



YAŞAR UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MASTER THESIS



**EARLY DESIGN STAGE MULTI LEVEL
ARCHITECTURAL FLOOR PLAN ORGANIZATION
BY USING GENETIC ALGORITHMS**

ERİNÇ YILDIRIM

THESIS ADVISOR: ASSOC. PROF. BAŞAK KUNDAKÇI KOYUNBABA
CO-ADVISOR: ASSOC. PROF. İPEK GÜRSEL DİNO

ARCHITECTURE

PRESENTATION DATE: 17.07.2019

BORNOVA / İZMİR
AUGUST 2019

We certify that, as the jury, we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Jury Members:

Signature:

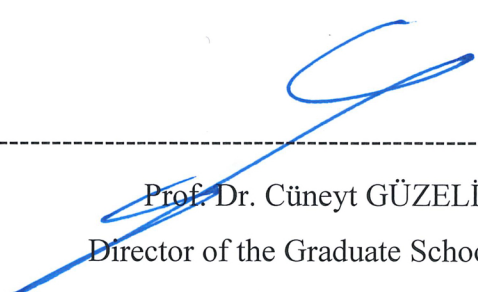
Assoc. Prof. Dr. Başak KUNDAKCI
KOYUNBABA
Yaşar University



Assoc. Prof. Dr. Tutku Didem ALTUN
Dokuz Eylül University



Asst. Prof. Dr. Ebru ALAKAVUK
Yaşar University



Prof. Dr. Cüneyt GÜZELİŞ
Director of the Graduate School

ABSTRACT

EARLY DESIGN STAGE MULTI LEVEL ARCHITECTURAL FLOOR PLAN ORGANIZATION BY USING GENETIC ALGORITHMS

Yıldırım, Erinç

Msc, Architecture

Advisor: Assoc. Prof. Başak Kundakçı Koyunbaba

Co-Advisor: Assoc. Prof. İpek Gürsel Dino

JULY 2019

Architectural layout design entails numerous decision variables, which interact in complex ways, and as such it is defined by combinatorial complexity. To deal with this complexity, computational optimization methods can be utilized. The aim of the research is to find a feasible set of architectural layout design alternatives for three objectives, which are minimization of the relations weighted circulation between spaces and minimization of the total solar radiation of spatial organization, subject to several constraints. The multi objective optimization problem is formulated where the constraints are real parameters. Multi objective evolutionary algorithms are utilized in order to gather a set of non-dominated solutions.

Key Words: architectural layout; multi-storey; circulation; multi-objective optimization; evolutionary algorithms

ÖZ

ERKEN TASARIM EVRESİ GENETİK ALGORİTMALAR KULLANILARAK ÇOK KATLI MİMARİ KAT PLANI ORGANİZASYONU

Yıldırım, Erinç

Yüksek Lisans, Mimarlık

Danışman: Doç. Dr. Başak Kundakçı Koyunbaba

Yardımcı Danışman: Doç. Dr. İpek Gürsel Dino

TEMMUZ 2019

Mimari plan tasarımı, karmaşık şekillerde etkileşime giren sayısız karar değişkenini içerir ve bu şekilde birleşimsel karmaşıklıkla tanımlanır. Bu karmaşıklıkla başa çıkmak için, hesaplama optimizasyon yöntemleri kullanılabilir. Araştırmanın amacı, çeşitli kısıtlamalara tabi olarak, mekansal organizasyonun toplam güneş ışığının azaltılması ve mekânlar arasındaki ilişkilerin ağırlıklı dolaşımının en aza indirilmesi olan iki amaç için uygun bir mimari yerleşim tasarımı alternatifleri kümesi bulmaktır. Çok amaçlı optimizasyon problemi, tam sayı parametreler olarak kurgulanmıştır. Baskın olmayan bir dizi çözümü toplamak için çok amaçlı evrimsel algoritmalar kullanılmıştır.

Anahtar Kelimeler: mimari plan, çok katlı, dolaşım, çok amaçlı optimizasyon, evrimsel algoritmalar

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Assoc. Prof. Bařak Kundakcı Koyunbaba and my co-advisor Assoc. Prof. İpek Gürsel Dino for their guidance, patience and support during this study. Also, I would like to express my deepest gratitude to Prof. Dr. İ. Sevil Sarıyıldız and Prof. Dr. Mehmet Fatih Tařgtiren

I would like to express my enduring love to my family, my father Aydın Yıldırım and my mother Ayře Gün Yıldırım have always supported and loved me in my life. I appreciate their endless trust with my decisions.

Additionally, my sincere thanks go to my companion Ayřeğöl Öykü Görğün and her family because of their support and kindness.

I thank my colleagues Berfin Yıldız, Cemre Çubukçuođlu, Duhan Ölmez, Ece Buldan, Feyza Durmuřlar, Fulya Özbey, İrem Deniz Akçam, Müge Sever, Selin Güngör, Selin Karagözler Gülerođlu in Yařar University also, for a year they have understand and supported me

Last but not least, I would like to thank then my instructors, now my friends; Özlem Akın and Gudjon Thor Erlendsson for guiding me through undergraduate education and graduation project in which I started studying on this topic.

Erinç Yıldırım

İzmir, 2019

TEXT OF OATH

I declare and honestly confirm that my study, titled “EARLY DESIGN STAGE MULTI LEVEL ARCHITECTURAL FLOOR PLAN ORGANIZATION BY USING GENETIC ALGORITHMS” and presented as a Master’s Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.

Erinç YILDIRIM

Signature

.....

August 26, 2019

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vii
ACKNOWLEDGEMENTS	ix
TEXT OF OATH	xi
TABLE OF CONTENTS	xiii
LIST OF FIGURES	xvii
LIST OF TABLES	xxi
SYMBOLS AND ABBREVIATIONS	xxiii
CHAPTER 1 INTRODUCTION	1
1.1. MOTIVATION	1
1.2. BACKGROUND.....	2
1.3. PROBLEM STATEMENT	4
1.4. RESEARCH OUTLINE.....	4
CHAPTER 2 LITERATURE REVIEW	6
2.1. PHYSICALLY-BASED MODELLING TECHNIQUES.....	7
2.2. EVOLUTIONARY ALGORITHMS BASED LAYOUT PLANNING	8
2.2.1. EVOARCH: AN EVOLUTIONARY ALGORITHM FOR ARCHITECTURAL LAYOUT DESIGN	9
2.2.2. ARCHITECTURAL LAYOUT DESIGN OPTIMIZATION.....	12
CHAPTER 3 METHODOLOGY	17
3.1. DATA INPUT	18
3.1.1. BUILDING PROGRAM.....	18
3.1.1.1. BUILDING PROGRAM CHART	18
3.1.1.2. RELATIONS MATRIX.....	19
3.1.2. SITE	20
3.1.2.1. BUILDING PLOT BOUNDARY	20
3.1.2.2. SITE OBJECTS	20
3.1.2.3. WEATHER DATA.....	20

3.2. IMPORTING BUILDING PROGRAM DATA INTO GRASSHOPPER.....	20
3.3. SPACEPOLER	20
3.4. SOLAR PERFORMANCE OF GENERATED SPATIAL ORGANIZATION	20
3.4.3. SOLAR RADIATION SIMULATION	21
3.4.4. SIMULATION TOOL.....	21
3.4.5. SIMULATION SETUP	21
3.4.5.1. TEST GEOMETRY.....	21
3.4.5.2. GRID SIZE	21
3.4.5.3. CONTEXT.....	21
3.4.6. SIMULATION RESULTS	22
3.5. OPTIMIZATION TOOL	22
3.6. GENERATIVE MODEL PARAMETERS.....	23
3.7. TIME SAVING MEASURES	24
3.7.7. 3 OBJECTIVES DO NOT EVALUATE INVALID SOLUTIONS.....	24
CHAPTER 4 SPACEEXPLORER	25
4.1. DATA VISUALIZATION.....	25
4.1.1. BUILDING PROGRAM DATA VISUALIZATION	25
4.1.2. RELATIONS DATA VISUALIZATION	26
4.2. CREATION OF 3D VOXELSPACE AND SELECTION POINTS	26
4.2.1. 3D VOXELSPACE	27
4.2.2. SELECTION POINTS.....	27
4.3. FLOOR AND LOCATION SELECTION.....	28
4.3.3. FLOOR SELECTION COMPONENT.....	28
4.3.4. LOCATION SELECTION COMPONENT	28
4.4. SPACEEXPLORER	29
4.4.5. SPACE ALLOCATION.....	29
4.4.6. FRAGMENTATION AND VALID COMBINATION CONSTRAINT	29
4.4.7. DOUBLE HEIGHT SPACES.....	30
4.4.8. CLOSENESS CALCULATION	31
4.4.8.1. CALCULATING EUCLIDIAN DISTANCE	31
4.4.8.2. CALCULATING RECTILINEAR DISTANCE.....	31

