidate solutions – which in many practical problems tends to grow very quickly as the size of the problem increases.” (Wikipedia, exhaustive search https://en.wikipedia.org/wiki/Brute-force_search)

Therefore, brute-force search is typically used when the problem size is limited, or when there are problem-specific heuristics that can be used to reduce the set of candidate solutions to a manageable

size. The method is also used when the simplicity of implementation is more important than speed. In this case, if the given shape is too difficult, the process time increases strongly and the program may collapse or take a long time to proceed. In typical buildings, elements like walls, floors or roofs do not have too complex geometries and often this technic can perform the calculation easily.

By using these two techniques, human solving process can be transferred to a digital process done by computers. Figure 23 shows the digital work-flow.

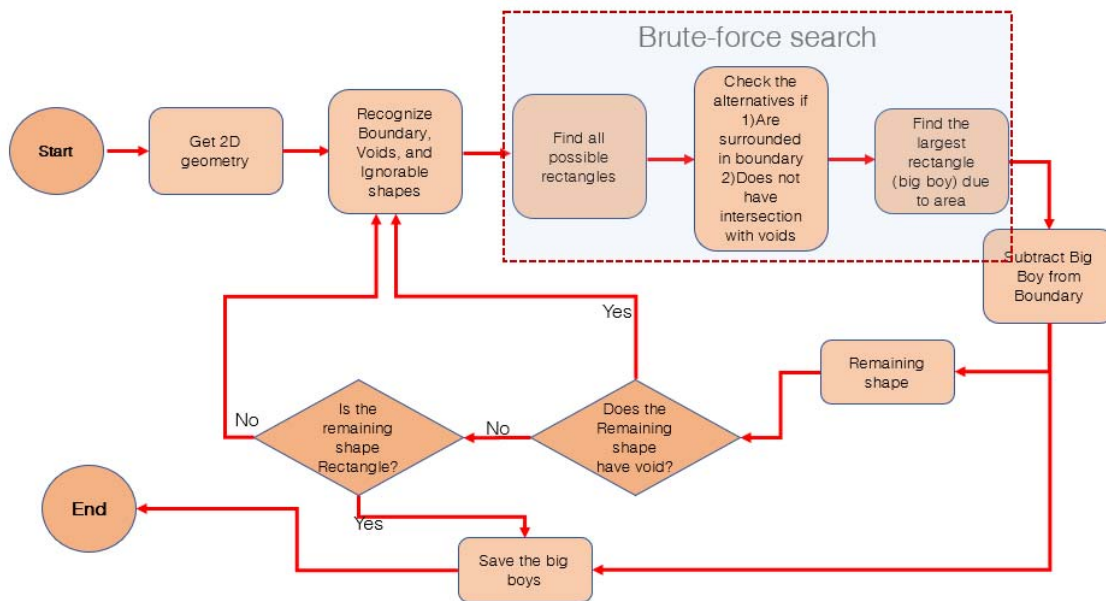


Figure 23 – Digital work-flow of rational optimisation extracted from Human work-flow

4.1.4. Structure of the algorithm

In this part, Division algorithm (stage 2 – according to Figure 19) is explained in detail. To see the Python codes, please refer to Appendix 1.

4.1.4.1. Main concept

As it is shown in Figures 24, 25, and 26 (Process flow), the algorithm should perform all the defined steps. To do those, various functions were defined using Cartesian geometry, point and line, and some of geometrical concepts such as Union, Difference, and Intersect. Three types of Functions were defined in the algorithm:

- Checker functions: they test a condition and always return True or False. These functions help the algorithm to decide and find out which statement is true in a particular condition.
- Recogniser functions: they recognise, organize and modify curves. These functions help the algorithm to recognize the shape properties, the relationship between given shapes and to reorganize them for other functions.

- Executive functions: they perform a work on inputs and extract or produce outputs. These functions usually do the main process and produce new shapes, data or other outputs.

Here we have defined some new data structures that are used as input and output in different functions. They are described in Table 4 and include:

- Curve
- Group
- Branch
- Void
- Boundary
- Ignorable Curve
- BigBoy

Some items from Grasshopper API were used to generate the algorithm. We tried to use as few API items as possible. The way these items are defined is aligned with specific goals of software developers and, in some cases, achieving desired goals with them is not completely possible.

These items include:

- Area
- RegionDifference
- RegionIntersection
- RegionUnion
- Decostruct
- Explode

Table 4 – Data types and function of Division Algorithm

No	Type	Name	Input	Output	Description
1	Data Type	Curve	-	-	Any kind of shape with only perpendicular angles
2		Group	-	-	A list of curves
3		Branch	-	-	An organised group of curves consist of a boundary, some voids and some ignorable curves surrounded in boundary
4		Void	-	-	A curve that is surrounded in a boundary and represent an opening in a building element such as a window in a wall
5		Boundary	-	-	The outer edge of a building element like a floor, as a 2D closed curve
6		Ignorable curve	-	-	A curve surrounded in a boundary but the area is less than the Minimum Area defined by user and algorithm ignore them in calculation process
7		BigBoy	-	-	The largest rectangle found in a Branch
8	Checker function	AreSimilar	-Two Curves	-Boolean	Check two curves if they are completely similar in shape and location (duplicated curve)
9		IsCurveInCurve	-Two Curves	-Boolean	Check if the first Curve is surrounded in the second Curve or not
10		HasIntersect	-Two curves	-Boolean	check if two curves have intersection or not
11		HasCommonEdge	-Two curves	-Boolean	Check if two curves have a common edge (touch each other) or not
12		IsInandAligned	-Two curves	-Boolean -Boolean	Check two conditions: one of curves is surrounded in the other curve or not, and they have a common edge or not
13		IsRectangle	-A curve	-Boolean	heck if a curve is a rectangle or not
14		IsBoundaryBigboy	-A Branch	-Boolean	Check if there is not any voids in branch and the boundary is a rectangle too, so the boundary is a bigboy.
15	Recogniser functions	ExtractBranch	-Group of curves -Double	-Branch -Group of curves	Get a list of curves and find the largest boundary, voids and ignorable curves in it. the double can be set by user and defines the minimum area of voids. The surrounded curves with less area are recognised as Ignorable curves. This

16	CleanBranch	-Branch	-Branch	Get a branch and check if any void is touching the boundary
17	RectAreaMat	-A curve	-A list consists of curve and its area	Creating n*2 Matrix of a curve and its Area as a double
18	BiggestCurve	-List of curves	-A curve	Finding the largest curve based on its area
19	XYSet	-List of curves	-Two list of double	It gets the x and y coordinates of one or more curves and collect them in two separated list.
20	UnitRectangles	-Branch	-List of rectangles	Finding smallest Rectangular Units in a bunch of curves and negative curve based on the virtual cartesian coordination extracted by XYSet function
21	NegativeCurve	-A curve	-A curve	Negative part of a curve when surrounded in a smallest possible rectangle.
22	BoundingRect	-A curve	-A rectangle	The smallest possible rectangle that the curve can be surrounded in it
23	AllRectangles	-A curve	-List of rectangles	The all-possible rectangles that can be created by virtual cartesian coordinates for a curve
24	allRectanglesOrg Void	-A Branch	-List of rectangles	The all-possible rectangles that can be created by virtual cartesian coordinates for a branch
25	EnterBigboy	-A branch	-A rectangle -A Group	Find the largest rectangle surrounded by boundary and does not have intersection with voids, and subtract it from boundary and returns the remaining shapes as a group

Table 4 – data types and function of Division Algorithm

Figures 24,25, and 26, the detailed work-flow of Division Algorithm is shown. After introducing the functions and object types in Table 4, now the detailed work-flow for dividing the shapes and preparing it for next stage – stage 3: Joining algorithm – is explained. Here we tried to describe the work-flow in point of view of programming language. In fact, we tried to connect the primary idea to Python scripts that has been developed for second stage of solution, stage 2: Division Algorithm.

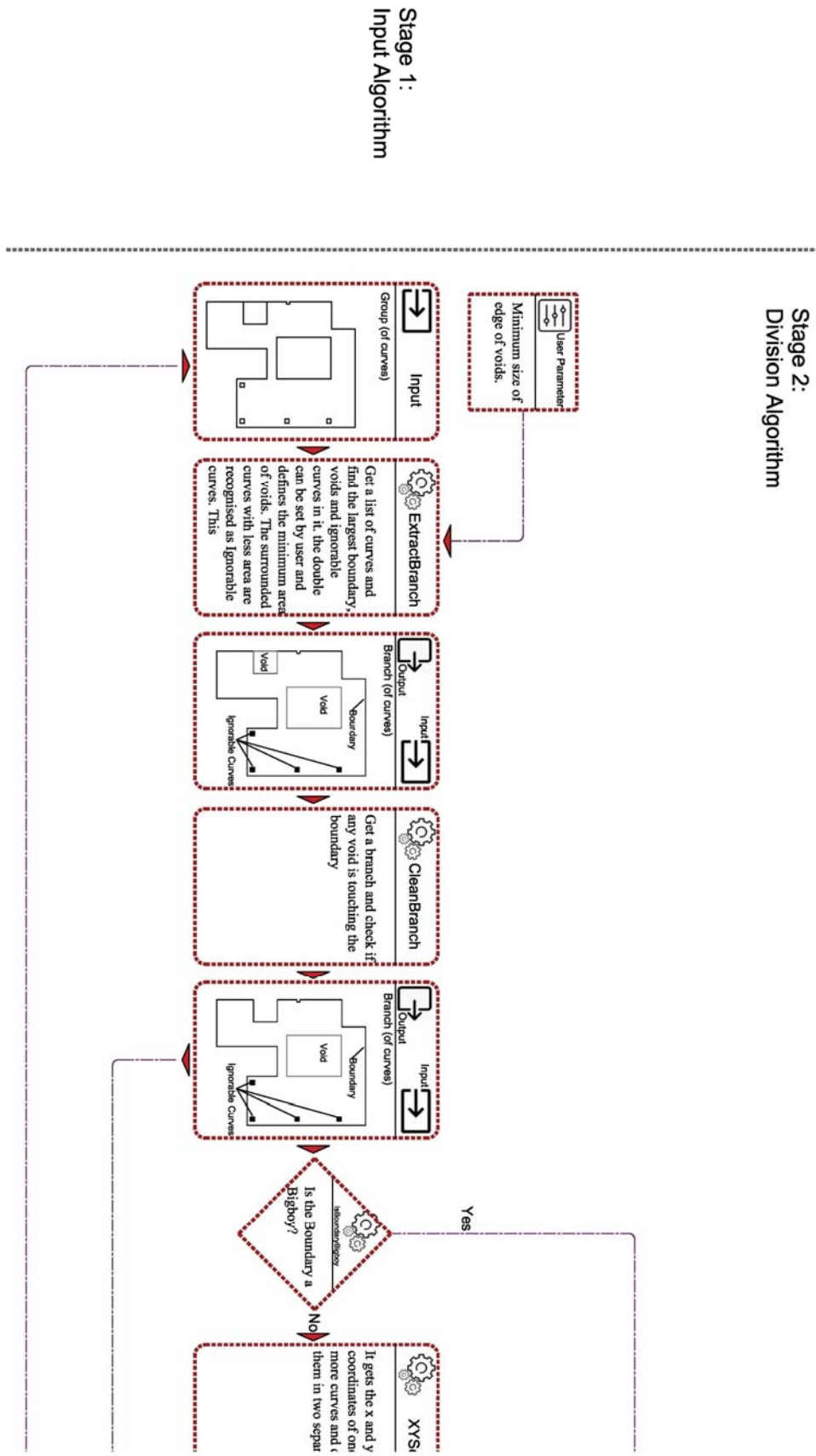


Figure 24 – Detailed process-flow of Division Algorithm (Part 1)

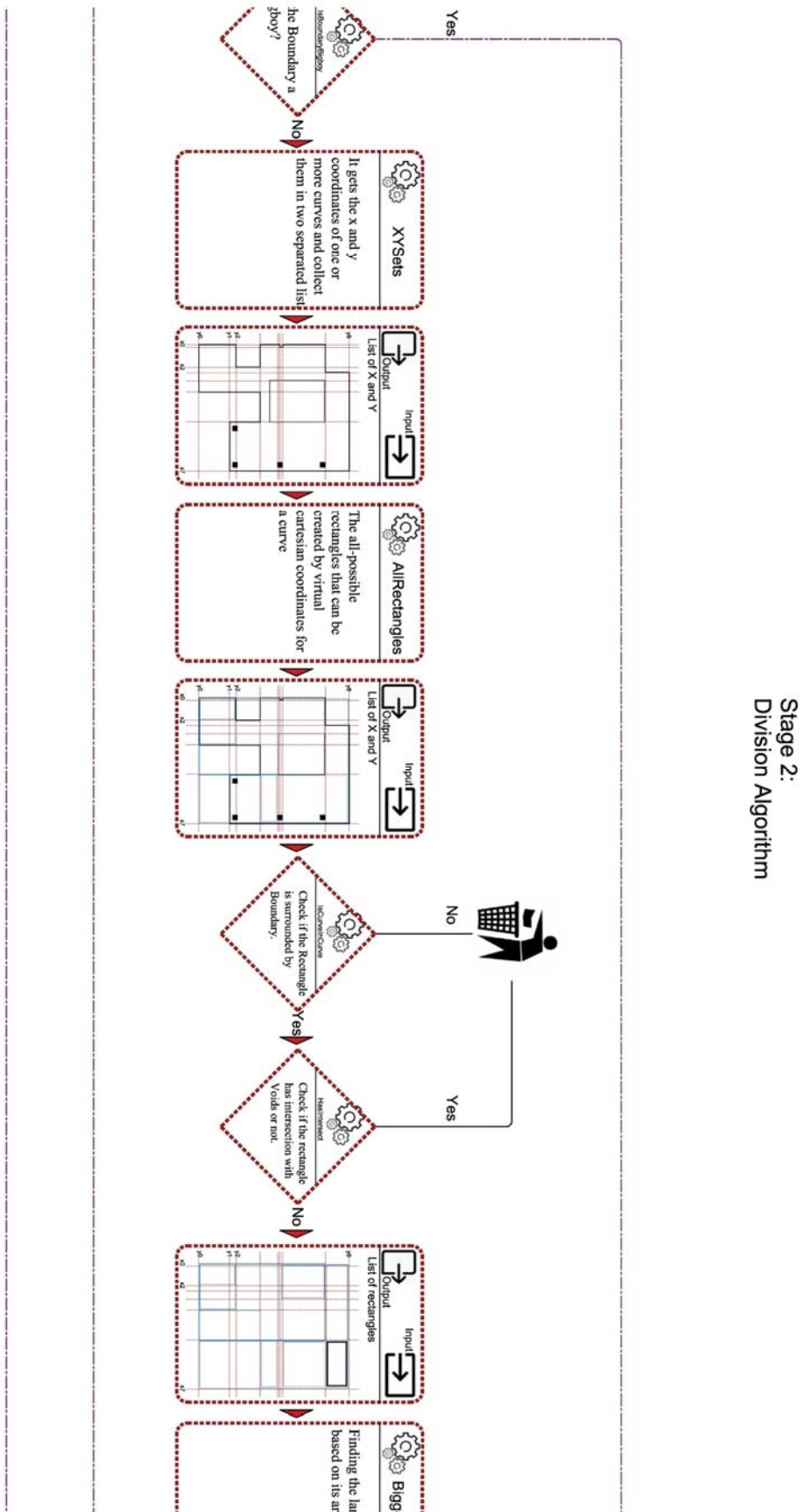


Figure 25 – Detailed process-flow of Division Algorithm (Part 2)