4.4.9. VISUALIZATION.....	32
4.4.9.1. VISUALIZATION OF CONSTRAINT VIOLATIONS	32
4.4.9.2. VISUALIZATION OF RELATED SPACES	33
4.4.9.3. VISUALIZATION OF FINALIZED SPATIAL ORGANIZATION	33
 CHAPTER 5 CASES.....	 35
5.1. CASE1: SEVEN SPACES: CORE AND COURTYARD.....	35
5.2. CASE2: SEVEN SPACES: DOUBLE CORE	36
 CHAPTER 6 RESULTS AND DISCUSSION.....	 39
6.1. CASE1: SEVEN SPACES: CORE AND COURTYARD.....	39
6.2. CASE2: SEVEN SPACES: DOUBLE CORE	42
6.3. CASE1: SEVEN SPACES: CORE AND COURTYARD.....	46
6.4. CASE2: SEVEN SPACES: DOUBLE CORE	49
6.5. CASE1: SEVEN SPACES: CORE AND COURTYARD.....	53
6.6. CASE2: SEVEN SPACES: DOUBLE CORE	56
 CHAPTER 7 CONCLUSIONS AND FUTURE RESEARCH	 61
REFERENCES.....	63
 APPENDIX 1 – FLOOR SELECTION COMPONENT C# CODE.....	 67
APPENDIX 2 – LOCATION SELECTION COMPONENT C# CODE	71

LIST OF FIGURES

Figure 1 Building Life Cycle	6
Figure 2 Iterative Design Workflow	6
Figure 3 Genetic Algorithm Workflow.....	9
Figure 4 Adjacency Matrix(right) and Graph Representation of Given Matrix(left)	10
Figure 5 Adjacency Preference Matrix	11
Figure 6 Initial Organization by User (Top Left),.....	12
Figure 7 Workflow of the Floor Layout Model	17
Figure 8 Building Program Data in Microsoft Excel.....	18
Figure 9 Portion of Building Program Data is Sent to Grasshopper.....	19
Figure 10 Relation Matrix in Microsoft Excel.....	19
Figure 11 Raw Data Output to Grasshopper.....	19
Figure 12 Building Program Visualization Pre-Optimization Process	25
Figure 13 Relations Matrix Visualization Pre-Optimization Process	26
Figure 14 Building Plot Boundary (Green), Selection Points (Purple), Voxel Centers (Pink)	27
Figure 15 Fragmentation Constraint Visualization with Valid and Invalid Combinations...	30
Figure 16 Spatial Configuration with Double Height Spaces (Purple and Yellow)	31
Figure 17 Constraint Violation Visualization, Layout Organization without Violation (Left), Layout Organization with Violation (Right).....	32
Figure 18 Visualization of Related Spaces (Left), Relations Between Spaces (Right)	33
Figure 19 Visualization of Spaces with Corresponding Colors	33
Figure 20 Example Building Program for Seven Space Case.....	35
Figure 21 Example Relation Matrix for Seven Space Case.....	36
Figure 22 Example Building Program for Seven Space Case.....	36
Figure 23 Example Relations Matrix for Seven Space Case	37
Figure 24 Solution Cluster After 60 Generation with pareto-front (dark red), first front (red)	

and history (yellow)	39
Figure 25 Samples from Solution Cluster	40
Figure 26 Visualization of Relations Between spaces of The Selected Solution	41
Figure 27 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)	42
Figure 28 Solution Cluster After 80 Generation with pareto-front (dark red), first front (red) and history (yellow)	42
Figure 29 Visualization of Relations Between spaces of The Selected Solution	44
Figure 30 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)	44
Figure 31 Visualization of Relations Between spaces of The Selected Solution	45
Figure 32 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)	45
Figure 33 Solution Cluster After 60 Generation with pareto-front (dark red), first front (red) and history (yellow)	46
Figure 34 Samples from Solution Cluster	47
Figure 35 Visualization of Relations Between spaces of The Selected Solution	48
Figure 36 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)	49
Figure 37 Solution Cluster After 80 Generation with pareto-front (dark red), first front (red) and history (yellow)	49
Figure 38 Visualization of Relations Between spaces of The Selected Solution	51
Figure 39 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)	51
Figure 40 Visualization of Relations Between spaces of The Selected Solution	52
Figure 41 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)	52
Figure 42 Solution Cluster After 60 Generation with pareto-front (dark red), first front (red) and history (yellow)	53
Figure 43 Samples from Solution Cluster	54

Figure 44 Visualization of Relations Between spaces of The Selected Solution.....	55
Figure 45 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)	56
Figure 46 Solution Cluster After 80 Generation with pareto-front (dark red), first front (red) and history (yellow)	56
Figure 47 Visualization of Relations Between spaces of The Selected Solution.....	58
Figure 48 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)	58
Figure 49 Visualization of Relations Between spaces of The Selected Solution.....	59
Figure 50 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)	59



LIST OF TABLES

Table 1 Literature Review.....	14
Table 2 Genetic Algorithm Parameters.....	22
Table 3 Fixed Parameters of Generative Model.....	23
Table 4 Optimization Driven Parameters.....	23
Table 5 Optimization Strategy	24
Table 6 Fitness Function Values of Selected Solutions from The Cluster.....	40
Table 7 Fitness Function Values of Given Solution Cluster.....	43
Table 8 Fitness Function Values of Selected Solutions from The Cluster.....	43
Table 9 Fitness Function Values of Selected Solutions from The Cluster.....	47
Table 10 Fitness Function Values of Given Solution Cluster.....	50
Table 11 Fitness Function Values of Selected Solutions from The Cluster.....	50
Table 12 Fitness Function Values of Selected Solutions from The Cluster.....	54
Table 13 Fitness Function Values of Given Solution Cluster.....	57
Table 14 Fitness Function Values of Selected Solutions from The Cluster.....	57



SYMBOLS AND ABBREVIATIONS

ABBREVIATIONS:

3D	3 Dimensional
MOO	Multi Objective Optimization
EPW	EnergyPlus Weather
HypE	Hypervolume Estimation Algorithm
SPEA 2	Strength Pareto Evolutionary Algorithm
RGB	Red Green Blue
GA	Genetic Algorithms
CPU	Central Processing Unit
m	Meter
GHz	Giga Hertz
GB	Giga Byte

SYMBOLS:

h	Height
l	Length



CHAPTER 1

INTRODUCTION

1.1. MOTIVATION

Architectural design is in a constant change in the light of technological developments of many sciences but especially computational sciences. Those developments first implemented as computerization of drafting and rendering technics of architectural design process and replaced how architects draft their ideas and documents the designs. Soon enough they have become part of design environment with the speed and precision they bring while drafting.

As the computer tools and technics advanced, their existence and effects are felt more in architectural design process and shape the design process of architects (Nagy et al., 2018). Such as architects not only utilized them to produce drawings but also as a computational manner to create generative models of complex architectural design problems in parametric design software (Menges & Ahlquist, 2011). In which, architect can model all design process beside a final geometric solution. Even though parametric model requires more effort at start than a conventional model of a single solution, it reduces the time in long term, it takes to iterate the and make possible to generate thousands of design alternatives since it includes the design procedures and series of operations to output design solution, this approach is much more feasible to change design as opposed to modelling each design by manual from scratch (Nagy et al., 2018).

Not only generating design alternatives but also evaluating each design alternative and calculating the performance value for each design alternative by utilizing computer simulations and analysis, helps architects make better design decisions.(Hornby, Globus, Linden, & Lohn, 2012; Preisinger & Heimrath, 2014; Stage, 2009)

It is not a simple task to tackle and find feasible design alternatives due to having huge search space that consists of all possible design alternatives. Therefore, choosing deterministic approaches to find feasible designs can take some time and even makes

it impossible to find as the complexity of the problem and count of decision variables increase (Dino, 2016). To overcome this, heuristics approaches which was already well implemented by many different disciplines such as industrial and electronics engineering are implemented to solve complex architectural problems.

1.2. BACKGROUND

Architectural layout design is a complex task by entailing numerous decision variables and criteria, which interact in complex ways, and as such it is defined by combinatorial complexity with set of relations between each space. These decision variables and criteria can be related with the sizes of spaces, views, accessibility, efficiency of circulation, solar performance of the layout. The role of architect is to satisfy the required criteria accordingly the case. Therefore, there are multiple architectural layout options in a huge design space which contains every possible combination of decision variable. architectural layout problems classed as NP-complete problems that cannot be solve in a reasonable time by deterministic approach (Jo & Gero, 1998).

Due to this, exhaustive search of the design space is not possible by manual design approaches through trial-error. Therefore, there are various researches is available which questions the possibility of automation of this complex task throughout the years (Michalek, Choudhary, & Papalambros, 2002; Nagy et al., 2018; Rodrigues, Gaspar, & Gomes, 2013; Skandhakumar, Salim, Reid, Drogemuller, & Dawson, 2016; Yeh, 2006; Yi & Malkawi, 2009). These researches on methodologies of general layout planning date back to 1950s and it aims to solve “Floor-plan design for industry” problem.

Problem of architectural layout organization has some similarities with the ones of other disciplines such as industrial and electronic engineering. For example, in architectural layout organization some of the spaces need to be close to each other while some of them need to be apart due to requirements based on user comforts and behaviors. We can see this sort of requirements facility layout problem of industrial engineering discipline and electronics component placement design on PCB(Ismail, Yusof, & Khalid, 2012).

Facility layout problem of industrial engineering discipline as spaces are translated to machines or production cells and the placement of those affects the workflow of production or in other word how materials circulate(Hathhorn, Sisikoglu, & Sir, 2013;

Korde & Shahare, 2017; Saraswat, Venkatadri, & Castillo, 2015), flow inside the facility therefore finding correct one plays a huge role of efficient production facilities due to that there are many research on this topic (Drira, Pierreval, & Hajri-Gabouj, 2007).

The first research which brings those layout planning technics to architecture discipline have been done by Mitchell and Dillion (Lobos & Donath, 2011). Many others have followed with different methods technics to find optimum architectural layout configurations automatically such as Expert Systems, Shape Grammar, Generative, Constraint-Based. The result of past researches become apparent that four major issues have aroused: how this complex problem is framed; how the combinatorial nature of the generated solutions is regulated; and how the solutions are evaluated according to multiple criteria related to the given requirement.

In recent years, in addition to these methods new ones are also implemented such as physically based, agent based, evolutionary algorithms, artificial intelligence

A physically-Based Modelling techniques research of Arvin and House is an example for utilizes “dynamic physics simulation” and the relations between spaces are defined as spring forces that pulls related spaces together on 2-dimensional plane therefore it can only solve single level floor plans. It calculates in real-time where architect can step in and according to the changes model responds by satisfying relations and sizes. This makes it a responsive design. This method does not aim to find the best solution rather it aims to enable designs to emerge.

A gradient and evolutionary algorithms-based research conducted by Michalek et al. grid based allocable spaces assigned to specific room activity. Due to having allocation and grid-based spaces makes the decision variables discrete which controlled by evolutionary algorithm.

As previous researches discussed before layout planning is a complex problem to solve by utilizing the deterministic approaches. Instead, one may make use of either heuristics, stochastic, or a combination of the two, in order to arrive at promising solutions. Another difficulty of the architectural layout design is to have time complexity that comes from the exponential function, in another words, finding the ultimate solution is not possible in a reasonable time period. This nature of the problem creates a necessity for a heuristic approach to find near optimal solutions. Evolutionary

algorithms are not only capable to solve complex design problems but also it is very possible that it may let design alternatives to emerge since the design solution space in relation with many decision variables in relation.

1.3. PROBLEM STATEMENT

This study aims to solve problem of architectural multi level layout planning with the use of evolutionary algorithms in grasshopper parametric design environment to guide architect early design stage. The framework of the study consists of; data collection about architectural building program defined by architect, generative model which is capable of generating multi level core based design alternatives which contains maximum of 50 different building functions including cores and courtyards, evaluation of generated one in terms of solar performance and circulation distances between related building functions, genetic algorithms based multi objective optimization.

1.4. RESEARCH OUTLINE

This research is organized in five chapters. Chapter 2 is devoted to a Literature Review. The review presents both architectural and other disciplines researches on topic of layout organization in terms of their limitations, methods and benefits. Also, implementations of genetic algorithms are given.

Chapter 3 is devoted to methodology in which in depth explanation of methodology of multi level layout planning with use of evolutionary algorithms step by step. Translation of buildable plot to voxels, decision variables and constraints, fitness functions, solar radiation analysis process and how architect interacts with framework are presented in this chapter

Chapter 4 devoted custom-made grasshopper plugin named spacexplorer in which explained its capabilities.

Chapter 5 devoted to cases are selected for this study

Chapter 6 devoted to results and discussion of Spacesearch in terms of how useful it is as an early stage design guide to find design alternatives that can developed to become architectural projects by examining how much they satisfy the requirements

Chapter 7 presents the Conclusion of this study. Limitations and the contribution to architectural layout planning and computational architecture are discussed and future studies can be found.



CHAPTER 2

LITERATURE REVIEW

Layout planning is essential part of architectural design problem since its is the first step of life cycle of building(Peckiene & Ustinovičius, 2017). Therefore, any decisions are taken during this early design phase has a huge effect on how building will perform accordingly user needs.

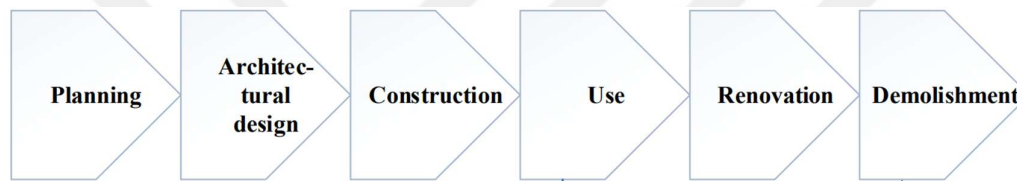


Figure 1 Building Life Cycle

Architects try to iterate as much as possible to come up with a suitable architectural layout plan, but it is labor and time intensive task. In addition that, it is an impossible task to conceptualize all possible architectural layout alternatives in search space by utilizing manual methods due to combinatorial complexity.

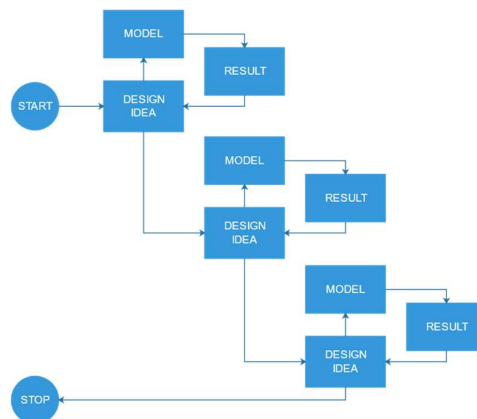


Figure 2 Iterative Design Workflow

This problem is also applicable to many other disciplines. For instance, in electronics engineering discipline when populating printed circuit boards, Electronic components that populate printed circuit boards have different thermal and power draw characteristics therefore organization of the electronic components are important for PCB design in terms of thermal performance of the PCB. On the other hand, in industrial engineering discipline, problem of facility layout needs to be solved for achieving efficient production lines by considering material handling and avoiding bottleneck in production flow.

Researches on layout planning problem first come up in industrial engineering with the aim to locate each component of factory layout in terms of efficiency and reduce the cost of material handling. Later, systematic layout planning was developed and accepted as go to method for this problem therefore research efforts were diminished.