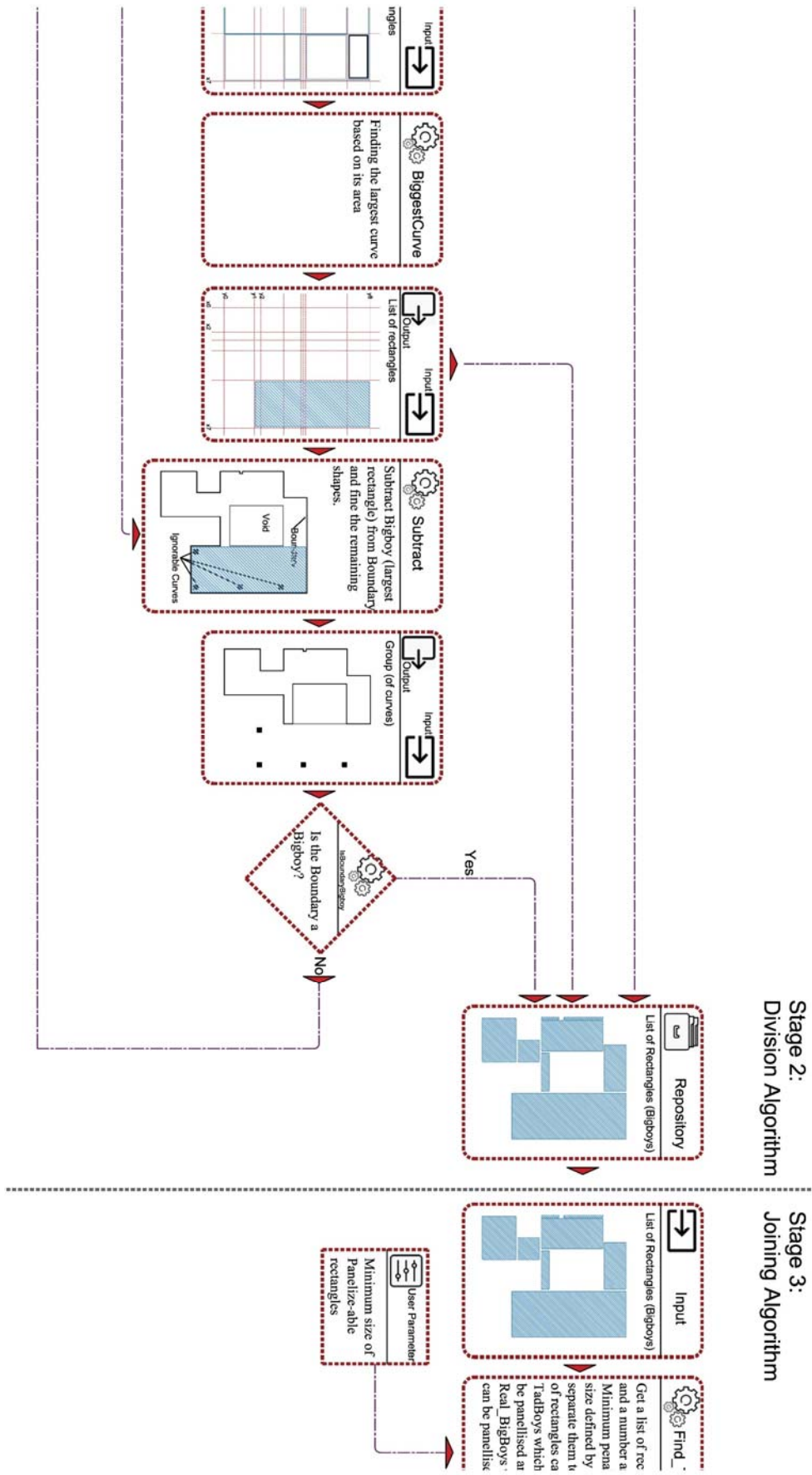


Figure 26 – Detailed process-flow of Division Algorithm (Part 3)

4.2. Joining Algorithm:

4.2.1. Main concept

The Joining algorithm is the third stage of the solution. This algorithm has been developed to simulate a part of human way of thinking in the problem solution. This part happens implicitly in a human brain and it is not possible to realise the exact process. When a human starts to divide a curve and prepare it for panelising, there are always some small rectangles that can be added to a neighbour larger rectangle and the resulting shape may be panelised in a more optimized way.

As it is shown in previous pages, the Division Algorithm produces a list of rectangles that cover the boundary. Not all of these rectangles can be panelised, due to an important production rule: the manufacturer, here Fractus, cannot produce panels with a width that is smaller than 30 cm. This means that if there is a rectangle where one of its edges is less than 30 cm, that rectangle cannot be covered by a panel. Here are two solutions for such cases:

- Change the geometry of the boundary, voids, or both, so that the design of that element is changed; or
- Join that irregular rectangle with another acceptable rectangle and create a L or T shape boundary and pass it for the Panelising algorithm.

Here, both of these alternatives are considered and the algorithm tries to respond to both. Some times the design stage has been proved and changing it is not possible. On the other hand, in some cases joining an irregular rectangle with one of its neighbours leads to a non-acceptable result. Considering Figure 27, it is possible to see regular and irregular rectangles in the output of the Division Algorithm that are respectively called **BigBoy** and **TadBoys**.

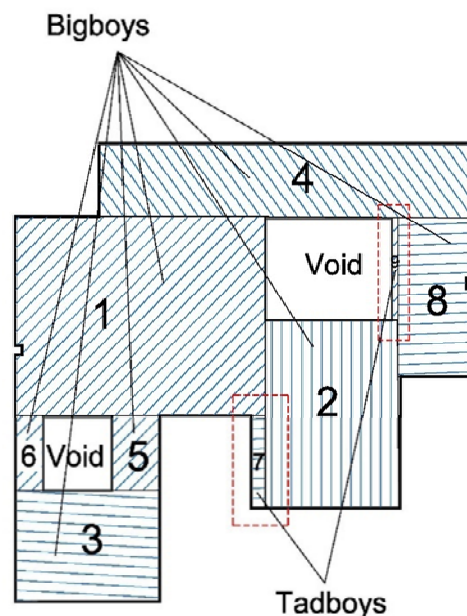


Figure 27 – Regular and irregular Rectangles resulted from Division Algorithm

Joining algorithm uses a simple logic to find the Tad Boys, recognise its neighbours, perform the junction and, at the end, evaluate the junction and select the best one, regarding the evaluation criteria. The evaluation rules are defined by the manufacturer considering their production methods.

Here the main work-flow is shown in Figure 28.

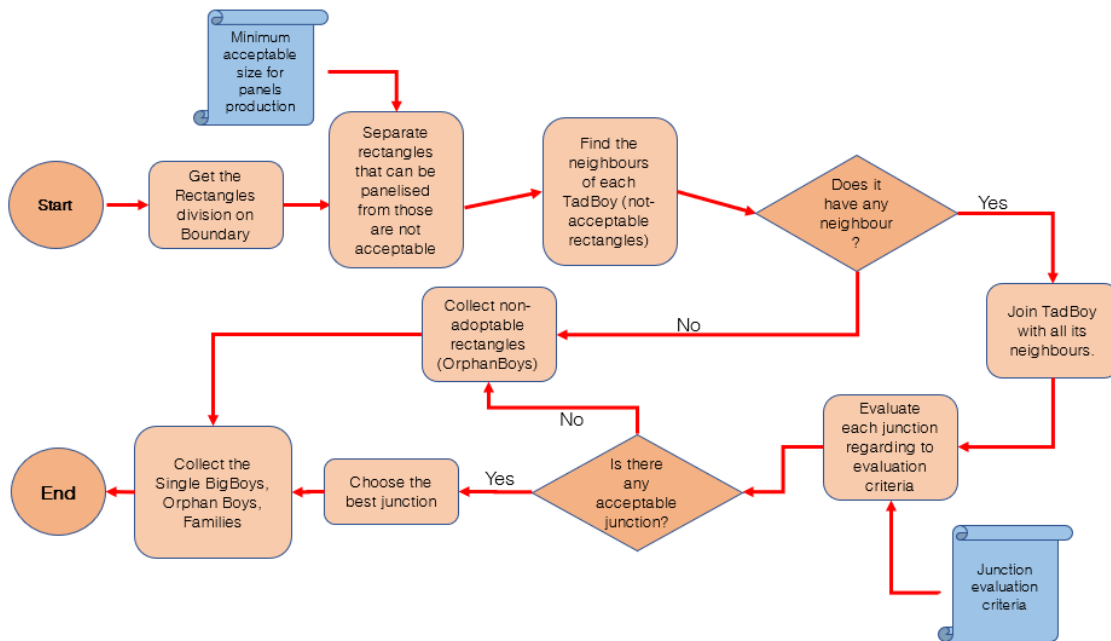


Figure 28 – Detailed work-flow of the joining Algorithm

4.2.2. Structure of Joining algorithm:

The structure of Joining algorithm is dependent on the rules that the user defines for it. Two different parts should be defined by the user:

- Minimum acceptable size for panel production. The Algorithm can separate acceptable rectangles from not-acceptable rectangles called TadBoy. At present, the minimum size declared by manufacturer is 30 cm.
- The criteria for evaluating junction. Three factors have been defined for this evaluation:
 - The direction of joining rectangles
 - The size of parent neighbour
 - Number of corners or edges of the resulting shape

This part explained by some figures in following tables.

Table 5 – Evaluation of direction of junctions

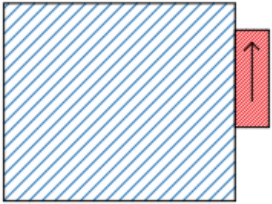
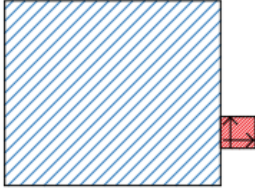
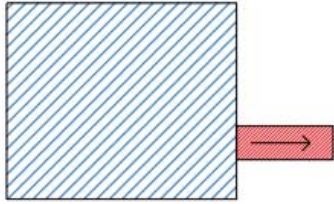
Evaluation Criteria	High	Medium	Low
The direction of joining rectangles			
	Join on longer edge	TadBoy is a square	Join on shorter edge

Table 6 – Evaluation of the number of corners of junctions

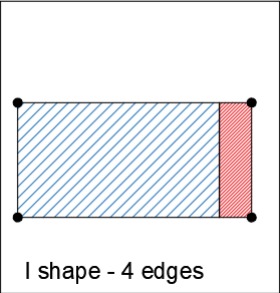
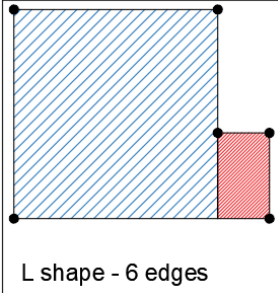
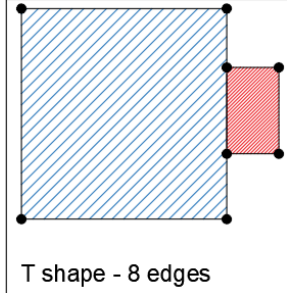
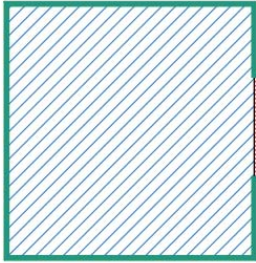
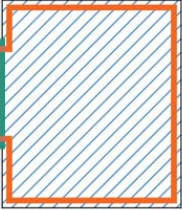
Evaluation Criteria	High	Medium	Low
Number of edges of resulted shape after junction			
	I shape - 4 edges	L shape - 6 edges	T shape - 8 edges

Table 7 – Evaluation of joining to the larger or smaller neighbour.

Evaluation Criteria	low	High
Joining with Larger or smaller neighbour?		
	Jion with larger neighbour	Jion with smaller neighbour

4.2.3. Detailed work-flow

The Joining algorithm performs two vital tasks: creates the junctions for each TadBoy (rectangles with an edge shorter than the defined minimum size), and evaluates the junction in order to select the best one. The steps of these processes are explained below:

- Get the list of rectangles from the previous stage.
- Get the minimum possible size for the panel. This parameter is set by the user.
- Distinguish panelise-able rectangles from non-panelise-able rectangles, respectively called BigBoys and TadBoys.
- Find the neighbours of each TadBoy in four directions (up, right, bottom, and left).
- Join each TadBoy with its neighbours, one by one, and collect them in a list.
- Get the evaluation criteria that are defined by user.
- Evaluate each junction regarding the evaluation criteria.
- Select the best junction.
- Modify the shapes and return the results that consist of:
 - Single BigBoys that do not join to any other rectangles.
 - Families that include one or more TadBoys adopted by a BigBoy.
 - OrphanBoys that cannot be adopted by any BigBoys.

Some data structures and functions were created to perform the described steps. Beyond the new data structures that were defined in the Joining algorithm, some functions were defined in tree context: 1) Recogniser functions, 2) Executive functions, and 3) Evaluation functions.

No	Type	Name	Input	Output	Description
1	Data Type	Real BigBoy			The rectangles that the dimension of their edges is equal or more than Minimum Panelling size.
2		TadBoy			The rectangles that one or both edges of them are shorter than Minimum Panelling size.
3		Minimum Panelling size			A number defined by user that rectangles with an edge shorter than that number, cannot be panellised due to production limits.
4		Neighbourhood			A list consists of a TadBoy and its neighbours in for directions
5		Junction			A 4*4 matrix consists of four list of a direction name, TadBoy, a neighbour in specific direction such as top, and the united shape of TadBoy and the neighbour
6		Evaluation			A list of scores that represents each evaluation score and total score of junctions.
7	Recogniser functions	Find_TadBoys	List of Rectangles Double	List of rectangles (real BigBoys) List of rectangles (TadBoys)	Get a list of rectangles and a number as Minimum penalizable size defined by user and separate them to two list of rectangles called TadBoys which cannot be panellised and Real_BigBoys which can be panellised.
8		Tad_neighbors	A TadBoy List of BigBoys	A list of rectangles	Find the neighbours of a TadBoy in 4 directions

Table 8 – data types and function of Joining Algorithm - continue

Table 8 – data types and function of Joining Algorithm - continue

				(top, right, bottom, and left) if exist and collect them in a list. If there is not a neighbour in specific direction, it returns None.
9	RectangleProp	A rectangle	A list of number	Extracts some properties of a rectangle consist of X and Y coordinates, edge dimensions and area
10	BoundingRect	A Curve	A rectangle	Create the smallest possible rectangle that can surround the given curve.
11	Adoption	A neighbourhood	A Junction	Join the neighbours of a TadBoy and create a junction for each direction.
12	EvJunctionDirection	A junction	A number	Evaluate the junction edge in TadBoy if it is longer edge or not and give a high or low value. If TadBoy be a square, It returns zero.
13	EvJunctionCorners	A junction	A number	Evaluate the number of corners of a united shape. Less the number of corners, get higher score.
14	EvAreaRate	A junction	A number	Evaluate the rate of TadBoy area divided by BigBoy area. As a requirement, joining with smaller BigBoy is demanded.
15	EvaluateJunction	A junction	A list consists of string and numbers	Collect the scores of junctions and returns a list consist of direction of junction, scores of 3 kind

				of evaluations, and total of them.
16	Eva_Result	An Evaluation	string	Evaluate if the best junction based on its scores.

Table 8 – data types and function of Joining Algorithm

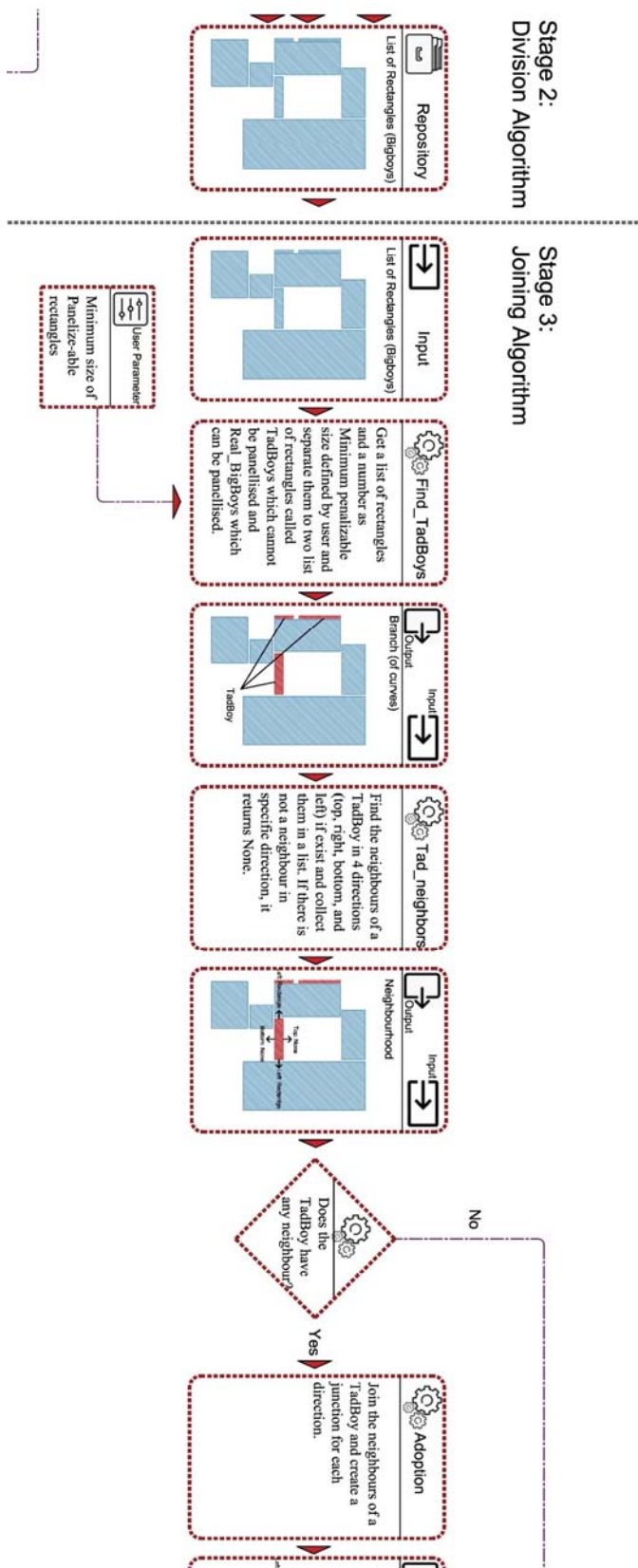


Figure 29 – Detailed process-flow of Joining Algorithm (Part 1)