Layout planning techniques that developed from those researches find its way to architecture discipline. The first research which brings those layout planning techniques to architecture discipline have been done by Mitchell and Dillion. Many others have followed with different methods techniques such as Expert Systems, Shape Grammar, Generative, Constraint-Based.

In recent years, researches focus on layout generation and optimization aspects, in addition to existing methods new ones are also implemented such as physically based, agent based, evolutionary algorithms, artificial intelligence to solve this complex problem. It is clear that most of the researches done on this topic depends on heuristic approaches due to their proven success of solving NP-Hard problems.

2.1. PHYSICALLY-BASED MODELLING TECHNIQUES

A physically-Based Modelling techniques research of Arvin and House is an example for utilizes “dynamic physics simulation” and the relations between spaces are defined as spring forces that pulls related spaces together on 2-dimensional plane therefore it can only solve single level floor plans. It calculates in real-time where architect can step in and according to the changes model responds by satisfying relations and sizes. This makes it a responsive design. This method does not aim to find the best solution rather it aims to enable designs to emerge.

2.2. EVOLUTIONARY ALGORITHMS BASED LAYOUT PLANNING

Architectural layout planning is non-deterministic polynomial-time hard (NP-hard) problem. Chance to find a solution decrease exponentially as the problem size increase therefore it not possible to solve by polynomial algorithm. Knowing the problem is computationally intractable implies that we may use heuristic approaches and that we also should aim to find nearly optimal solutions for which sometimes even approximation guaranties cannot be given. (Žerovnik, 2015)

Evolutionary algorithms are classed under heuristic approaches There are various Evolutionary algorithms even though they share basic principle: of the environmental pressure affects the population makes survival of the fittest or in other words natural selection. This increases the overall fitness value of population.

Based on the fitness values algorithm selects some of the individuals with better fitness values from the population to apply variant operators which are recombine and mutation to populate the next generation, rest of the individuals with weak fitness are die off.

Recombining requires two or more individuals and those individuals are called the parents to create one or more new individuals in other words the children.

Mutating is done to one or more individuals from the selected individuals to create new individuals. This operator helps to sustain variety of the population.

All those selection and variant operators are stochastic. Even though it is a small chance to select less fit one to survive become parent it is possible due to being random Also this stochastic nature can be observed while recombining and mutation the parents which pieces of the parents are transferred to the child, which pieces of selected individual to mutate.

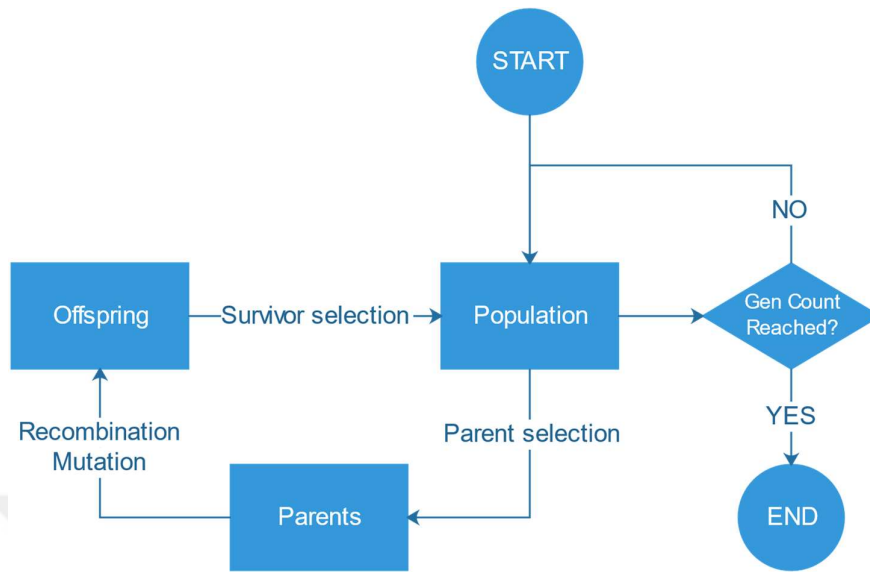


Figure 3 Genetic Algorithm Workflow

This process run until required generation count is reached. Size of the population, the mutation probability and crossover rate affect population characteristics therefore they affect the performance of the algorithm.

The premise of an evolutionary algorithm (to be further known as an EA) is quite simple given that you are familiar with the process of natural selection. An EA contains four overall steps: initialization, selection, genetic operators, and termination. These steps each correspond, roughly, to a particular facet of natural selection, and provide easy ways to modularize implementations of this algorithm category. Simply put, in an EA, fitter members will survive and proliferate, while unfit members will die off and not contribute to the gene pool of further generations, much like in natural selection.

2.2.1. EVOARCH: AN EVOLUTIONARY ALGORITHM FOR ARCHITECTURAL LAYOUT DESIGN

EvoArch utilizes evolutionary algorithm-based algorithm called genetic algorithm. The aim of the research is to automate the architectural design by offering efficient topological architectural organization based on the architectural building program. Author of the research states that current studies on automated architectural space planning which generates finalized layouts provide dull and predictable result and they

make architect to lose his or her freedom of creativity. As outcome of this critic framework of the study is developed in order to optimize locations of the spaces relative to each other topological level and let architect develop one of the evolved topologies and design spaces according to the topology.

EvoArch represents the building program as graph that consists of nodes and edges. Nodes corresponds to the spaces while edges show adjacencies. Adjacency matrices are converted from graphs and run genetic algorithm on them.

adjacency matrices hold data of nodes and edges, cells with 1 means edge will be created and, 0 means there is no edge between two nodes.

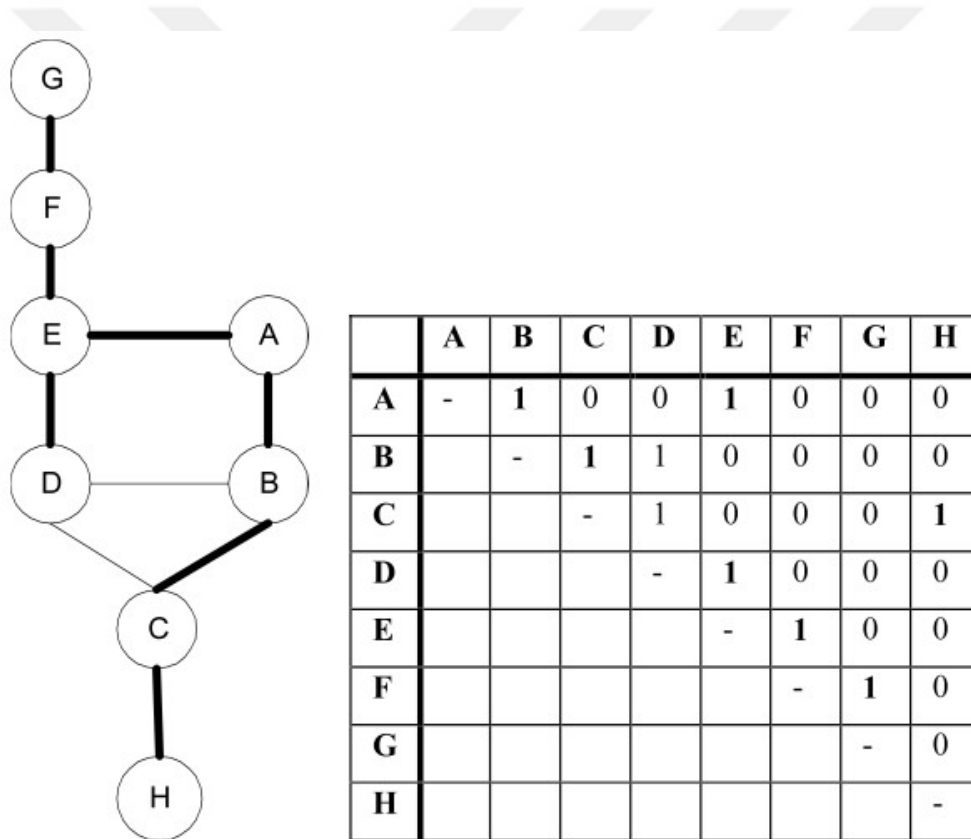


Figure 4 Adjacency Matrix(right) and Graph Representation of Given Matrix(left)

Algorithm can add edges or remove edges and swap labels of nodes to generate new graphs that represent architectural spatial topologies.