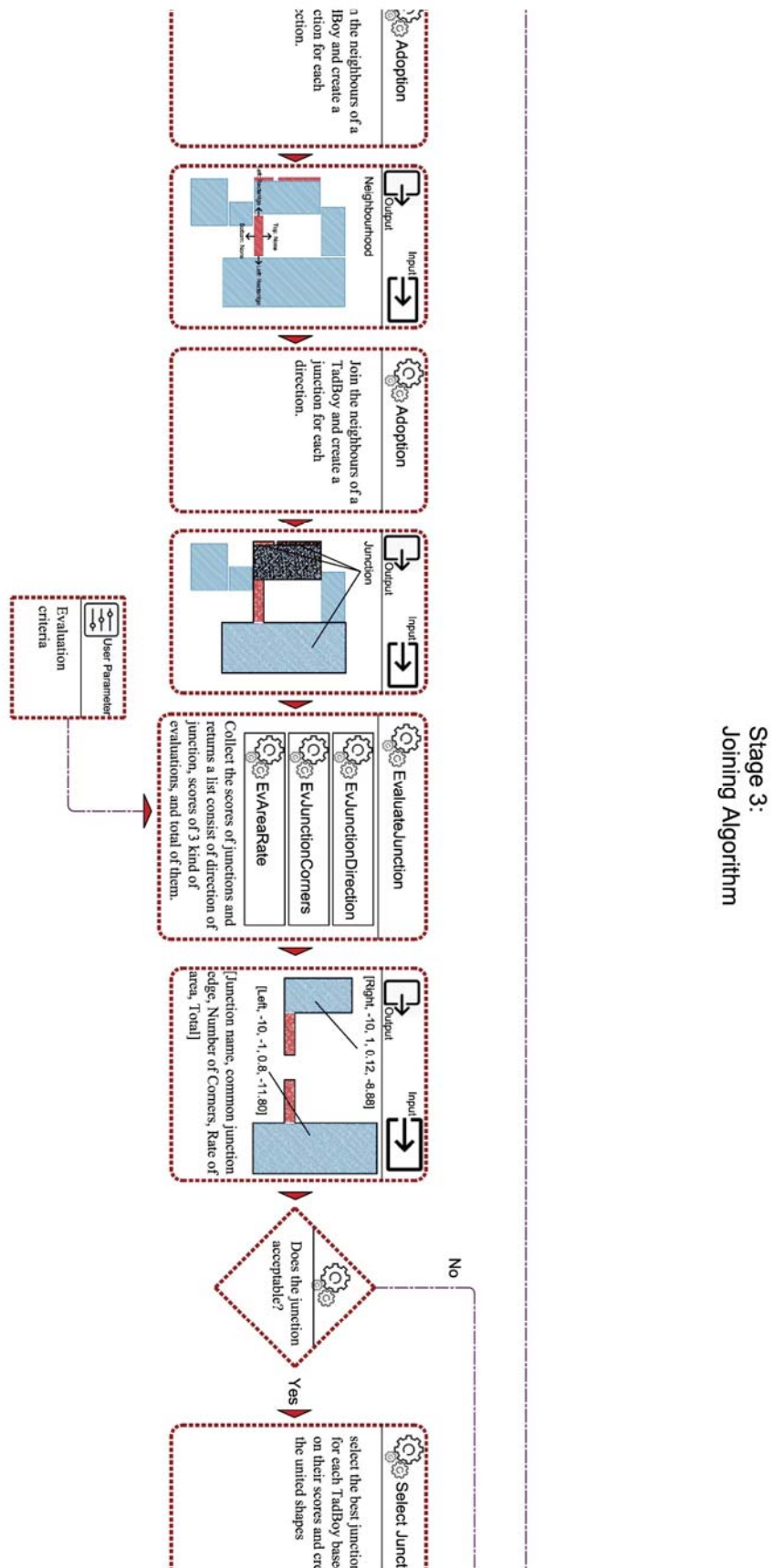


Figure 30 – Detailed process-flow of Joining Algorithm (Part 2)

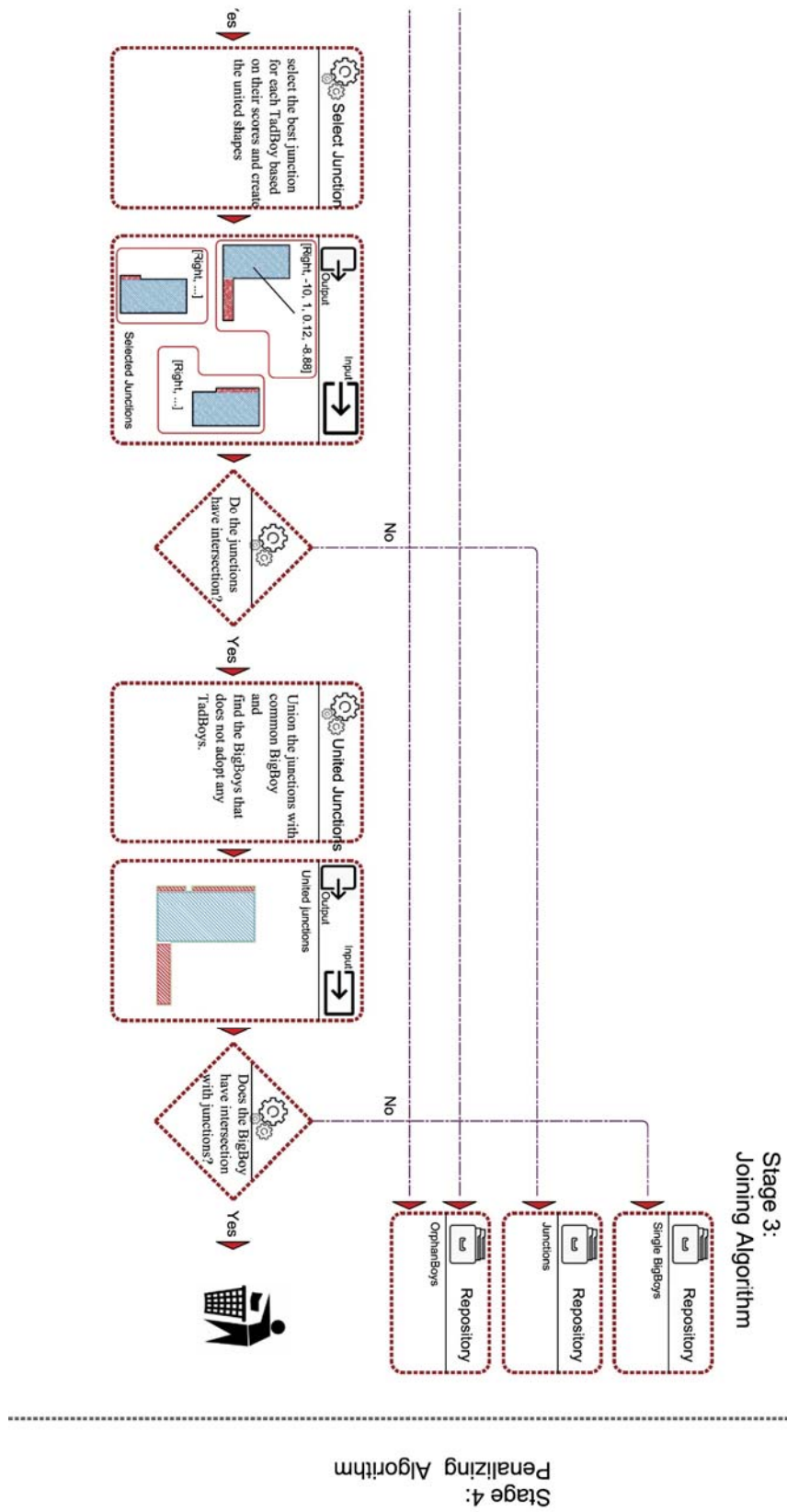


Figure 31 – Detailed process-flow of Joining Algorithm (Part 3)

4.3. Panelising Algorithm

4.3.1. Main Concept

The Panelising algorithm is the last stage of the calculation process and ends up the “Calculation” algorithms. Contrary to its simple appearance, the calculation associated with this stage is complex and needs an extra effort to be solved. The main reasons that make it difficult are:

- Engineering to Order approach for creating panels that the dimension of panels can be various.
- Various adjustable factors that are set by user
- Existing of non-rectangular shapes in the panelling process.

As previously described, the third stage, “Joining” algorithm finalizes the division of the element boundary, creates a group of rectangular and non-rectangular shapes and pass them to the fourth stage, the Panelising algorithm. In this stage, the algorithm tries to find the best arrangement direction and the best size of panels for each shape separately, considering some predefined requirements. As it mentioned before, the manufacturer, Fractus company, has some limitation and rules for producing the panels:

- The edge of a panel cannot be shorter than 30 cm.
- The longest edge possible to produce is 700 cm.
- Due to the size of composite boards that are used to cover the surface of panels, the maximum breadth size of a panel is 122 cm. (see figure 32)
- For the length of the panels, it is possible to go up to 700 cm but 350 cm is preferred.
- when one of a shape edge is divided by 122 cm and the remaining number is shorter than 30 cm, only the last two panels should be divided equally and the others must be kept 122 cm.
- The direction of panels is not important and depends on which arrangement direction is more optimised.

In another word, we can say that one edge of a panel should be longer than 30 cm and shorter than 122 cm. The other edge of that panel should be longer than 30 cm and shorter than 700 cm but, 350 cm is preferred (see Figure 33).

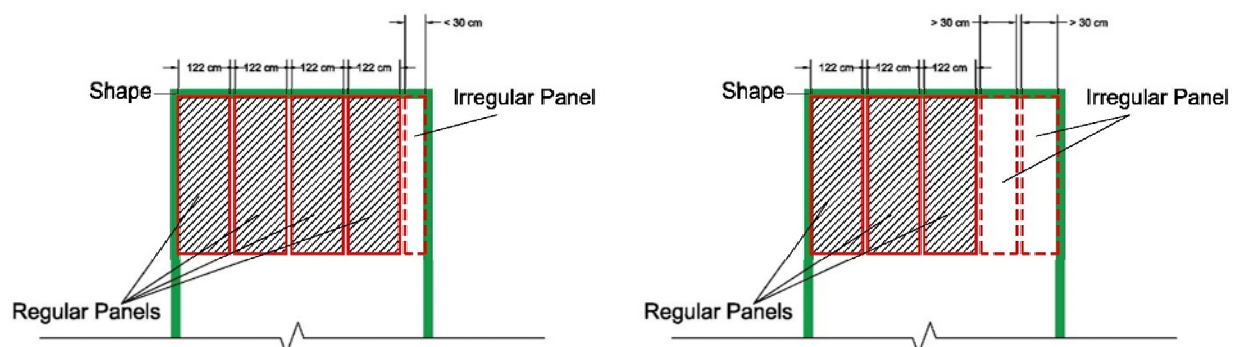


Figure 32 – Rules for dividing the breadth of panels, considering regular panels and irregular panels

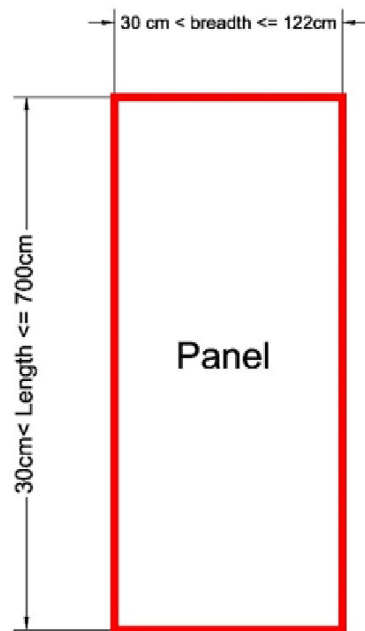


Figure 33 – Production rules of breadth and length of panels

4.3.2. Structure of the Panelling algorithm

According to the rules and requirements established by Fractus Company, a work-flow was designed. The main steps defined in work-flow are listed below:

- Getting the final shape from previous stage (Joining algorithm)
- Extracting the length and breadth of the various shapes.
- Arranging the panels in both directions X and Y.
- Counting the number of panels in each direction.
- Selecting the direction with a smaller number of panels.
- Arranging the panels in the selected direction.
- Getting the intersection of panels with the shape, if the shape is not rectangular.
- Collecting the panels.

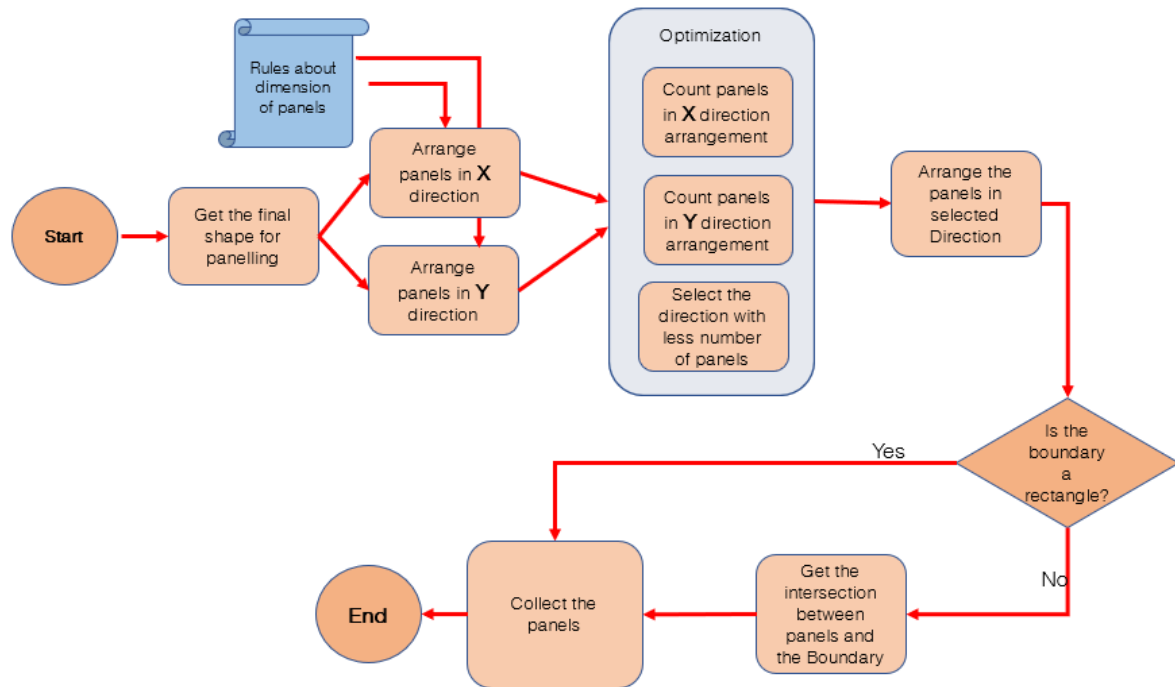


Figure 34 – Work-flow of Panelling Algorithm

4.3.3. Detailed work-flow

At a first step, the algorithm draws a rectangle around the shape, called Bounding rectangle. As it is possible to see in the previous stage – Joining algorithm – to create both type of rectangular and non-rectangular shapes and passes them to Panelling Algorithm. So, for panelling them, the algorithm should convert them into rectangles. The next step is to examine the length and breadth of the resulting rectangles according to the defined rules. In other words, each edge of the rectangle is divided, considering two groups of rules:

- Dividing the edge by breadth of panel
- Dividing the edge by length of panel

The mentioned rules can be organised in two groups for length and breadth of panels:

Rules of group A: Division rules considering the breadth of panels:

- The breadth of panels cannot be shorter than 30 cm
- The breadth of panels should be 122 cm
- If the remaining distance, after dividing the edge by 122 cm is less than 30 cm, this panel must be added to a previous panel and the result should be divided by 2.

Rules group B: Division rules considering the length of panels:

- The length of panels should be longer than 30 cm and shorter than 700 cm.

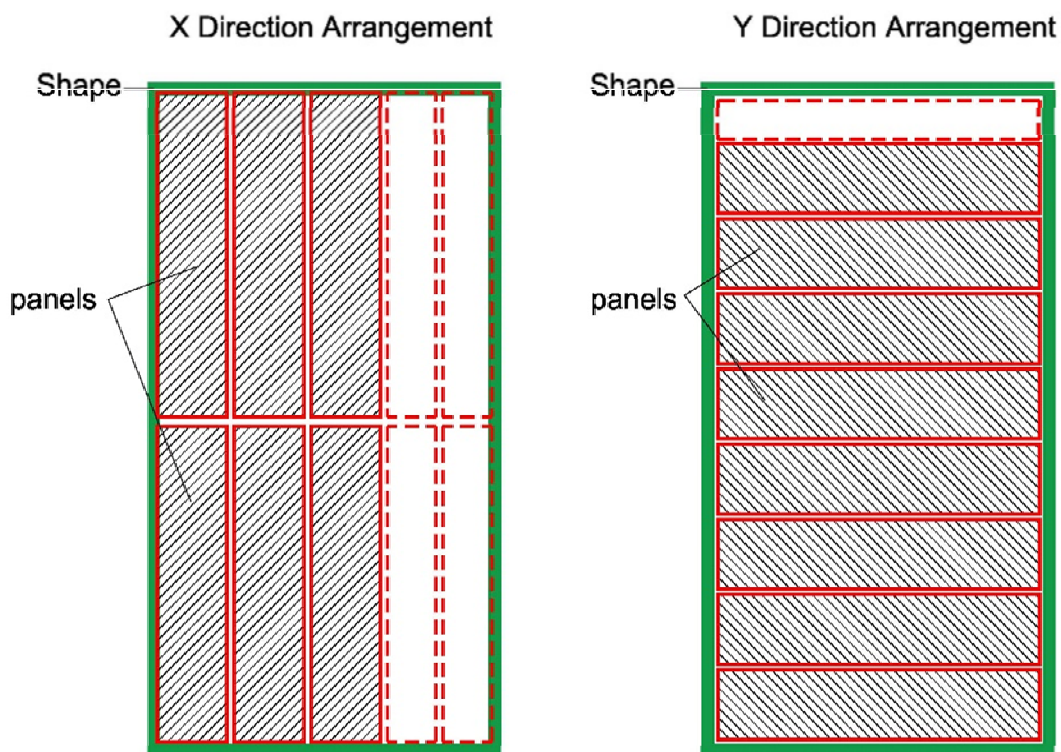


Figure 35 – Arrangement of panels in a same rectangle in both direction X and Y

For example, in the above figure, Panelising algorithm starts to divide both sides of the shape (green rectangle) according to the two groups of rules. The result is an integer or a natural number that can calculate the number of panels in each direction. The algorithm compares these numbers and selects the smaller one and the corresponding direction of arrangement.

Based on selected direction, the panels should be arranged. When the shape is a rectangle, the panels are directly sent to repository. When the shape is not a complete rectangle, the intersection between each panel and the original shape will be calculated and after that will be sent to repository.

As in previous parts, the data types and functions defined for the algorithm in Python are explained and also show the detailed work-flow through a diagram.

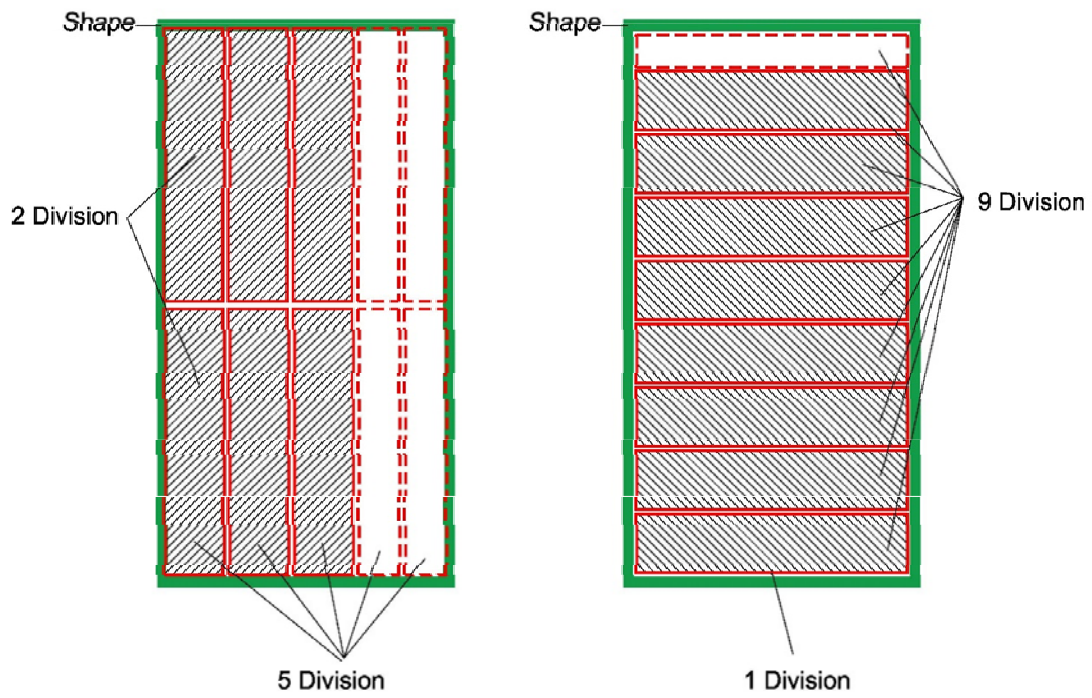


Figure 36 – Arrangement of panels in both direction X and Y and the results: 10 panels in X direction and 9 panels in Y direction.

Table 9 – data types and function of Panelling Algorithm

No	Type	Name	Input	Output	Description
1	Data Type	Curve	-	-	a boundary by only perpendicular angles
2		Shape	-	-	A curve that is ready for panelising
3		Minimum possible dimension	-	-	The dimension that defined by user and each edge of panels should be larger than it.
4		Dimension of Breadth	-	-	The maximum and optimised dimension defined by user for breadth of panels
5		Dimension of optimized length	-	-	Dimension that is defined for optimised length
6		Dimension of maximum length	-	-	The maximum dimension that is possible for the length
7	Executive functions	BoundingRect	- A Curve	- A rectangle	Create the smallest possible rectangle that can surround the given curve.
8		Generate_Panels	- A Shape - Dimension of Breadth - Dimension of optimized length - Dimension of maximum length - Minimum possible dimension	- Panels	Generate the panels for each shape. All the evaluation functions are embedded in it. It can evaluate the optimized direction, performs the division and intersect panels with original shape and returns the final panels.
9	Evaluation Functions	Division_A	- An edge - Dimension of Breadth - Minimum dimension	- Number of regular panels - Dimension of regular panels - Number of irregular panels	Divides the edge by the first group of rules and return the number and dimension of regular and irregular panel.

Table 9 – data types and function of Panelling Algorithm

			- Dimension of irregular panels	
10	Division_B	<ul style="list-style-type: none"> - An edge - Dimension of optimized length - Dimension of maximum length - Minimum possible dimension 	<ul style="list-style-type: none"> - Number of regular panels - Dimension of regular panels - Number of 	Divides the edge by the second group of rules and return the number and dimension of panels.
11	Evaluate_DirectionX	<ul style="list-style-type: none"> - A rectangle - Dimension of Breadth - Dimension of optimized length - Dimension of maximum length - Minimum possible dimension 	<ul style="list-style-type: none"> - Number of panels in X direction 	Returns the number of panels created in X direction arrangement for the given rectangle.
12	Evaluate_DirectionY	<ul style="list-style-type: none"> - A rectangle - Dimension of Breadth - Dimension of optimized length - Dimension of maximum length - Minimum possible dimension 	<ul style="list-style-type: none"> - Number of panels in Y direction 	Returns the number of panels created in Y direction arrangement for the given rectangle.
13	Select_Direction	<ul style="list-style-type: none"> - A rectangle - Dimension of Breadth - Dimension of optimized length 	<ul style="list-style-type: none"> - Division in X direction - Division in Y direction 	calculates the number of panels created in each direction and returns two integer.

-
- Dimension of maximum length
 - Minimum possible dimension
-

Table 9 – data types and function of Panelling Algorithm

Here the detailed work-flow of Panelling algorithm is shown in figure 37 and 38.

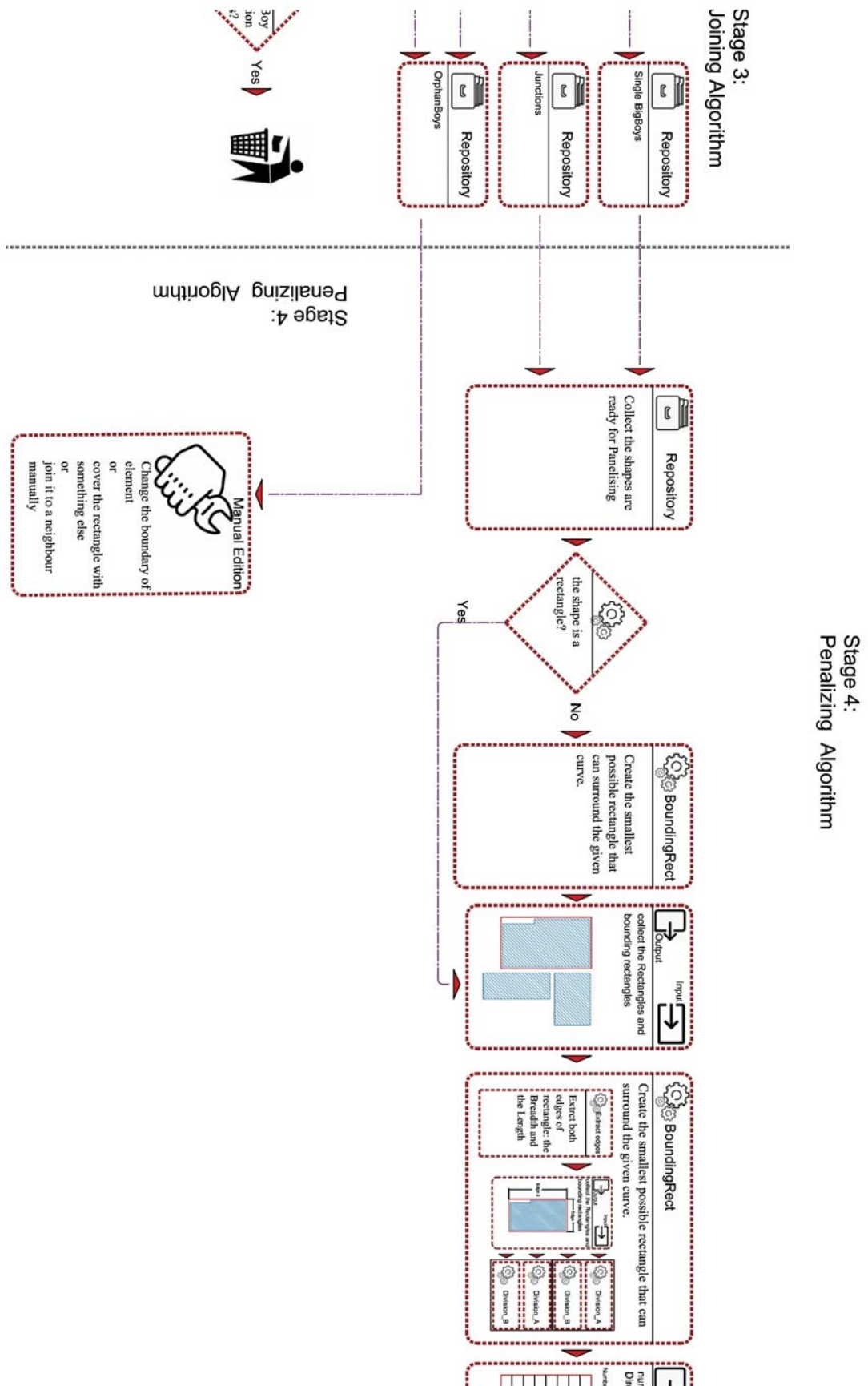


Figure 37 – Detailed process-flow of Panelling Algorithm (Part 1)

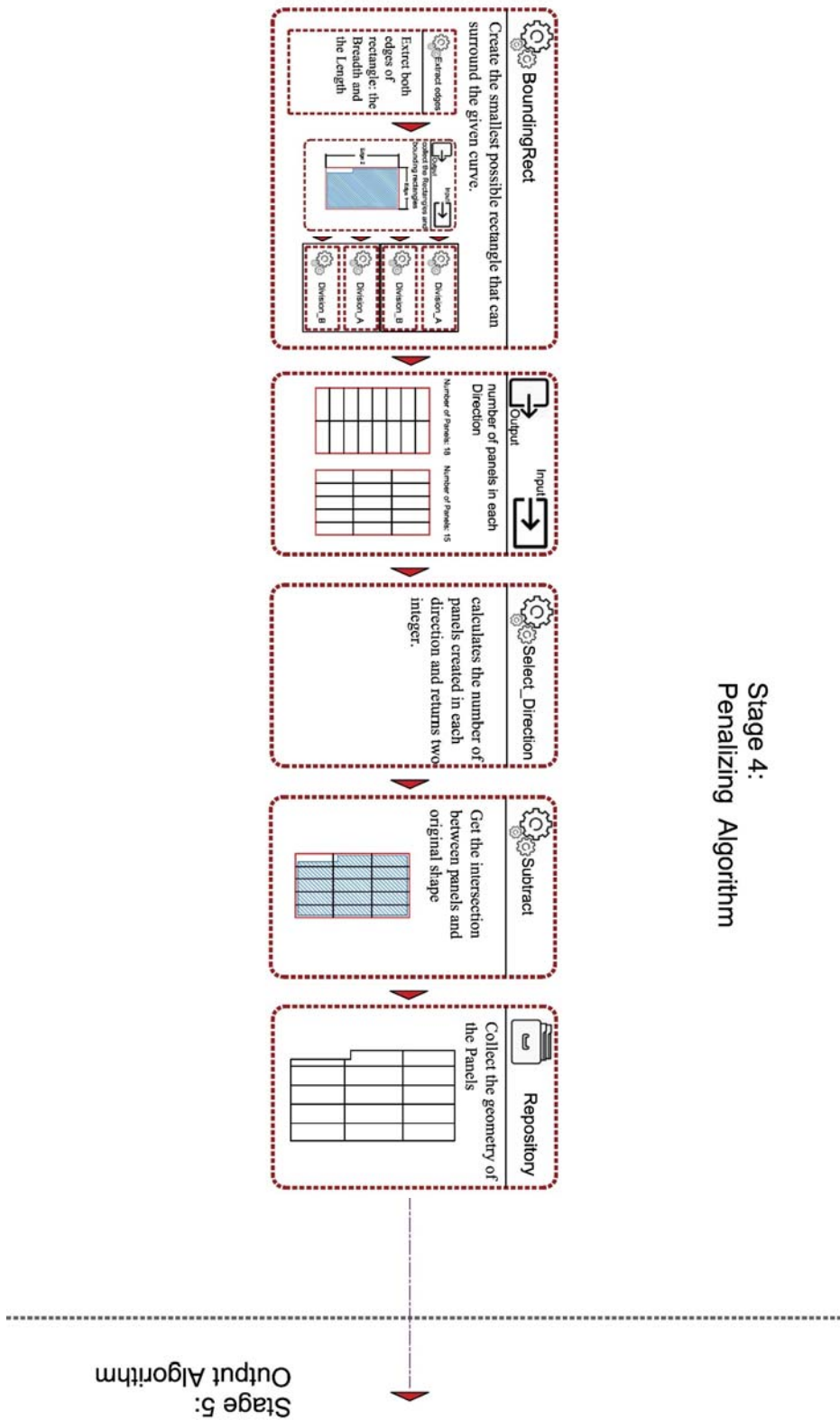


Figure 38 – Detailed process-flow of Panelling Algorithm (Part 2)

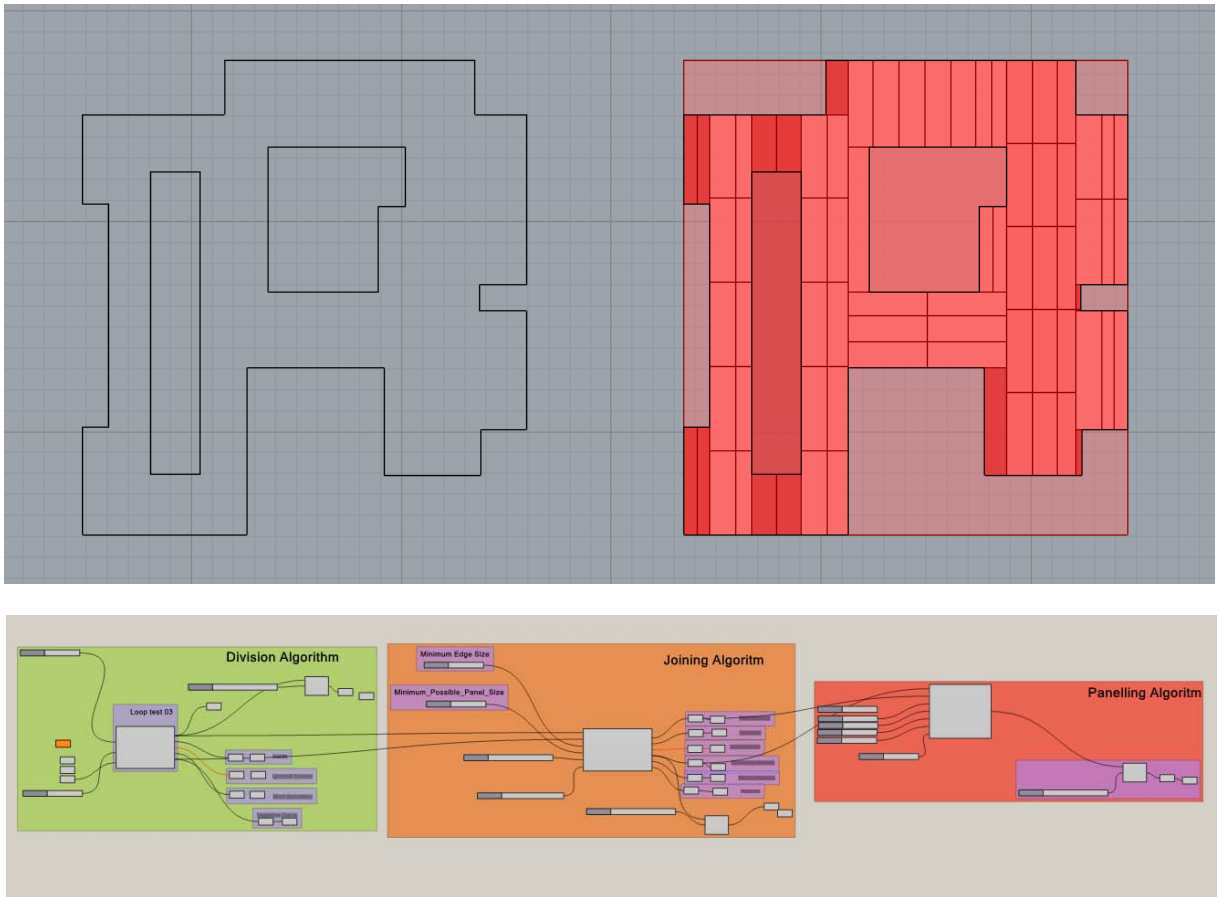


Figure 39 – A sample result of Algorithms calculation

5. CONCLUSION AND FUTURE DEVELOPMENTS

Prefabrication and manufacture in the AEC industry can be a way to increase the productivity, and reduce costs, time and other resources used in buildings life cycle. The main challenge in this improvement is the specific characteristics of the final product: each building is as a product of the AEC industry and at the same time is unique. This means that the ordinary and common methods of manufacturing, such as Mass-production, cannot be easily adapted by the AEC industry. Each product is unique and so many of its sub-products (elements) will be unique.