EvoArch is similar to the other evolutionary algorithms in terms of the algorithm step to find optimal architectural spatial topologies and utilizes roulette wheel selection

scheme to select individual which reproduce or mutate. Reproduction of the individual is based on steady state reproduction scheme.

The steady state scheme and generational reproduction are similar since the steady state bases on generational one. It does not converge as fast as generational one due to it only selects two parents to be crossed and produce two children, on the other hand on generational one selects larger percentage of the population to be crossed accordingly elitism value of the algorithm. The resulting children under goes to mutation operation before inserting to population therefore they replace old individuals.

Fitness function of evaluated graph is based on the adjacency preference matrix (fig). It represents preferred adjacency between spaces on a scale of -2 to 2 in which -2 means adjacency condition of spaces is not highly unwanted, -1 means it is unwanted, 1 means it is wanted, 2 means it highly wanted. Algorithm tries to maximize the fitness value by multiplying the preferred adjacency value with

	1. SA	2. MA	3. BED	4. LR	5. D&K	6. B	7. CP	8. P	9. CIR	10. EXT
1. Study area (SA)	-2	2	1	-2	-2	-2	-2	-1	3	3
2. Master ensuite (ME)	2	-2	2	-2	-2	-2	-2	3	3	3
3. Bedroom (BED)	1	2	2	-2	-2	3	-2	1	3	3
4. Living room (LR)	-2	-2	-2	-2	1	2	1	3	3	3
5. Dining & kitchen (D&K)	-2	-2	-2	1	-2	2	-2	1	3	2
6. Bathroom (B)	-2	-2	3	2	2	3	-2	-2	3	1
7. Car park (CP)	-2	-2	-2	1	-2	-2	-2	-2	3	3
8. Patio (P)	-1	3	1	3	1	-2	-2	-2	-2	3
9. Hall/stair/circulation area (CIR)	3	3	3	3	3	3	3	-2	3	-2
10. Exterior (EXT)	3	3	3	3	2	1	3	3	-2	3
Average	-0.2	0.3	0.9	0.5	-0.1	0.6	-0.7	0.2	2	2.2

Figure 5 Adjacency Preference Matrix

2.2.2. ARCHITECTURAL LAYOUT DESIGN OPTIMIZATION

This study aims to solve single level architectural floor plan in two steps one is topology optimization by considering energy performance such as heating, cooling and lighting cost while considering space efficiency of the layout.

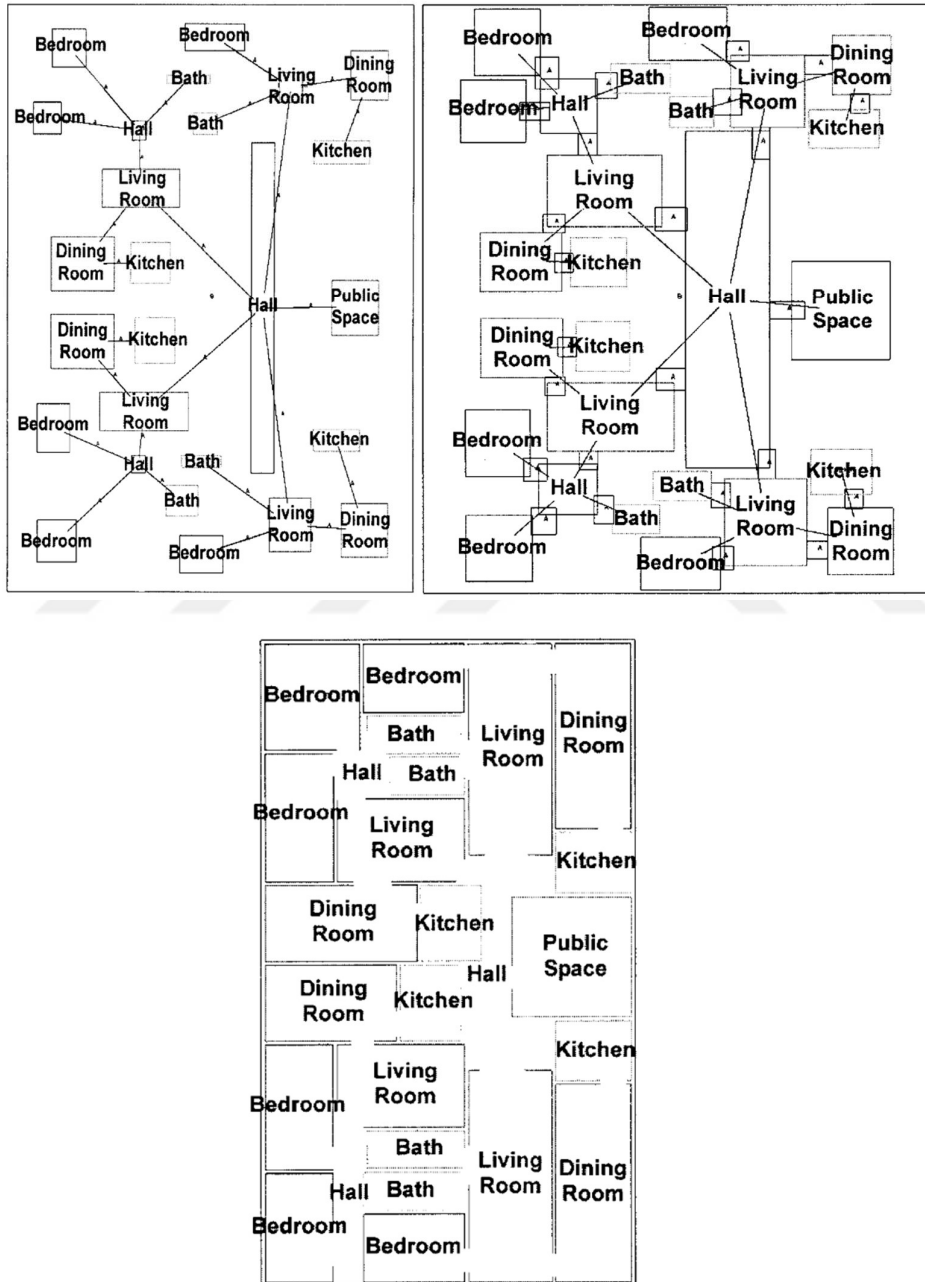


Figure 6 Initial Organization by User (Top Left), Intermediate Feasible Iteration (Top Right), Completed Design (Bottom)

Designer provide initial layout organization then the first algorithm optimizes the topology and if generated one is feasible the second algorithm which optimize the geometry of layout where walls and doorways are placed. By not applying second algorithm for the infeasible solutions computational time reduces.

Even though the results of this study provide some architectural layout design alternatives, but it depends on the initial layout provided by the designer therefore it can only help designer as a design exploration tool to find starting design ideas in the conceptual phase of architectural project.