This research went through the new technologies related to the industrialization of building production and gradually narrowed down to the goals set.

The main challenges were extracted from practical experiences experienced in the AEC industry: How can we make the process of designing specific elements automatic? In this case the specific challenge was: how to cover a building floor with panels automatically?

The main aspects reviewed include:

- Manufacturing methods in AEC industry
- Panel properties, rules and restrictions of the production process
- BIM tools that can be used in the process
- Programming logic associated with the division and optimization of panel arrangement

There are two main approaches in manufacturing: Mass-production and Engineer-To-Order. ETO is a method that can be adapted in AEC industry. It provides enough freedom for the designer to satisfy client's requirement and meanwhile respond to production rules and limits. The coordination of these two far sides of design and fabrication is difficult, complex and error pron.

Building information Modelling (BIM) and Artificial Intelligence (AI) can be associated to achieve a solution. BIM provides an environment for geometrical and non-geometrical data and AI can perform the process automatically and accurately. In this research, a task that is usually done manually by a human, was analysed and translated to a digital process. The thinking process was extracted from human's behaviour. BIM as an environment provides a context for the whole process. Enquiring data from model, processing, generating result and applying the results back to the model were performed in the BIM environment. BIM is able to support all these aspects without interruption, errors, and data loss.

For certain challenges, such as the one addressed in this research work, there is more than one solution. One of the alternatives is to use Heuristic. A heuristic, or a heuristic technique, is an approach to problem-solving that uses a practical method or various shortcuts in order to produce solutions that may not be optimal but are sufficient given a limited timeframe or deadline. In this research, the human logic for solving the problem was examined and analysed and the digital process was extracted from it. The algorithm may not find the best solution, but in comparison to a human, its performance is much higher in quality, takes less time and always finds a single solution.

There are always some rules and restrictions in the process of designing and fabrication of an element. These should be considered. Also, some of them may be changed by the designer, the producer, the

client or even depended on the condition they are in. The algorithm should provide a situation that can be changed and modified by user. They can be called adjustable parameters.

Python is a programming language, that has been more and more popular in recent years. Using it in this research because of its simple syntax, flexibility and rich library and users' community, was a great success. Grasshopper plug-in in Rhinoceros works with Python version 2 and some new functions defined in version 3, are not supported.

There are two alternatives for usage the APIs. Rhinoceros API and Grasshopper API. Both alternatives were examined in this research. Rhinoceros API is really richer and stronger than Grasshopper API but, working with lists and matrixes posed some problems. The solution was to switch to Grasshopper API and regenerate all the codes to fix the problems.

To work with API, regarding to the first developer logic and goals, each function should be examined and carefully tested in a separated environment by as much as possible conditions.

The algorithm generated in this research has more than 50 functions. Creating functions for the algorithm instead of using API, during the developing process, can lead to a stronger and more accurate algorithm. Also maintaining and modifying the algorithm in the future is easier and more accessible.

This research can be developed in some different directions in the future:

- Transferring to a plug-in for BIM authoring software like Revit, Tekla structures, Bentley and so on
- Be able to work with shapes which have some non-perpendicular angles.
- To provide the ability to define more rules and restrictions to the user.
- To make it possible for users to have more than a single solution and to select one of them based on their properties.

REFERENCES

Journal article

- Abdul Nabi, M., & El-adaway, I. H. (2020). Modular Construction: Determining Decision-Making Factors and Future Research Needs. *Journal of Management in Engineering*, 36(6), 04020085. [https://doi.org/10.1061/\(asce\)me.1943-5479.0000859](https://doi.org/10.1061/(asce)me.1943-5479.0000859)
- Almashaqbeh, M., & El-Rayes, K. (2021). Optimizing the modularization of floor plans in modular construction projects. *Journal of Building Engineering*, 39(November 2020), 102316. <https://doi.org/10.1016/j.jobbe.2021.102316>
- Curletto, G. (2014). Parametric modeling in form finding and application to the design of modular canopies. *WIT Transactions on the Built Environment*, 136, 223–234. <https://doi.org/10.2495/MAR140181>
- Gharehbaghi, K., Mulowayi, E., Rahmani, F., & Paterno, D. (2021). Case studies in modular prefabrication: Comparative analysis and discoveries. *Journal of Physics: Conference Series*, 1780(1). <https://doi.org/10.1088/1742-6596/1780/1/012009>
- Hammad, A. W. A., & Akbarnezhad, A. (2017). Modular vs conventional construction: A multi-criteria framework approach. *ISARC 2017 - Proceedings of the 34th International Symposium on Automation and Robotics in Construction*. <https://doi.org/10.22260/isarc2017/0029>
- He, R., Li, M., Gan, V. J. L., & Ma, J. (2021). BIM-enabled computerized design and digital fabrication of industrialized buildings: A case study. *Journal of Cleaner Production*, 278, 123505. <https://doi.org/10.1016/j.jclepro.2020.123505>
- Lu, N., & Korman, T. (2010). Implementation of Building Information Modeling (BIM) in Modular Construction: Benefits and challenges. *Construction Research Congress 2010: Innovation for Reshaping Construction Practice - Proceedings of the 2010 Construction Research Congress*. [https://doi.org/10.1061/41109\(373\)114](https://doi.org/10.1061/41109(373)114)
- Messaoudi, M., & Nawari, N. O. (2021). Virtual Permitting Framework for Off-site Construction Case Study: A Case Study of the State of Florida. In *Lecture Notes in Civil Engineering* (Vol. 98). https://doi.org/10.1007/978-3-030-51295-8_52
- N. Nawari (2012). BIM Standard in Offsite Construction. *Journal of Archit. Eng.*, no. June, pp. 107–113.
- Samarasinghe, T., Mendis, P., Ngo, T., & Fernando, W. J. B. S. (2015). BIM software framework for prefabricated construction: case study demonstrating BIM implementation on a modular house. *6th International Conference on Structural Engineering and Construction Management 2015*.
- Singh, M. M., Sawhney, A., & Borrmann, A. (2015). Modular Coordination and BIM: Development of Rule Based Smart Building Components. *Procedia Engineering*, 123, 519–527. <https://doi.org/10.1016/j.proeng.2015.10.104>
- Wei, P., Liu, Y., Dai, J. G., Li, Z., & Xu, Y. (2021). Structural design for modular integrated construction with parameterized level set-based topology optimization method. *Structures*, 31(March), 1265–1277. <https://doi.org/10.1016/j.istruc.2020.12.090>

- Mohamad Abdul Nabi, (2020). Modular Construction: Determining Decision-Making Factors and Future Research Needs
- Singh, M. M., Sawhney, A., & Borrmann, A. (2015). Modular Coordination and BIM: Development of Rule Based Smart Building Components. *Procedia Engineering*, 123, 519–527. <https://doi.org/10.1016/j.proeng.2015.10.104>
- Serdar Durdyev & Syuhaida Ismail (2019): Offsite Manufacturing in the Construction Industry for Productivity Improvement, *Engineering Management Journal*, DOI:10.1080/10429247.2018.1522566 To link to this article: <https://doi.org/10.1080/10429247.2018.1522566> (PDF) Offsite Manufacturing in the Construction Industry for Productivity Improvement.
- He, R., Li, M., Gan, V. J. L., & Ma, J. (2021). BIM-enabled computerized design and digital fabrication of industrialized buildings: A case study. *Journal of Cleaner Production*, 278, 123505. <https://doi.org/10.1016/j.jclepro.2020.123505>
- Monty Sutrisna , Jack Goulding,(2019). Managing information flow and design processes to reduce design risks in offsite construction projects
- S. Taylor and HSE, “Offsite Production in the UK Construction Industry – prepared by HSE: A Brief Overview,” 2012.
- Lopes, G., Vicente, R., Azenha, M., & Ferreira, T. M. (2018). A systematic review of Prefabricated Enclosure Wall Panel Systems: Focus on technology driven for performance requirements. *Sustainable Cities and Society*, 40, 688–703. doi:10.1016/j.scs.2017.12.027
- Lacey, A. and Chen, W. and Hao, H. and Bi, K. 2018. Structural response of modular buildings – An overview. *Journal of Building Engineering*. 16: pp. 45-56.
- Modular construction: From projects to products by Nick Bertram, Steffen Fuchs, Jan Mischke, Robert Palter, Gernot Strube, and Jonathan Woetzel, 2019
- Hisham M. Said, Tejaswini Chalasani, Stephanie Logan, (2015), Exterior prefabricated panelized walls platform optimization
- Lacey, Andrew William; Chen, Wensu; Hao, Hong; Bi, Kaiming (2019). "Review of bolted inter-module connections in modular steel buildings". *Journal of Building Engineering*. 23: 207–219. doi:10.1016/j.job.2019.01.035
- Lacey, Andrew William; Chen, Wensu; Hao, Hong; Bi, Kaiming (2018). "Structural Response of Modular Buildings – An Overview". *Journal of Building Engineering*. 16: 45–56. doi:10.1016/j.job.2017.12.008
- Mark Burnett, "Blocking Brute Force Attacks" Archived 2016-12-03 at the Wayback Machine, UVA Computer Science, 2007

Book

BIM handbook : a guide to building information modelling for owners, managers, designers, engineers and contractors / Chuck Eastman . . . [et al.]. — 2nd ed.

BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers / Rafael Sacks, Chuck Eastman, Ghang Lee, Paul Teicholz; John Wiley & Sons, Aug 14, 2018

Prefab Architecture: A Guide to Modular Design and Construction; By Ryan E. Smith, John Wiley & Sons, Dec 14, 2010

Dissertations

Jonatan Francisco Fernandes Salgado, L (2019) ‘Ciclo de Estudos Integrados Conducente ao Grau de Mestre em Engenharia Civil’ Master dissertation, University of Uminho, Braga, Portugal

Camilo Mercado Siles, 2020 ‘BIM-based Framework for Deconstructability Assessment of Steel Structures’ Master dissertation, University of Uminho, Braga, Portugal

Mohammed Refaat Mekawy Mohammed, 2020 ‘A Framework for Using BIM in Mass-Customization and Prefabrication in the AEC Industry’ Master Dissertation, Technische Universität München

Website

Mitchell, J.A. (2017) How and when to reference [Online]. Available at:
<https://www.howandwhentoreference.com/> (Accessed: 27 May 2017)

(<https://en.wikipedia.org/wiki/Prefabrication>)

LIST OF ACRONYMS AND ABBREVIATIONS

List of acronyms and abbreviations

2D	2 dimension
3D	3 dimension
AEC	Architecture, Engineering and Construction
AI	Artificial Intelligence
API	Application Programming Interface
BIM	Building Information Modelling
C#	C-Sharp programming language
CAD	Computer Aided Design
Python	Python programming language

This page is intentionally left blank

Appendices

APPENDIX 1: PANELLING CODES IN PYTHON**Division Algorithm codes in Python**

```

__author__ = "parsa"
__version__ = "2021.06.14"

import rhinoscriptsyntax as rs
import ghpythonlib.components as gh
import Grasshopper.Documentation
import math as math
Decimal = Decimal_No
# ok # Organize curves in Bigboy Curve, Void Curve and ignor Curve
def OrganizeCrv(crvs,minArea):
    bigboyCrv = BigBoy(crvs)
    restCrv = []
    inCrv = []
    ignorCrv = []
    VoidCrv = []
    for crv in crvs:
        if gh.Area(crv)[0] < gh.Area(bigboyCrv)[0]:
            restCrv.append(crv)
    for rc in restCrv:
        if IsCurveInCurve(rc,bigboyCrv):
            inCrv.append(rc)
    for ic in inCrv:
        if gh.Area(ic)[0] <= minArea:
            ignorCrv.append(ic)
        else: VoidCrv.append(ic)
    return bigboyCrv,VoidCrv,ignorCrv

# cleaning a bunch of curves
def CleanBranch(Branch):
    new_Boundary = Branch[0]
    new_VoidCurves = [i for i in Branch[1]]
    new_IgnoredCurves = [j for j in Branch[2]]
    for v in Branch[1]:
        if IsCurveInCurve(v,new_Boundary) and
HasCommonEdge(v,new_Boundary):
            new_Boundary = gh.RegionDifference(new_Boundary,v)
            new_VoidCurves.remove(v)
    Branch = [new_Boundary, new_VoidCurves, new_IgnoredCurves]
    return Branch

# ok # check if two are completely similar in shape and location
(duplicated curve)
def AreSimilar(a,b):

```

```

if IsCurveInCurve(a,b)==True and IsCurveInCurve(b,a) == True:
    return True
return False

# ok # Getting x and y coordinat of a rectangular curve
def XYSetCurve(Curve):
    while type(Curve) == type([]):
        Curve = Curve[0]
    points = []
    points = gh.Explode(Curve,True)[1]
    #points.pop(-1)
    xpoint=[]
    ypoint=[]
    zpoint=[]
    for point in points:
        f = gh.Deconstruct(point)
        i = round(f[0],Decimal)
        j = round(f[1],Decimal)
        xpoint.append(i)
        ypoint.append(j)
    #Sorting and removing duplicated values of X and Y lists
    X=set(xpoint)
    Y=set(ypoint)
    Xcord = sorted(list(X))
    Ycord = sorted(list(Y))
    return (Xcord,Ycord)

# ok # get the x and y of a list of curves
def xySets(Curves):

    if Curves == None:
        #return False,"No Input!"
    if type(Curves) != type([]):
        Curves = [Curves]
    if len(Curves)==0:
        return None,None
    points = []
    xpoint=[]
    ypoint=[]
    zpoint=[]
    for curve in Curves:
        points = gh.Explode(curve,True)[1]
        if type(points) == type([]):
            for point in points:
                f = gh.Deconstruct(point)
                i = round(f[0],Decimal)
                j = round(f[1],Decimal)
                xpoint.append(i)
                ypoint.append(j)
            else: return None,None
        #Sorting and removing duplicated values of X and Y lists
        X=set(xpoint)
        Y=set(ypoint)

```

```

Xcord = sorted(list(X))
Ycord = sorted(list(Y))
return (Xcord,Ycord)

def xySetsNew(Curves):
    if type(Curves) != type([]):
        Curves = [Curves]
    if len(Curves)==0:
        return False,"No Input"
    else:
        for crv in Curves:
            x , y = XYSetCurve(crv)
            x += x
            y += y
        Xpoint = set(x)
        Ypoint = set(y)
        Xcord = sorted(list(Xpoint))
        Ycord = sorted(list(Ypoint))
        return (Xcord,Ycord)

def IsCurveInCurve(inc,outc):
    if inc == None or outc == None:
        return False
    unc= gh.RegionUnion([inc,outc])
    if type(unc)==type([]) or unc==None:
        return False
    else:
        uncAr= round(gh.Area(unc)[0],4)
        incAr = round(gh.Area(inc)[0],4)
        outcAr = round(gh.Area(outc)[0],4)
        if uncAr == outcAr:
            return True
        if uncAr != outcAr:
            return False
# ok # check if two curves has intersection or not.
def HasIntersect(curve,refCurve):
    result = gh.RegionIntersection(curve,refCurve)
    if result == None:
        return False
    return True
# ok # Check if two curve have a common edge without intersect
def HasCommonEdge(incurve,outcurve):
    p0=gh.ConstructPoint(0,0,0)
    p1=gh.ConstructPoint(1,0,0)
    p2=gh.ConstructPoint(0,1,0)
    Inincurve = gh.OffsetCurve(incurve,+0.01, gh.Plane3Pt(p0,p1,p2),1)
    Outincurve = gh.OffsetCurve(incurve,-0.01, gh.Plane3Pt(p0,p1,p2),1)
    if IsCurveInCurve(incurve,outcurve):
        if IsCurveInCurve(Inincurve,outcurve)==True and
IsCurveInCurve(Outincurve,outcurve)==False:
            return True

```

```

    elif IsCurveInCurve(Inincurve,outcurve)==False:
        if HasIntersect(Outincurve,outcurve)==True and
HasIntersect(Inincurve,outcurve)==False:
            return True
        return False

def IsInandAligned(Curve1,Curve2):
    p0=gh.ConstructPoint(0,0,0)
    p1=gh.ConstructPoint(1,0,0)
    p2=gh.ConstructPoint(0,1,0)
    a = gh.OffsetCurve(Curve1,-0.01, gh.Plane3Pt(p0,p1,p2),0)
    if IsCurveInCurve(Curve1,Curve2)==True:
        if IsCurveInCurve(a,Curve2) == True:
            return True,False
        elif IsCurveInCurve(a,Curve2) == False:
            return True,True
    elif IsCurveInCurve(Curve1,Curve2)==False and HasIntersect(Curve1,
Curve2)==False and HasIntersect(a, Curve2)==True:
        return False,True
    else:
        return False, False
# ok # Finding smallest Rectangular Units in a Curve and negative curve
def UnitRectangles (OrgCurve):

    Xcord = XYSetCurve(OrgCurve)[0]
    Ycord = XYSetCurve(OrgCurve)[1]
    #Creating smallest possible rectangles called Rectangular Unit
    RecPoints = []
    UnitRecList = []
    for m in range(len(Xcord) - 1):
        for n in range(len(Ycord) - 1):
            p01 = gh.ConstructPoint(Xcord[m],Ycord[n],0)
            p02 = gh.ConstructPoint(Xcord[m+1],Ycord[n],0)
            p03 = gh.ConstructPoint(Xcord[m+1],Ycord[n+1],0)
            p04 = gh.ConstructPoint(Xcord[m],Ycord[n+1],0)
            Rec = gh.x4PointSurface(p01,p02,p03,p04)
            #if IsCurveInCurve(Rec,OrgCurve):
            UnitRecList.append(Rec)
    return UnitRecList

# ok # Finding smallest Rectangular Units in a bunch of curves and
boundingbox curve
def UnitRectangles (Curves):

    Xcord = xySets(Curves)[0]
    Ycord = xySets(Curves)[1]
    #Creating smallest possible rectangles called Rectangular Unit
    RecPoints = []
    UnitRecList = []
    for m in range(len(Xcord) - 1):
        for n in range(len(Ycord) - 1):
            p01 = gh.ConstructPoint(Xcord[m],Ycord[n],0)
            p02 = gh.ConstructPoint(Xcord[m+1],Ycord[n],0)

```

```

    p03 = gh.ConstructPoint(Xcord[m+1],Ycord[n+1],0)
    p04 = gh.ConstructPoint(Xcord[m],Ycord[n+1],0)
    Rec = gh.x4PointSurface(p01,p02,p03,p04)
    #if IsCurveInCurve(Rec,OrgCurve):
        UnitReclList.append(Rec)
return UnitReclList

# ok # Negative part of a curve when surrounded in a smallest possible
rectangle.
def NegativeCurve(Curve):
    bRect = BoundingRect(Curve)
    negCurve = gh.RegionDifference(bRect,Curve)
    return negCurve

# ok # Bouding Rectangle of a Curve
def BoundingRect(Curve):

    bbXcord,bbYcord = XYSetCurve(Curve)
    xs = bbXcord[0]
    xe = bbXcord[-1]
    ys = bbYcord[0]
    ye = bbYcord[-1]
    p01 = gh.ConstructPoint(xs,ys,0)
    p02 = gh.ConstructPoint(xe,ys,0)
    p03 = gh.ConstructPoint(xe,ye,0)
    p04 = gh.ConstructPoint(xs,ye,0)
    return gh.x4PointSurface(p01,p02,p03,p04)

# ok # finding all rectangles possible with x , y coordination
def AllRectangles(OrgCurve):
    Xcord, Ycord = xySets([OrgCurve])
    xLen = len(Xcord)
    yLen = len(Ycord)
    Rectangles = []
    for a in range(xLen-1):
        for b in range (a+1,xLen):
            for c in range (yLen-1):
                for d in range (c+1,yLen):
                    corner=[]
                    p01 = gh.ConstructPoint(Xcord[a],Ycord[c],0)
                    p02 = gh.ConstructPoint(Xcord[b],Ycord[c],0)
                    p03 = gh.ConstructPoint (Xcord[b],Ycord[d],0)
                    p04 = gh.ConstructPoint(Xcord[a],Ycord[d],0)
                    rec = gh.x4PointSurface(p01,p02,p03,p04)
                    Rectangles.append(rec)
    return Rectangles

# ok # finding all rectangles possible with x , y coordination in a Curve
and voidcurves surrounded by it, this function ignores the curves are in
ignore curves such as columns
def allRectanglesOrgVoid(OrgCurve,VoidCurves):
    curves = [OrgCurve]
    if type(VoidCurves)==type([]):
        for vc in VoidCurves:

```

```

        curves.append(vc)
    else:
        curves.append(VoidCurves)
    Xcord, Ycord = xySets(curves)
    xLen = len(Xcord)
    yLen = len(Ycord)
    Rectangles = []
    for a in range(xLen-1):
        for b in range (a+1,xLen):
            for c in range (yLen-1):
                for d in range (c+1,yLen):
                    corner=[]
                    p01 = gh.ConstructPoint(Xcord[a],Ycord[c],0)
                    p02 = gh.ConstructPoint(Xcord[b],Ycord[c],0)
                    p03 = gh.ConstructPoint (Xcord[b],Ycord[d],0)
                    p04 = gh.ConstructPoint(Xcord[a],Ycord[d],0)
                    rec = gh.x4PointSurface(p01,p02,p03,p04)
                    Rectangles.append(rec)
    return Rectangles

# ok # Creating a n*2 Matrix of Rectangles and Area /// input: list of
# rectangles /// output: List of list of each rectangles and its area
# [rectangle, area]
def RectAreaMat(Curve):
    area = gh.Area(Curve)[0]
    mat = [Curve,area]
    return mat

# ok #
def BigBoy(Curves):
    RectAreaMat = []
    if not type(Curves) == "list":
        Curves = [Curves]
    for r in Curves:
        area = gh.Area(r)[0]
        mat = [r,area]
        RectAreaMat.append(mat)
    objects = RectAreaMat
    a = objects[0][1]
    r= None
    for obj in objects:
        if a <= obj[1]:
            a = obj[1]
            r = obj[0]
    return r

# finding the biggest curve by area
def BiggestCurve(curves):
    mat=[]
    if curves == []:
        return False
    if type(curves) != type([]):
        curves = [curves]
    crv = [i for i in curves]
    for c in crv:

```



```

    mat.append(RectAreaMat(c))
a = mat[0][1]
for obj in mat:
    if a <= obj[1]:
        a = obj[1]
        r = obj[0]
return r

# Check if a curve is rectangle or not
def IsRectangle(shape):
    if shape:
        if len(XYSetCurve(shape)[0])==2 and len(XYSetCurve(shape)[1])==2 :
            return True
    return False

# ok # divide a curve to bigboy and the rest shapes
def CurveDivision(OrgCurve,VoidCurve):
    allRect = allRectanglesOrgVoid(OrgCurve,VoidCurve)
    InOrgCrvRects = []
    for i in allRect:
        if IsCurveInCurve(i,OrgCurve):
            InOrgCrvRects.append(i)
    tshape = [i for i in InOrgCrvRects]
    fshape = []
    print len(InOrgCrvRects)
    for r in InOrgCrvRects:
        for v in VoidCurve:
            if HasIntersect(v,r) == True:
                fshape.append(r)
                if r in tshape:
                    tshape.remove(r)
    bigboy = BigBoy(tshape)
    restboys = gh.RegionDifference(OrgCurve,bigboy)
    return bigboy,restboys

def ReognOrgViod(Shape, ViodCurves):
    NewVoids = []
    newShape = Shape
    for vc in ViodCurves:
        if IsInandAligned(vc,Shape):
            newShape = gh.RegionDifference(Shape,vc)
        elif IsInandAligned(vc,Shape)[0] == True and
IsInandAligned(vc,Shape)[1] == False:
            NewVoids.append(vc)
        #else:
    return newShape,NewVoids

#**** finding boundary, void, columns and out curves = branch
# Check if a curve is in a bigger curve and have an aligned common edge,
then subtract the curves
def ExtractBranch(Curves,minArea):
    restCrv = []

```

```

group = []
ignorCrv = []
VoidCrv = []
bb01 = BiggestCurve(Curves)
for crv in Curves:
    if AreSimilar(bb01,crv)==False:
        restCrv.append(crv)
for rc in restCrv:
    if IsCurveInCurve(rc,bb01):
        if gh.Area(rc)[0] <= minArea:
            ignorCrv.append(rc)
        else:
            VoidCrv.append(rc)
    else:
        group.append(rc)
branch = [bb01, VoidCrv,ignorCrv]
return branch, group

def EnterBigboy(branch):
    SC_Boundary,SC_Viods,SC_Ignores = branch
    ##### find the rectangles are surraounded by boundary curve and don't
    have intesect with void curves
    SC_allRect = allRectanglesOrgVoid(SC_Boundary,SC_Viods)
    inBoundRect = []
    for i in SC_allRect:
        if IsCurveInCurve(i,SC_Boundary):
            inBoundRect.append(i)
    #print(len(inBoundRect))
    inBoundNotIntersect = [i for i in inBoundRect]
    for j in inBoundRect:
        for k in SC_Viods:
            if (HasIntersect(j,k) or IsCurveInCurve(j,k) or
IsCurveInCurve(k,j) ) and j in inBoundNotIntersect :
                inBoundNotIntersect.remove(j)
    print(len(inBoundNotIntersect))
    ##### find the bigboy rectangle
    SC_bigboy = BiggestCurve(inBoundNotIntersect)

    ##### finding the rest shape after subtracting bigboy from boundary
    curve
    SC_RestShapes = gh.RegionDifference(SC_Boundary,SC_bigboy)

    ##### creating new group of curvs after finding the bigboy and rest
    shapes
    void = SC_Viods[:]
    ignore = SC_Ignores[:]
    group = void + ignore
    if type(SC_RestShapes) != type([]):
        SC_RestShapes=[SC_RestShapes]
    for rc in SC_RestShapes:
        group.append(rc)

    return SC_bigboy,group

```

```

def EvaluateBoundary(branch):
    SC_Boundary, SC_Viods, SC_Ignores = branch
    if IsRectangle(SC_Boundary) and len(SC_Viods) == 0 :
        bigboy = SC_Boundary
        group = SC_Viods + SC_Ignores
        newbranch = ReOrganize(group,minArea)
        return newbranch, bigboy
    return branch,None

def IsBoundaryBigboy(branch):
    SC_Boundary, SC_Viods, SC_Ignores = branch
    if IsRectangle(SC_Boundary) and len(SC_Viods) == 0:
        return True
    return False

def MakeGroup(branch):
    SC_Boundary, SC_Viods, SC_Ignores = branch
    void = SC_Viods[:]
    ignore = SC_Ignores[:]
    group = void + ignore
    group.append(SC_Boundary)
    return group
"""

"""
Voids = ExtractBranch(Sample_Curves, minArea)[0][1]
Ignored_Curves = ExtractBranch(Sample_Curves, minArea)[0][2]
Main_Boundary = ExtractBranch(Sample_Curves, minArea)[0][0]

i = 0
Groups = []
BigBoys = []
while len(Sample_Curves) > 0 :
    i +=1
    branch , group0 = ExtractBranch(Sample_Curves, minArea)
    cl_branch = CleanBranch(branch)
    Groups.append(group0)
    if IsBoundaryBigboy(cl_branch):
        BigBoys.append(cl_branch[0])
        group1=[]
    else:
        bigboy,group1 = EnterBigboy(cl_branch)
        BigBoys.append(bigboy)
    Sample_Curves = group0 + group1
    print i

ignore = [i for i in Ignored_Curves]
for i in ignore:
    for b in BigBoys:
        if AreSimilar(i,b) and b in BigBoys:
            BigBoys.remove(b)

```

```
#---- Cleaning BigBoys list: check if they are similar to voids.
voids_backup = [v for v in Voids]
for i in voids_backup:
    for b in BigBoys:
        if (AreSimilar(i,b) or IsCurveInCurve(b,i) or IsCurveInCurve(i,b))
and b in BigBoys:
        BigBoys.remove(b)

#---- Finding negative curve of boundary
Negative_Curve = NegativeCurve(Main_Boundary)
```

Joining Algorithm codes in Python

```

__author__ = "parsa"
__version__ = "2021.07.20"

import rhinoscriptsyntax as rs
import ghpythonlib.components as gh
import Grasshopper.Documentation
import math as math
Decimal = 6
a = 0
# ok # check if two are completely similar in shape and location
(duplicated curve)
def AreSimilar(a,b):
    if IsCurveInCurve(a,b)==True and IsCurveInCurve(b,a) == True:
        return True
    return False

# ok #Check if a Curve is surrounded in another Curve
#def IsCurveInCurve(inCurve,outCurve):
#    int = gh.RegionDifference(inCurve,outCurve)
#    if int == None:
#        return True
#    return False
def IsCurveInCurve(inc,outc):
    if inc == None or outc == None:
        return False
    unc= gh.RegionUnion([inc,outc])
    if type(unc)==type([]) or unc==None:
        return False
    else:
        uncAr= round(gh.Area(unc)[0],4)
        incAr = round(gh.Area(inc)[0],4)
        outcAr = round(gh.Area(outc)[0],4)
        if uncAr == outcAr:
            return True
        if uncAr != outcAr:
            return False

# ok # check if two curves has intersection or not.
def HasIntersect(curve,refCurve):
    result = gh.RegionIntersection(curve,refCurve)
    if result == None:
        return False
    return True

# ok # Bouding Rectangle of a Curve
def BoundingRect(Curve):

    bbXcord,bbYcord = XYSetCurve(Curve)
    xs = bbXcord[0]
    xe = bbXcord[-1]
    ys = bbYcord[0]
    ye = bbYcord[-1]

```

```

p01 = gh.ConstructPoint(xs,ys,0)
p02 = gh.ConstructPoint(xe,ys,0)
p03 = gh.ConstructPoint(xe,ye,0)
p04 = gh.ConstructPoint(xs,ye,0)
return gh.x4PointSurface(p01,p02,p03,p04)

```

```

def XYSetCurve(Curve):
    while type(Curve) == type([]):
        Curve = Curve[0]
    points = []
    points = gh.Explode(Curve,True)[1]
    #points.pop(-1)
    xpoint=[]
    ypoint=[]
    zpoint=[]
    for point in points:
        f = gh.Deconstruct(point)
        i = round(f[0],Decimal)
        j = round(f[1],Decimal)
        xpoint.append(i)
        ypoint.append(j)
    #Sorting and removing duplicated values of X and Y lists
    X=set(xpoint)
    Y=set(ypoint)
    Xcord = sorted(list(X))
    Ycord = sorted(list(Y))
    return (Xcord,Ycord)

```

```

def RectangleProp(Rectangle):
    x,y = XYSetCurve(Rectangle)
    Dx = abs(x[0] - x[1])
    Dy = abs(y[0] - y[1])
    area = Dx * Dy
    return [x[0],x[1],y[0],y[1],Dx,Dy,area]

```

```

def Find_TadBoys(rectangles):
    Real_BigBoys = []
    TadBoy_rects = []

    for r in rectangles:
        rect_prop = RectangleProp(r)
        if rect_prop[4] < Minimum_Edge_Size or rect_prop[5] <
Minimum_Edge_Size:
            TadBoy_rects.append(r)
        else:
            Real_BigBoys.append(r)

    return Real_BigBoys,TadBoy_rects