Table 1 Literature Review

Date	Title	Method	Author	Case	Level	Simulation
2016	An evolutionary approach for 3D architectural space layout design exploration	Evolutionary algorithm	Ipek Gürsel Dino		Multi Level	
2016	Irregular architectural layout synthesis with graphical inputs	Simulated annealing algorithm	Hao Hua	House with 7 required functional spaces	Single Level	
2016	Evolutionary approach for spatial architecture layout design enhanced by an agent-based topology finding system	Agent Model, Evolutionary algorithm	Zifeng Guo, Biao Li	Three-level house that includes 17 rooms and 1 staircase, three-level office building that includes 1 staircase, 20 rooms, and 3 corridors	Multi Level	
2015	Architectural Layout Evolution through Similarity-Based Evaluation	Evolutionary algorithm	N. Onur Sönmez			
2014	Performance Based Architectural Design Optimization: Automated 3d Space Layout Using Simulated Annealing	Simulated annealing algorithm	Hwang Yi, Yun Kyu Yi	High-rise residential building	Multi Level	Yes (ECOTECT)
2011	Evolution of Architectural Floor Plans	Multi-objective genetic Algorithm	Robert W. J. Flack, Brian J. Ross	House	Multi Level	
2011	Estimation of solar radiation for buildings with complex architectural layouts		Stoyanka M. Ivanova		Single Level	Yes (ArchiPLAN)
2010	Architectural layout planning using genetic algorithms	Genetic Algorithm	Manish K Thakur, Monika Kumari, Madhabananda Das	Single flat having regular shaped spaces	Single Level	
2010	Architectural Space Planning using Genetic Algorithms	Genetic Algorithm	Manisha Verma, Manish K Thakur	single flat, arrangement of several flats on a single floor and extend the design for each floor and find out collective plan for a multi-storey apartment building	Multi Level	
2009	EvoArch: An evolutionary algorithm for architectural layout design	Evolutionary algorithm	Samuel S.Y. Wong, Keith C.C. Chan	House with 9 required functional spaces	Single Level	
2009	Architectural room planning support system using methods of generating spatial layout plans and evolutionary multi-objective optimization	Multi objective evolutionary algorithm	Makoto Inoue, Hideyuki Takagi	Floorplan with four spaces	Single Level	
2009	Spatial Layout Game-An Interactive Tool for Spatial Layout of Architectural Design		CHIEH-JEN LIN		Single Level	
2007	Prototypes for Automated Architectural 3D-Layout					
2005	Architectural layout optimization using annealed neural network	Simulated annealing algorithm	I-Cheng Yeh	Hospital building with 28 facilities	Single Level	
2005	Optimizing Architectural Layout Design via Mixed Integer Programming	Mixed integer programming	Keatruangkamala Kamol, Sinapiromsaran Krung		Single Level	
2004	Optimization of architectural layout by the improved genetic algorithm	Genetic Algorithm	Romualdas Baušys & Ina Pankrašovaite	House with 8 required functional spaces	Single Level	
2002	Architectural layout design optimization	Evolutionary algorithm	Jeremy Michalek, Ruchi Choudhary & Panos Papalambros	Small apartment complex with three separate apartments	Single Level	

Date	Title	Method	Author	Case	Level	Simulation
2002	Modeling architectural design objectives in physically based space planning	Physically based modeling	Scott A. Arvin, Donald H. House	9 spaces	Single Level	
2002	Architectural Interpretation of Cellular Automata		Robert J. Krawczyk			
2001	A hybrid representation of architectural precedents					
1997	Evolving design genes in space ayout planning problems	Genetic Algorithm	John S. Gero, Vladimir A. Kazakov	Office Layout, Hospital Layout	Single Level	
1995	Space layout planning using an evolutionary approach	Genetic Algorithm	Jun H.Jo, John S.Gero		Single Level	
1992	Heuristic Generation of Layouts (HeGeL): Based on a Paradigm for Problem Structuring					
1995	A Genetic Search Approach to Space Layout Planning	Multi-objective genetic Algorithm	Jun H. Jo, John S. Gero		Single Level	
2015	A 3-Dimensional Architectural Layout Generation Procedure for Optimization Applications : DC-RVD	Multi-objective constrained genetic algorithm	Ioannis Chatzikonstantinou	House with 7 required functional spaces	Multi Level	
	Architectural Layout Algorithm Based on Genetic Algorithm	Genetic Algorithm	GAO Liping, LIU Hong			



CHAPTER 3 METHODOLOGY

Spatial Explorer's workflow consists of four main parts as data input, generative model, and objective function evaluation and optimization.

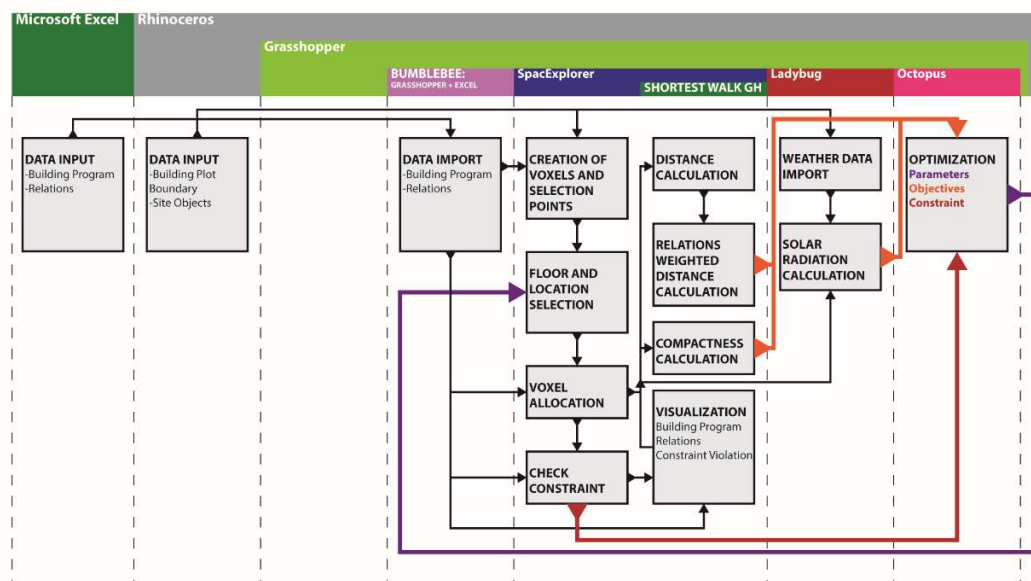


Figure 7 Workflow of the Floor Layout Model

In addition to first chart, relationship matrix of building functions --- according to relations between functions. Each relation gets a value in between -3 to 3. -3 means two building functions are related negatively in other words, they are expected to be far away from each other. 3 means they are related positively so they are expected to be closer to each other.

3.1. DATA INPUT

Data input step consists of collecting building program related data through Microsoft Excel. Site related data is set in rhino 3d environment.

3.1.1. BUILDING PROGRAM

3.1.1.1. BUILDING PROGRAM CHART

In this chart, user sets how many different building functions are present. This chart also has information of required floor area, being floor constrained, being double height space, being courtyard or core and whether it is possible to passthrough for each building function.

Name	Area	Height	Color	RGB Value	Ground Level Access	Daylight	Core(1) Courtyard(2)	Pass-Through
Core1	50	0	Blue		TRUE		1	1
Core2	50	0	Orange		TRUE		1	1
Courtyard	100	0	Green		TRUE		2	1
chemotherapy DPU	220	0	Red		TRUE	FALSE	0	0

Figure 8 Building Program Data in Microsoft Excel

First column of the BPC contains the names of each building function as string also according to this data, count of building functions is calculated, and these values are sent to relations matrix chart.

Second column of the BPC contains desired floor areas for each building function in integer values.

Third column of BPC contains binary values for each building function, if the building function desired to be double height space true value is set, otherwise it is set to false.

Fourth and fifth columns of BPC contain the color data of each building function, user set a color by changing the fill color of cell under Color Column, according to the fill color, RGB value is calculated and written to the RGB Value Column.

Fifth column of BPC contains binary values for each building function, if the building function is constraint to the ground level, then true value is set, otherwise it is set to false.

Sixth column of BPC contains data for each building function whether it is core or courtyard or normal space. If it functions as core, then it gets “1”. If it functions as courtyard then it gets “2”.

Seventh column of BPC contains binary values for each building function, if the building function is pass through space, then true value is set, otherwise it is set to false.

After filling the data in BPC, data except the headers are sent to a new sheet to be exported to Grasshopper. Data are sent to grasshopper as following figure.

Core1	50	0	0	TRUE	0	1	1
Core2	50	0	0	TRUE	0	1	1
Courtyard	100	0	0	TRUE	0	2	1
chemotherapy DPU	220	0	0	TRUE	FALSE	0	0

Figure 9 Portion of Building Program Data is Sent to Grasshopper

3.1.1.2. RELATIONS MATRIX

Relations matrices of building functions form according to relations between functions. Names of the elements of matrix in other words building functions are called from spacenames column of BPC. User fills the matrix with values between -3 to 3 according to relations between building functions -3 means two building functions are related negatively in other words, they are expected to be far away from each other. 3 means they are related positively so they are expected to be closer to each other.