```

```

*** to find rectangles are likely neighbour with the given rectangle.
def Tad_neighbors(tadboy, BigBoys):

```

```

Top_neighbor = None
Right_neighbor = None
Bottom_neighbor = None
Left_neighbor = None
Neighborhood=[]
tadboy_prop = RectangleProp(tadboy)
for bg in BigBoys:
    bg_prop = RectangleProp(bg)
    if bg_prop[0] == tadboy_prop[1] or bg_prop[1] == tadboy_prop[0] or
bg_prop[2] == tadboy_prop[3] or bg_prop[3] == tadboy_prop[2]:
        Neighborhood.append(bg)

for nr in Neighborhood:
    nr_prop = RectangleProp(nr)
    if nr_prop[2] == tadboy_prop[3] and nr_prop[0] <= tadboy_prop[0]
and nr_prop[1] >= tadboy_prop[1]:
        Top_neighbor = nr
    else:
        if nr_prop[0] == tadboy_prop[1] and nr_prop[2] <=
tadboy_prop[2] and nr_prop[3] >= tadboy_prop[3]:
            Right_neighbor = nr
        else:
            if nr_prop[3] == tadboy_prop[2] and nr_prop[0] <=
tadboy_prop[0] and nr_prop[1] >= tadboy_prop[1]:
                Bottom_neighbor = nr
            else:
                if nr_prop[1] == tadboy_prop[0] and nr_prop[2] <=
tadboy_prop[2] and nr_prop[3] >= tadboy_prop[3]:
                    Left_neighbor = nr
    return
tadboy,Top_neighbor,Right_neighbor,Bottom_neighbor,Left_neighbor

def Adoption(Neighborhood):
    tadboy = Neighborhood[0]
    topNeib = Neighborhood[1]
    rightNeib = Neighborhood[2]
    bottomNeib = Neighborhood[3]
    leftNeib = Neighborhood[4]
    dx = RectangleProp(tadboy)[4]
    dy = RectangleProp(tadboy)[5]

    if topNeib:
        top_union = gh.RegionUnion([tadboy,topNeib])
        top_junction = ["Top", tadboy,topNeib,top_union]
    else: top_junction = ["Top",None]

    if rightNeib:
        right_union = gh.RegionUnion([tadboy,rightNeib])
        right_junction = ["Right",tadboy,rightNeib,right_union]
    else: right_junction = ["Right",None]

    if bottomNeib:
        bottom_union = gh.RegionUnion([tadboy,bottomNeib])
        bottom_junction = ["Bottom",tadboy,bottomNeib,bottom_union]

```

```

else: bottom_junction = ["Bottom",None]

if leftNeib:
    left_union = gh.RegionUnion([tadboy,leftNeib])
    left_junction = ["Left",tadboy,leftNeib,left_union]
else: left_junction = ["Left",None]

return top_junction, right_junction, bottom_junction, left_junction

def EvJunctionDirection(junction):
    if not None in junction :
        Direction,tadboy, bigboy, union = junction
        bound_rect = BoundingRect(union)

        it,jt,mt,nt,xt,yt,areat = RectangleProp(tadboy)
        ib,jb,mb,nb,xb,yb,areab = RectangleProp(bigboy)

        if xt/yt == 1:
            result = 1
        if nt == mb or mt == nb:
            if xt/yt > 1:
                result = 2
            elif xt/yt < 1 and xt < Minimum_Panel:
                result = -10
            else: result = 0

        if it == jb or ib == jt:
            if xt/yt < 1:
                result = 2
            elif xt/yt > 1 and yt < Minimum_Panel:
                result = -10
            else: result = 0
        return result
    return None

def EvJunctionCorners(junction):
    if not None in junction :
        Direction,tadboy, bigboy, union = junction
        x,y = XYSetCurve(union)
        if len(x) + len(y) == 4:
            result = 10
        if len(x) + len(y) == 6:
            result = 1
        if len(x) + len(y) > 6:
            result = 0
        return result
    return None

def EvAreaRate(junction):
    if not None in junction :
        Direction,tadboy, bigboy, union = junction
        r = gh.Area(tadboy)[0]/gh.Area(bigboy)[0]
        return r
    return None

```



```

def EvaluateJunction(junction):
    Direction = junction[0]
    total = None
    Dir = EvJunctionDirection(junction)
    Cor = EvJunctionCorners(junction)
    Ar = EvAreaRate(junction)
    if not None in junction:
        total = Dir+Cor+Ar
    return Direction,Dir , Cor , Ar,total

def Eva_Result(Evaluations):
    result = Evaluations[0]
    for eva in Evaluations:
        if type(eva[4]) == type(1.23456) and eva[4] < 0:
            return False
        if eva[4] > result[4]:
            result = eva
    if None in result:
        return False
    return result[0]

**** finding TadBoys and their Neighborhoods
Neighborhoods = []
Real_Bigboys,TadBoys = Find_TadBoys(BigBoys)
for tady in TadBoys:
    neighbors = Tad_neighbors(tady,Real_Bigboys)
    Neighborhoods.append(neighbors)

if len(TadBoys)>0:
    Neighborhood = Neighborhoods[a]
    Tadboy = TadBoys[a]

**** Making Junction and Evaluate them
Scores_list = []
Adoptoin_list = []
for nei in Neighborhoods:
    adoptions = Adoption(nei)
    Scores = []
    for junc in adoptions:
        Ev = EvaluateJunction(junc)
        Scores.append(Ev)
    Scores_list.append(Scores)
    Adoptoin_list.append(adoptions)

if len(TadBoys)>0:
    junc_score = Scores_list[a]
#    adoption = Adoptoin_list[1][1]

**** finding the best junction for each tadboy and finding the related
United shape
results = []

```

```
if len(TadBoys)>0:
    scores = Scores_list[a]

for scores in Scores_list:
    n = Eva_Result(scores)
    results.append(n)

**** Finding tadboys that can not be adopted
Orphans_index=[]
Final_Junctions=[]
for index in range(len(results)):
    print results[index]
    junction = Adoptoin_list[index]
    if results[index]==False or results[index]=="impossible":
        orphanboy_index = index
        Orphans_index.append(orphanboy_index)

    for j in junction:
        if j[0] == results[index]:
            Final_junction = j
            Final_Junctions.append(Final_junction)
OrphanBoys = []
for i in Orphans_index:
    OrphanBoys.append(TadBoys[i])
**** finding the selectet junctions and cleaning them because
**** some have a same bigboys. in such cases, we union them.
Families=[]

Final_Junction3=[]

for i in Final_Junctions:
    Final_Junction3.append(i[3])
    Families.append(i[3])

step_counter = 0
for i, curve_a in enumerate(Families):

    if step_counter == k:
        import sys
        break
    step_counter += 1
    for j, curve_b in enumerate(Families):

        if j <= i : # j = 1 and i = 1 continue
            continue

        # i = 0 and j = 1

        if curve_a!=None and curve_b!=None and HasIntersect(curve_a,
curve_b):

            Families[i] = gh.RegionUnion([Families[i], curve_b])
            print(i, j)
            Families[j] = None
```

```
# finding single bigboys:
Single_BigBoys = [bb for bb in Real_BigBoys]
for b in Real_BigBoys:
    for u in Families:
        if (IsCurveInCurve(b,u) == True or HasIntersect(b,u)==True) and b
in Single_BigBoys:
            Single_BigBoys.remove(b)

Panelized_Shapes = []
#for i in OrphanBoys:
#    Panelized_Shapes.append(i)
for j in Single_BigBoys:
    Panelized_Shapes.append(j)
for k in Families:
    if k != None:
        Panelized_Shapes.append(k)
```

Panelling Algorithm codes in Python

```
__author__ = "parsa"
__version__ = "2021.07.20"

import rhinoscriptsyntax as rs
import ghpythonlib.components as gh
import Grasshopper.Documentation
import math as math
Decimal = 6
#a = 0
# ok # check if two are completely similar in shape and location
(duplicated curve)
def AreSimilar(a,b):
    if IsCurveInCurve(a,b)==True and IsCurveInCurve(b,a) == True:
        return True
    return False

# ok #Check if a Curve is surrounded in another Curve
def IsCurveInCurve(inCurve,outCurve):
    int = gh.RegionDifference(inCurve,outCurve)
    if int == None:
        return True
    return False

# ok # check if two curves has intersection or not.
def HasIntersect(curve,refCurve):
    result = gh.RegionIntersection(curve,refCurve)
    if result == None:
        return False
    return True

# ok # Bouding Rectangle of a Curve
def BoundingRect(Curve):

    bbXcord,bbYcord = XYSetCurve(Curve)
    xs = bbXcord[0]
    xe = bbXcord[-1]
    ys = bbYcord[0]
    ye = bbYcord[-1]
    p01 = gh.ConstructPoint(xs,ys,0)
    p02 = gh.ConstructPoint(xe,ys,0)
    p03 = gh.ConstructPoint(xe,ye,0)
    p04 = gh.ConstructPoint(xs,ye,0)
    return gh.x4PointSurface(p01,p02,p03,p04)

def XYSetCurve(Curve):
    while type(Curve) == type([]):
        Curve = Curve[0]
```

```

points = []
points = gh.Explode(Curve, True)[1]
#points.pop(-1)
xpoint=[]
ypoint=[]
zpoint=[]
for point in points:
    f = gh.Deconstruct(point)
    i = round(f[0],Decimal)
    j = round(f[1],Decimal)
    xpoint.append(i)
    ypoint.append(j)
#Sorting and removing duplicated values of X and Y lists
X=set(xpoint)
Y=set(ypoint)
Xcord = sorted(list(X))
Ycord = sorted(list(Y))
return (Xcord,Ycord)

def RectangleProp(Rectangle):
    x,y = XYSetCurve(Rectangle)
    Dx = abs(x[0] - x[1])
    Dy = abs(y[0] - y[1])
    area = Dx * Dy
    return [x[0],x[1],y[0],y[1],Dx,Dy,area]

# finding the biggest curve by area
def BiggestCurve(curves):
    mat=[]
    if curves == []:
        return False
    if type(curves) != type([]):
        curves = [curves]
    crv = [i for i in curves]
    for c in crv:
        mat.append(RectAreaMat(c))
    a = mat[0][1]
    for obj in mat:
        if a <= obj[1]:
            a = obj[1]
            r = obj[0]
    return r
# ok # Creating a n*2 Matrix of Rectangles and Area /// input: list of
rectangles /// output: List of list of each rectangles and its area
[rectangle, area]
def RectAreaMat(Curve):
    area = gh.Area(Curve)[0]
    mat = [Curve,area]
    return mat

# Check if a curve is rectangle or not
def IsRectangle(shape):

```

```

if shape:
    if len(XYSetCurve(shape)[0])==2 and len(XYSetCurve(shape)[1])==2 :
        return True
return False

def XYSetCurve(Curve):
    while type(Curve) == type([]):
        Curve = Curve[0]
    points = []
    points = gh.Explode(Curve,True)[1]
    #points.pop(-1)
    xpoint=[]
    ypoint=[]
    zpoint=[]
    for point in points:
        f = gh.Deconstruct(point)
        i = round(f[0],Decimal)
        j = round(f[1],Decimal)
        xpoint.append(i)
        ypoint.append(j)
    #Sorting and removing duplicated values of X and Y lists
    X=set(xpoint)
    Y=set(ypoint)
    Xcord = sorted(list(X))
    Ycord = sorted(list(Y))
    return (Xcord,Ycord)

# organise the panel shapes in three groups: Rectangulare (4 corners), L
shape (6 corners), T shape (8 corners), and S shape (more than 8 corners)
def Organise_Shapes( curves):
    if len( curves) > 0:
        edge_list = []
        corner_list = []
        for crv in curves:
            edge = len(gh.Explode(crv,False)[0])
            corner = len(gh.Explode(crv,False)[1])
            edge_list.append(edge)
            corner_list.append(corner)
        return edge_list,corner_list
    return False

# find the bounding rectangle of shape and divide the edges by panel size
def Panel_Number( curve, a , ob,sb, MinDim):
    Bound_crv = BoundingRect( curve)
    Bound_crv_prop = RectangleProp( Bound_crv)
    Dx = Bound_crv_prop[4]
    Dy = Bound_crv_prop[5]
    if Dx % a <MinDim:
        No_X_a = floor(Dx/a)-1
    else: No_X_a = floor(Dx/a)
    if Dx % b <MinDim:
        No_X_b = floor(Dx/b)-1
    else: No_X_b = floor(Dx/b)

```

```

# ok # finding all rectangles possible with x , y coordination in a family
def AllRectangles_Family(family):
    Xcord,Ycord = XYSetCurve(family)
    xLen = len(Xcord)
    yLen = len(Ycord)
    Rectangles = []
    for a in range(xLen-1):
        for b in range (a+1,xLen):
            for c in range (yLen-1):
                for d in range (c+1,yLen):
                    corner=[]
                    p01 = gh.ConstructPoint(Xcord[a],Ycord[c],0)
                    p02 = gh.ConstructPoint(Xcord[b],Ycord[c],0)
                    p03 = gh.ConstructPoint (Xcord[b],Ycord[d],0)
                    p04 = gh.ConstructPoint(Xcord[a],Ycord[d],0)
                    rec = gh.x4PointSurface(p01,p02,p03,p04)
                    Rectangles.append(rec)
    return Rectangles

def FamilyDivision(family,Real_Bigboys):
    for b in Real_Bigboys:
        if IsCurveInCurve(b,family):
            BigBoy = b
            rest_shape = gh.RegionDifference(family,BigBoy)
            if type(rest_shape) != type([]):
                rest_shape = [rest_shape]
    childs = []
    twain = []
    for r in rest_shape:
        if IsRectangle(r):
            childs.append(r)
        else:
            twain.append(r)

    return BigBoy,rest_shape
def Division_A(edge,a,Panel_Min):
    if edge <= a and edge >= Panel_Min :
        N_reg_a = 1
        D_reg_a = edge
        N_ireg_a = 0
        D_ireg_a = 0
    if edge > a and (edge % a) > Panel_Min :
        N_reg_a = math.floor(edge/a)
        D_reg_a = a
        N_ireg_a = 1
        D_ireg_a = edge % a
    if edge > a and (edge % a) <= Panel_Min :
        N_reg_a = math.floor(edge/a)-1
        D_reg_a = a
        N_ireg_a = 2
        D_ireg_a = abs(edge - (N_reg_a*a))/2
    return [N_reg_a, D_reg_a, N_ireg_a, D_ireg_a]

def Division_B(edge, ob, mb, Panel_Min):

```

```

if edge >= Panel_Min and edge <= ob:
    N_reg_b = 1
    D_reg_b = edge
if edge >= ob and edge <= mb :
    N_reg_b = 1
    D_reg_b = edge
if edge > mb :
    N_reg_b = math.floor(edge/ob)
    D_reg_b = edge/N_reg_b
return [N_reg_b, D_reg_b]

```

```

def Evaluate_DirectionX(rectangle, a, ob, mb, Panel_Min):
    edge01 = RectangleProp(rectangle)[4]
    edge02 = RectangleProp(rectangle)[5]

    m01 = Division_A(edge01,a,Panel_Min)[0]
    m02 = Division_A(edge01,a,Panel_Min)[2]
    m03 = Division_B(edge02, ob, mb, Panel_Min)[0]
    P_X_Nu = (m01 + m02)*m03

    return P_X_Nu

```

```

def Evaluate_DirectionY(rectangle, a, ob, mb, Panel_Min):
    edge01 = RectangleProp(rectangle)[4]
    edge02 = RectangleProp(rectangle)[5]

    m01 = Division_A(edge02,a,Panel_Min)[0]
    m02 = Division_A(edge02,a,Panel_Min)[2]
    m03 = Division_B(edge01, ob, mb, Panel_Min)[0]
    P_Y_Nu = (m01 + m02)*m03
    return P_Y_Nu

```

```

def Select_Direction(rectangle, a, ob, mb, Panel_Min):
    edge01 = RectangleProp(rectangle)[4]
    edge02 = RectangleProp(rectangle)[5]
    XD = Evaluate_DirectionX(rectangle, a, ob, mb, Panel_Min)
    YD = Evaluate_DirectionY(rectangle, a, ob, mb, Panel_Min)
    if XD <= YD:
        SD = XD
        XDiv = Division_A(edge01,a,Panel_Min)
        YDiv = Division_B(edge02, ob, mb, Panel_Min)
    if XD > YD:
        SD = YD
        YDiv = Division_A(edge02,a,Panel_Min)
        XDiv = Division_B(edge01, ob, mb, Panel_Min)
    else: SD = None
    return XDiv, YDiv

```

```

def XY_Coordinate(rectangle, a, ob, mb, Panel_Min):
    x1 , y1 = Select_Direction(rectangle, a, ob, mb, Panel_Min)
    x0 = RectangleProp(rectangle)[0]
    y0 = RectangleProp(rectangle)[2]
    if len(x1) == 4 and len(y1) ==2:

```



```

Xcord = [x0]
Ycord = [y0]
for i in range(int(x1[0])):
    x0 += x1[1]
    Xcord.append(x0)
for j in range(x1[2]):
    x0 += x1[3]
    Xcord.append(x0)
for k in range(int(y1[0])):
    y0 += y1[1]
    Ycord.append(y0)
if len(x1) == 2 and len(y1) == 4:
    Xcord = [x0]
    Ycord = [y0]
    for i in range(int(y1[0])):
        y0 += y1[1]
        Ycord.append(y0)
    for j in range(int(y1[2])):
        y0 += y1[3]
        Ycord.append(y0)
    for k in range(int(x1[0])):
        x0 += x1[1]
        Xcord.append(x0)
return Xcord, Ycord

```

```

def Create_Panels(rectangle, a, ob, mb, Panel_Min):
    Xcord,Ycord = XY_Coordinate(rectangle, a, ob, mb, Panel_Min)
    RecPoints = []
    UnitRecList = []
    for m in range(len(Xcord) - 1):
        for n in range(len(Ycord) - 1):
            p01 = gh.ConstructPoint(Xcord[m],Ycord[n],0)
            p02 = gh.ConstructPoint(Xcord[m+1],Ycord[n],0)
            p03 = gh.ConstructPoint(Xcord[m+1],Ycord[n+1],0)
            p04 = gh.ConstructPoint(Xcord[m],Ycord[n+1],0)
            Rec = gh.x4PointSurface(p01,p02,p03,p04)
            #if IsCurveInCurve(Rec,OrgCurve):
            UnitRecList.append(Rec)
    return UnitRecList

```

```

def Generate_Panels (curve, a, ob, mb, Panel_Min):
    rectangle = BoundingRect(curve)
    panels = Create_Panels(rectangle, a, ob, mb, Panel_Min)
    Final_Panels = []
    for p in panels:
        final_panel = gh.RegionIntersection(p , curve)
        Final_Panels.append(final_panel)
    return Final_Panels

```

```
Final_Paneling = []
```

```
#Final_Paneling = Generate_Panels (Shape_to_Panels[mm], a, ob, mb,  
Panel_Min)  
for shape in Shape_to_Panels:  
    panels = Generate_Panels (shape, a, ob, mb, Panel_Min)  
    Final_Paneling += panels
```

APPENDIX 2: HIGH QUALITY FIGURES

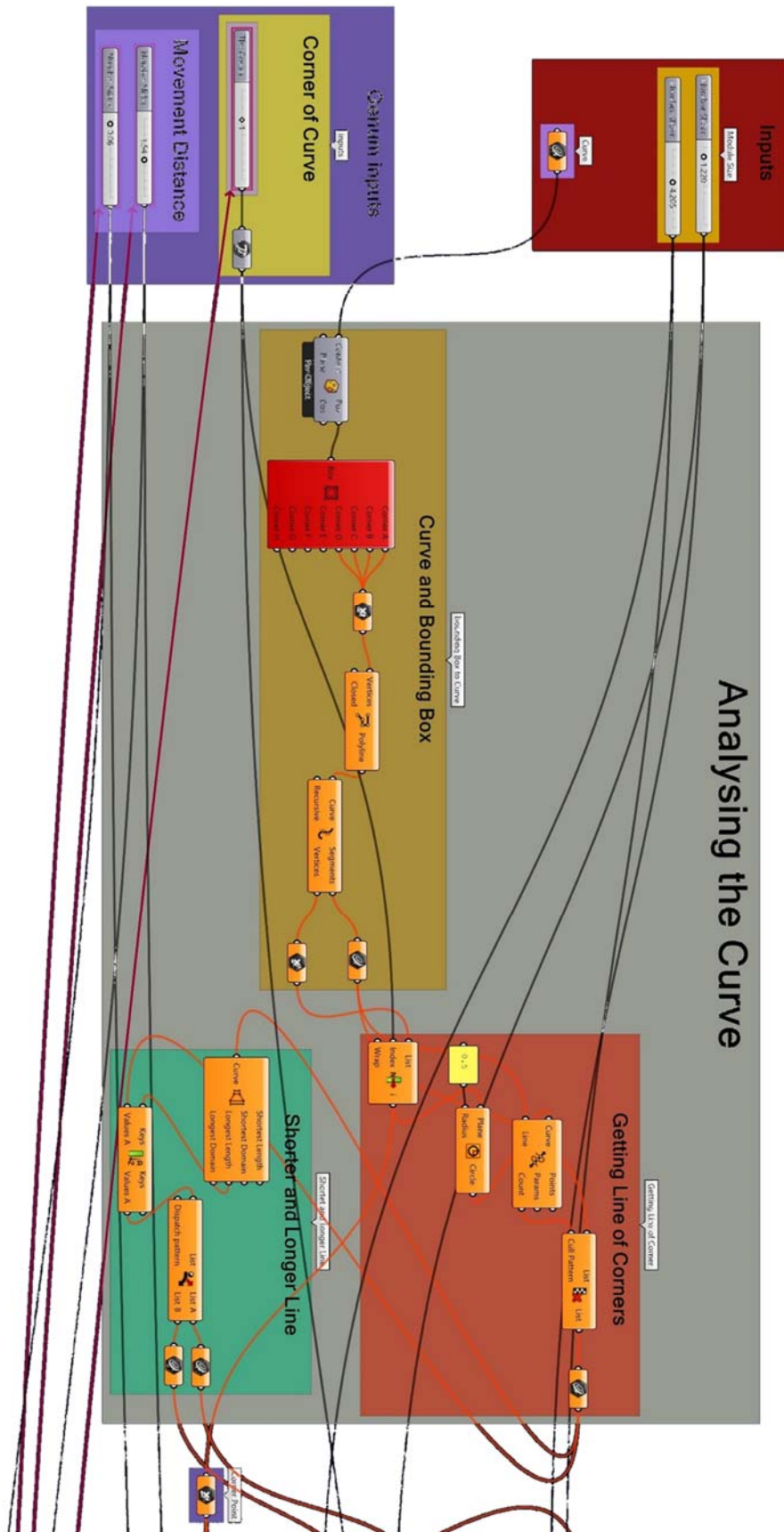


Figure 13 – Grasshopper script, Analysing part

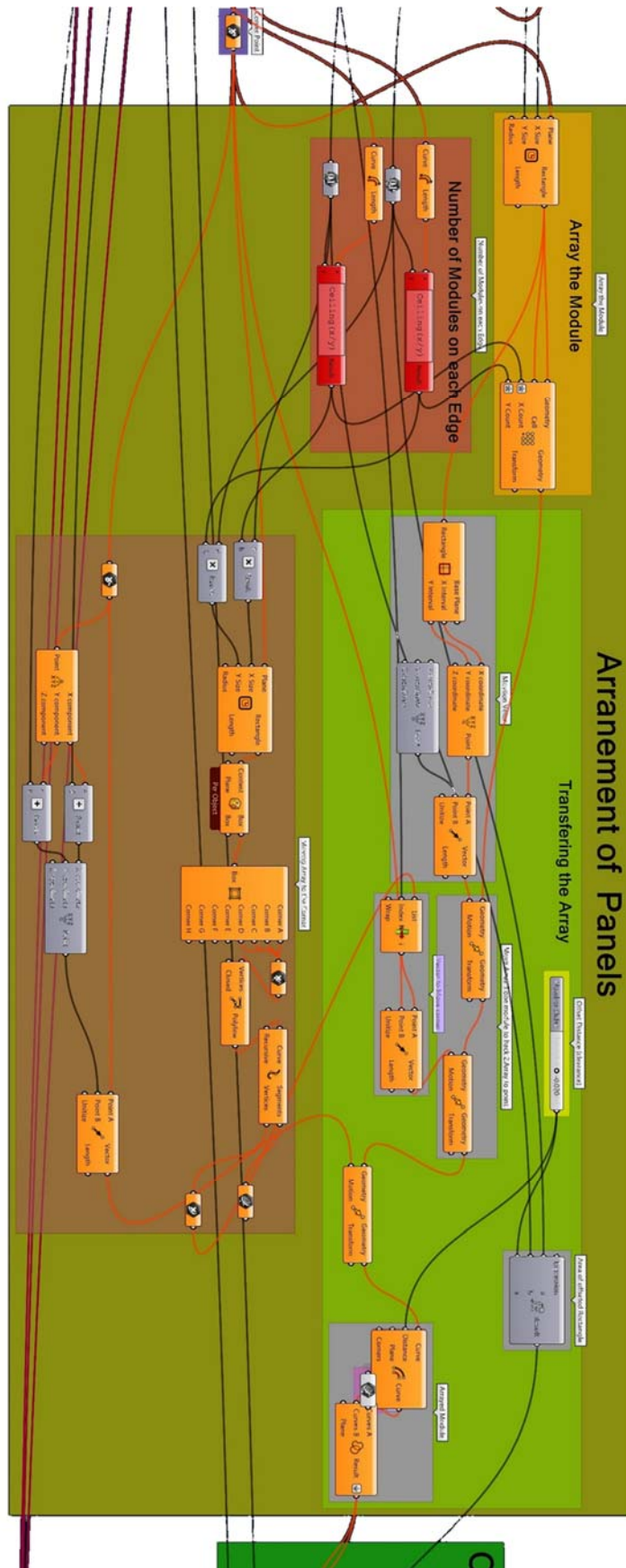


Figure 14 – Grasshopper script, Panelling part

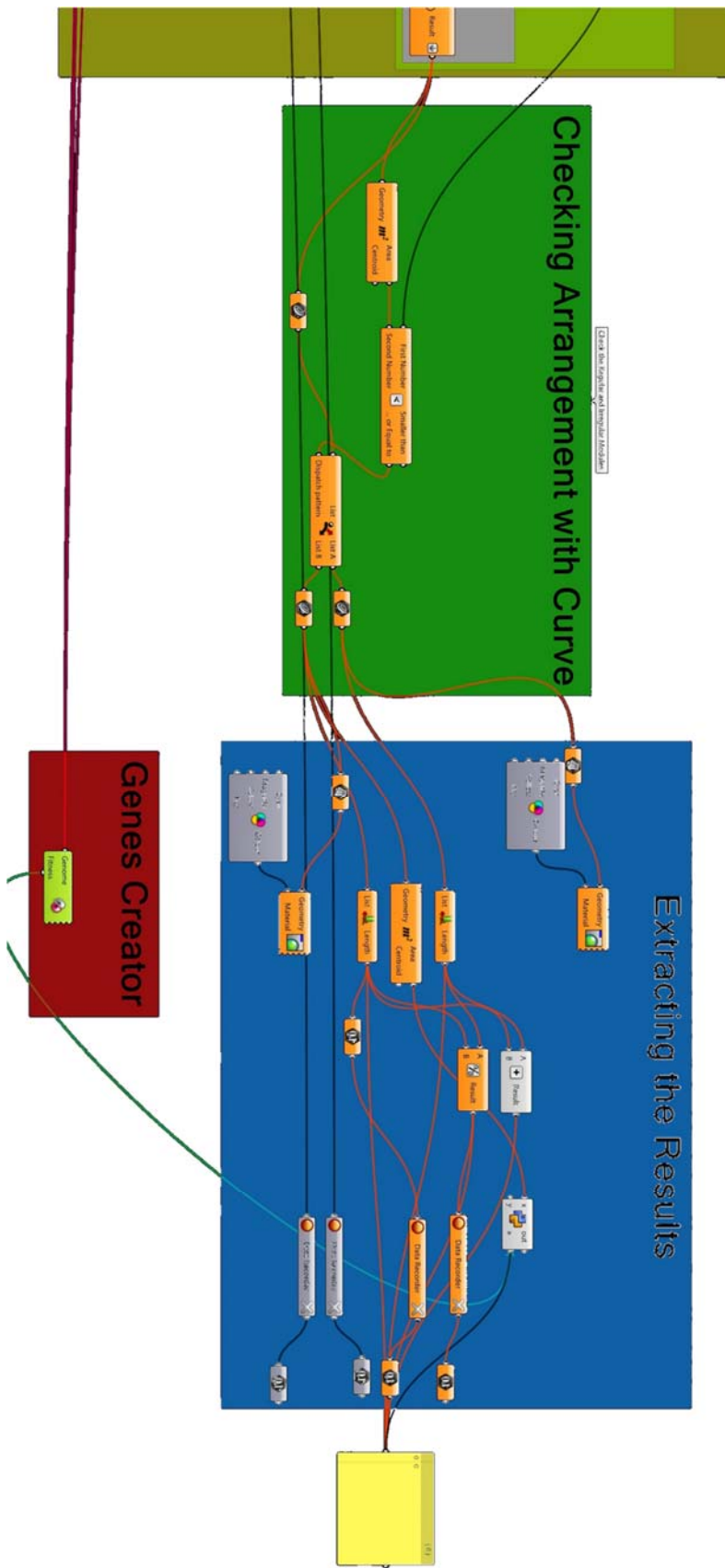


Figure 15 – Grasshopper script, Optimization part

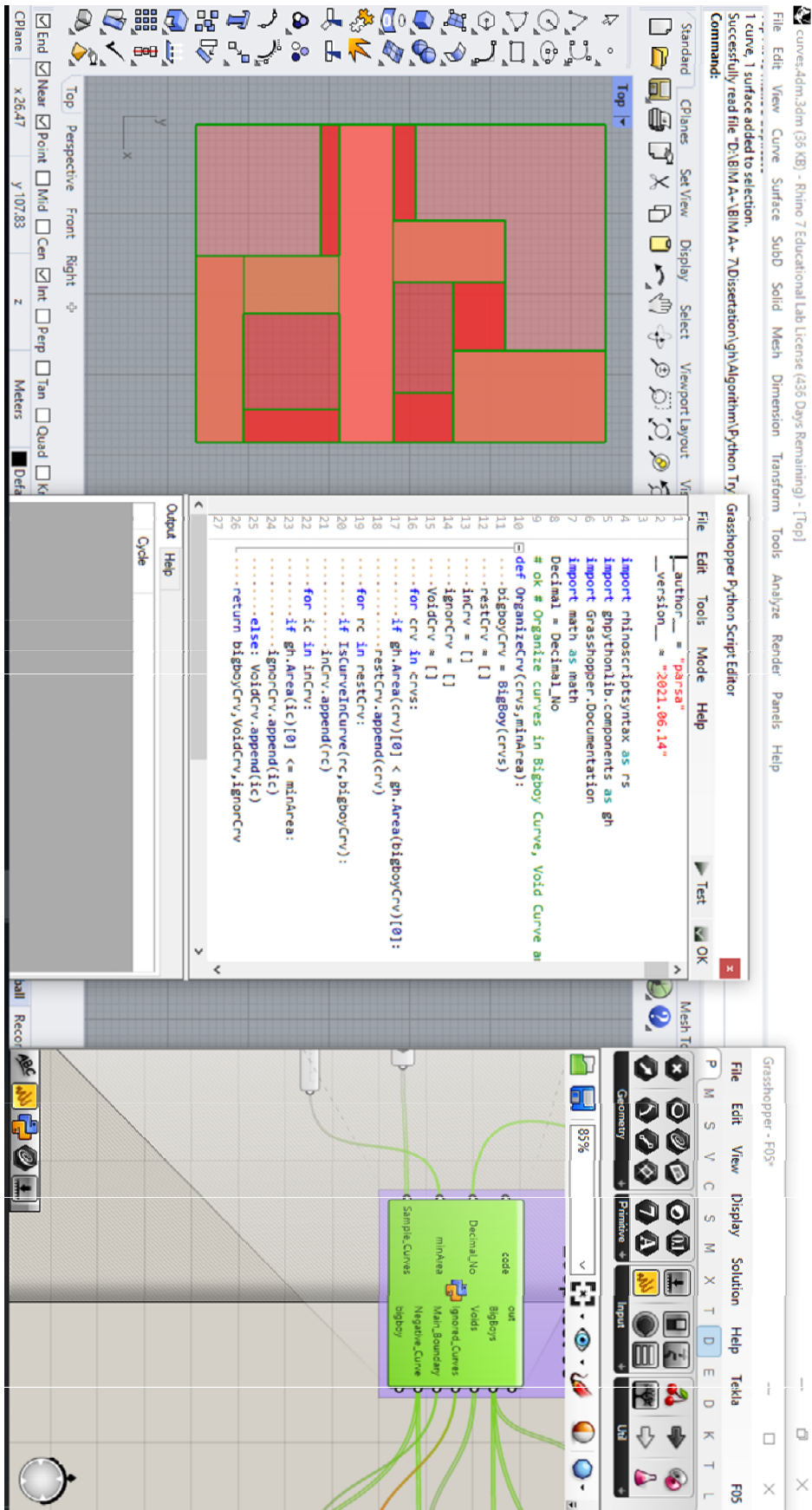


Figure 18 – A sample result of Rational Optimisation by Python in Grasshopper