	Core1	Core2	Courtyard	chemotherapy DPU
Core1	0	3	3	1
Core2	3	0	-3	1
Courtyard	3	-3	0	3
chemotherapy DPU	1	1	3	

Figure 10 Relation Matrix in Microsoft Excel

After filling the data in RMC, only relations representative integer values data are sent without headers to Grasshopper. Data are sent to grasshopper as following figure

0	3	3	1
3	0	-3	1
3	-3	0	3
1	1	3	0

Figure 11 Raw Data Output to Grasshopper

3.1.2. SITE

3.1.2.1. BUILDING PLOT BOUNDARY

Building plot boundary's geometrical data is set in rhino as curve. This geometry is not only limited by rectangular shapes, but it can be irregular shapes.

3.1.2.2. SITE OBJECTS

Site objects are modelled in rhino then imported in grasshopper as rhino referenced Boundary represented (BREP) geometries.

3.1.2.3. WEATHER DATA

Weather data of the site location is downloaded in .epw format to be opened by Ladybug GenCumulativeSkyMtx.

3.2. IMPORTING BUILDING PROGRAM DATA INTO GRASSHOPPER

Importing data from Microsoft Excel is carried out by “Bumblebee: Grashopper + Rhino”. It is an add-on that makes possible to exchange data between Microsoft Excel and Grasshopper. This add-on is chosen due to its capability to dynamically read excel data which makes whole process more user friendly.

3.3. SPACEPOLER

Custom grasshopper component which is capable of visualization of building program data; generating architectural layouts; evaluating generated layouts in terms of compactness and proximity of the related spaces. The details of this custom grasshopper component will be explained in chapter 4

3.4. SOLAR PERFORMANCE OF GENERATED SPATIAL ORGANIZATION

The solar performance of generated spatial organization is evaluated in terms of solar radiational analysis by using Ladybug. This implementation

3.4.3. SOLAR RADIATION SIMULATION

Solar radiation simulation is simple compared to more advanced simulation methods such as single or multi zone energy simulations but due to being less computationally heavy makes it more preferable one. Besides that, since the material decisions are not present utilizing energy simulation is not possible at this stage of design.

3.4.4. SIMULATION TOOL

Ladybug is an open source grasshopper plugin which is capable importing “.epw” weather files, visualization of weather data, analyze weather data and running simulations. There are many examples of utilization of ladybug and honeybee as a simulation tool for architectural studies.

3.4.5. SIMULATION SETUP

Simulation setup is a required step for running solar radiation simulation. During this stage test geometry and grid size plays a huge role how much computational time required for evaluating solar performance of each generated spatial organization.

3.4.5.1. TEST GEOMETRY

Only the outer surfaces of generated spatial organization geometry is set as a test geometry for the solar radiation analysis instead of all surfaces. Therefore, it only generates test points on outer surfaces and by having less test points it takes less time to evaluate.

3.4.5.2. GRID SIZE

Grid size is set to one meter to generate test points. More test points provides more defined results but also takes significant amount of time compared to less dense test point based simulation

3.4.5.3. CONTEXT

Previously modelled and referenced rhino objects which represent the site objects near building plot is as a context for the solar radiation analysis. Due to their casted shadows on building plot affects the outcome of the simulation results.

3.4.6. SIMULATION RESULTS

Simulation gives the amount of collected solar radiation of each test point on test geometry in kWh and visualize the results in blue to red gradient where the surface with the lowest value gets blue color and the surface with the highest value gets red

3.5. OPTIMIZATION TOOL

Octopus plugin is a optimization plugin which is developed for Grasshopper and it is capable of finding cluster of near optimal solutions by two different genetic algorithms and Non dominated sorting. One of genetic algorithms is SPEA-2 and the other one is HypE. Also, there are many researches that utilize octopus for optimization of architectural problems. Parameters of genetic algorithms are defined as follows

Table 2 Genetic Algorithm Parameters

Genetic Algorithm	HypE
Mutation Operator	HypE Mutation
Elitism Rate	0.5
Mutation Probability	0.3
Mutation Rate	0.5
Crossover Rate	0.8
Population Size	100

3.6. GENERATIVE MODEL PARAMETERS

Fixed parameters of generative model can be found table below.

Table 3 Fixed Parameters of Generative Model

Name	Range	Type
Floor Count	2	Independent
Voxel Count	158	Dependent (Voxel Size, Site Boundary, Floor Count)
Voxel Size		Independent
Selection Point Count	471	Dependent (Voxel Count, Site Boundary, Floor Count)
Floor Selection (Ground Constrained)	0	Dependent
Area		Independent
Voxel Required Per Space		Dependent (Area)

Table 4 Optimization Driven Parameters

Name	Range	Type
Selection Point Location	[0, Selection Point Count]	Impendent
Floor Selection	[0, Floor Count]	

Combinatorial hardness can be seen according to the problem formulated for the given parameters in the equation down below

$$C(\text{SelPtCNT}, \text{SpaceCNT}) = \frac{\text{SelPtCNT}!}{(\text{SpaceCNT}! (\text{SelPtCNT} - \text{SpaceCNT})!)}$$

$$C(972,7) = \frac{972!}{(7! (972 - 7)!)}$$

$$C(972,7) = 1.277090615E + 17$$

3.7. TIME SAVING MEASURES

Spaceexplorer is capable of generating and evaluating a design alternative per second. While evaluation process makes the large portion of the computational load. Average computational time for solar radiation simulation is 750 ms whereas calculation shortest paths between spaces changes in between 100 ms and 200 ms.

The four different approach has been tried to reduce the computational time by not evaluating computational heavy objective functions of invalid solutions which explained below.

3.7.7. 3 OBJECTIVES DO NOT EVALUATE INVALID SOLUTIONS

This approach takes in to account total solar radiation, relations weighted circulation and compactness. Any solution violates fragmentation constraint is not evaluated in terms of solar radiation and relations weighted circulation in order to reduce computational time.

Table 5 Optimization Strategy

Objective Functions	Evaluate for invalid solutions	Constraints
Total Solar Radiation	NO	Fragmentation
Relations Weighted Circulation	NO	
Compactness	YES	

CHAPTER 4 SPACEEXPLORER

4.1. DATA VISUALIZATION

After importing the building program data to grasshopper, two custom grasshopper components create rhino grasshopper geometries in rhino viewport to visualize the data.

4.1.1. BUILDING PROGRAM DATA VISUALIZATION

This custom grasshopper component visualizes the building program data. User inputs Building Program data and relations data. According to the relation count, it colors the spheres that represents building functions. Red to blue gradient is chosen, max value set to be red. The size of the spheres represents areas of building functions.

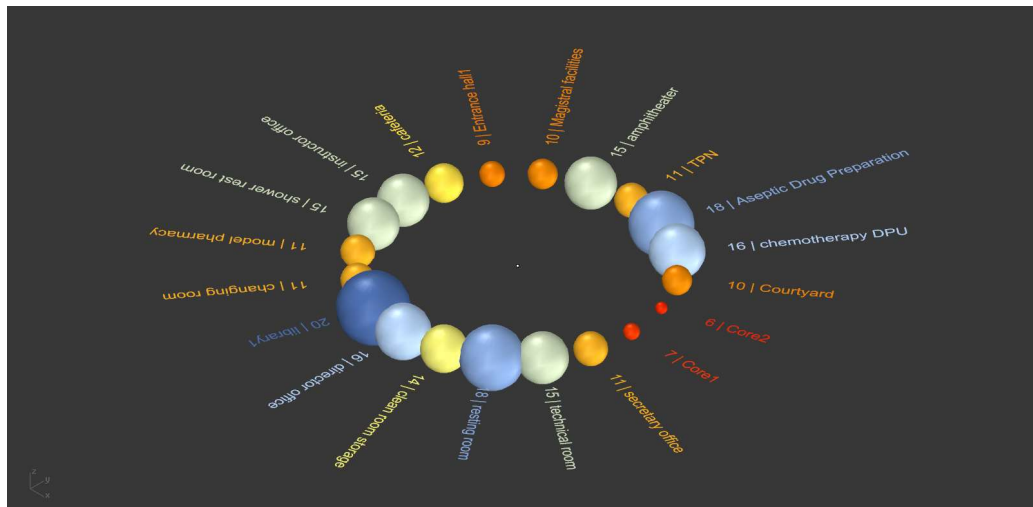


Figure 12 Building Program Visualization Pre-Optimization Process

4.1.2. RELATIONS DATA VISUALIZATION

This custom grasshopper component visualizes the relations between selected building function and other building functions. User inputs Building Program data and relations data and selects building function by the help of grasshopper's parameter input component named Control Knob. According to the relation value, it colors the connecting curve between building functions. Red to blue gradient is chosen, red corresponds to -3 and blue corresponds to 3.

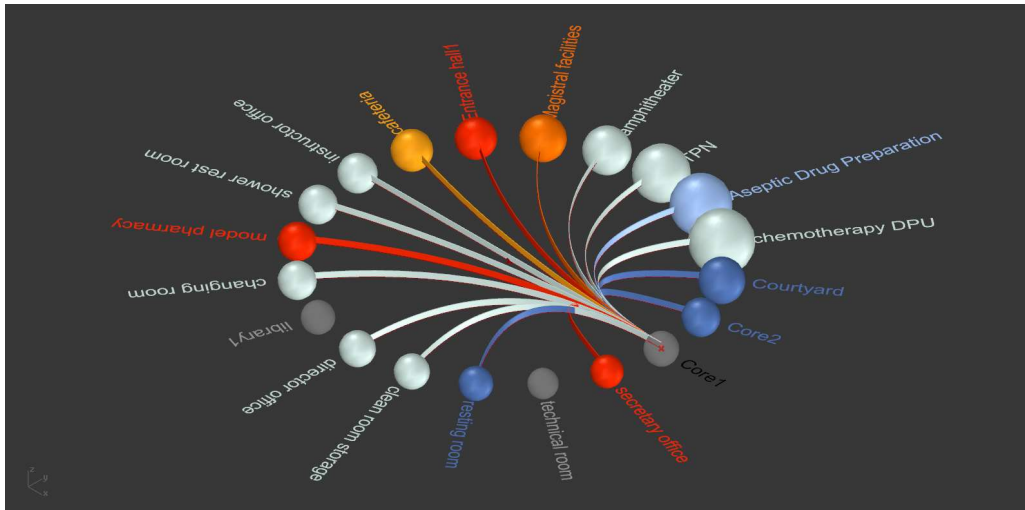


Figure 13 Relations Matrix Visualization Pre-Optimization Process

4.2. CREATION OF 3D VOXELSPACE AND SELECTION POINTS

Custom component is compiled to create voxels inside irregular building plot boundary and selection points. The inputs are site boundary, voxel count, floor height, floor count, voxel size and desired areas of building function. It outputs selection points, selection points count, voxel count, voxel centers, voxel sizes, floor count and floor height.

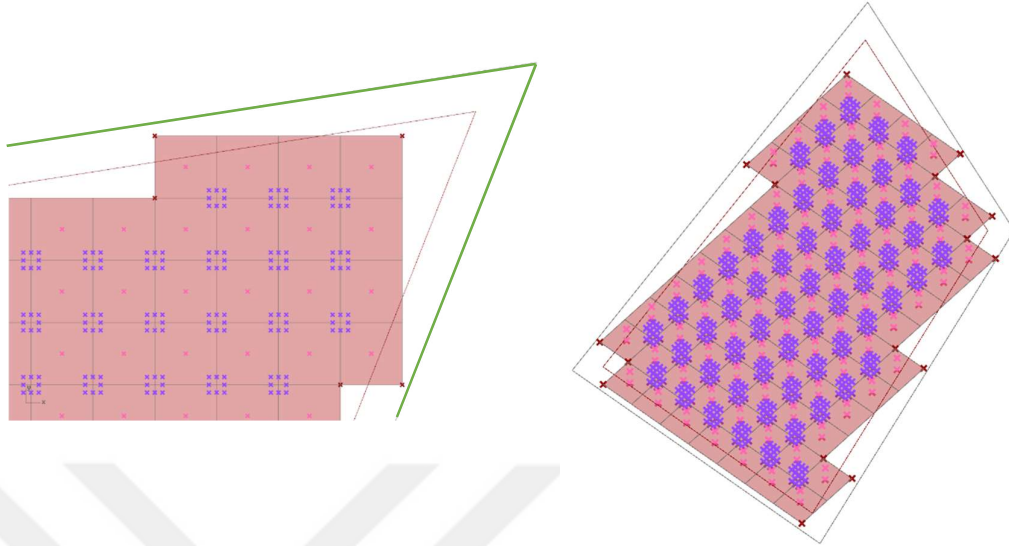


Figure 14 Building Plot Boundary (Green), Selection Points (Purple), Voxel Centers (Pink)

4.2.1. 3D VOXELSPACE

The component creates two dimensional grid with desired grid sizes then checks if cells of the newly created grid, inside or not. The cells are outside the boundary removed. Afterwards remainder cells are extruded to desired floor height and copied on z axis to reach desired floor count. At the end of these steps.

4.2.2. SELECTION POINTS

Selection points creation procedure is as follows. Remained cells are joined then the outer edge of the surface is used to find outer edge nodes and put them in a data list. It finds nodes of each remained cell and creates second data list that contains many duplicate ones. The second data list is cleaned by removing duplicate points and the points that are on the outer edge.

XY coordinate planes are placed on the remained points and 1 meter by 1 meter rectangles are created at the origin points of those XY coordinate planes. Corner points of the rectangles and the mid points of each edges of rectangle. These points are copied in Z direction floor count times and floor height apart.

4.3. FLOOR AND LOCATION SELECTION

Selecting the location for building function among available selection points is a complex task due to selection point count is not fix value and dependent to voxel sizes, plot shape and size also the count of building functions is not a same for every case. Utilizing generic list item component and fixed number slider component of grasshopper is not possible therefore two c# component compiled to tackle this limitation. They are similar to each other in terms of how they interact with the number sliders to change bounds of the sliders and manage the data inside.

4.3.3. FLOOR SELECTION COMPONENT

Floor selection component instantiates named integer number slider or panel component with the data “0” inside for each building function rankly. If building function is constraint to ground level, component instantiates panel component with the data “0” inside. If it is free to locate itself in upper floors, component instantiates integer number slider component and sets its maximum accordingly floor count.

After first instantiation it interacts with the number sliders that connects to its input to modify and set maximum and values of every slider, in case of floor count and ground level constraints are changed.

Floor selection component outputs an integer value for each building function that shows which floor building floor will be.

4.3.4. LOCATION SELECTION COMPONENT

Location selection component instantiates named integer number slider for each building function rankly, maximum value of each slider is same and equal to the selection point count per floor.

After first instantiation it interacts with the number sliders that connects to its input to modify and set maximum and values of every slider in case of voxel count therefore selection point count is changed.

Location selection slider value and floor selection slider value combine in to one value as follows

$$\begin{aligned} & \text{floorselectionsliderValue} * \text{pointcountperfloor} \\ & + \text{locationselectionsliderValue} \end{aligned}$$

Then, component selects that specific selection point for each building function from array of selection points and outputs it.

4.4. SPACEEXPLORER

Space explorer component gets selection points and selects nearest available voxels accordingly their required voxel count to satisfy desired area for each building function. Not only that but also it calculates circulation between spaces and check if the combination of selected voxels are valid. In addition to that it visualizes constraint violations, relations between selected building function space and other spaces.

4.4.5. SPACE ALLOCATION

Space allocation process is a straightforward and sequentially process, it searches among available voxels for nearest to building function location selection point and selects some voxels to satisfy the desired area. Then those selected voxels removed from list to prevent overlapping spaces by not selected the ones already selected. This repeats for each building functions, therefore sort of building functions has a huge impact on success of this process for that reason first core and courtyard spaces locate themselves due to their location affect all floor levels then other spaces locates themselves by locating larger ones first. After this process first allocation finalize.

4.4.6. FRAGMENTATION AND VALID COMBINATION CONSTRAINT

Being this straightforward it has some limitations such as this process can produces results with fragmented spaces due to selected voxel combination, therefore to identify and constrain those invalid combination. The following constraint applied. The constraint violation of each space is calculated according to the edge count of allocated voxel and allowed edge count of voxel count. For example, if the space need to allocates; two voxels, it needs to have six edges when voxels combined, three voxels, it needs to have eight edges when voxels combined. Any other number of edges means

that space allocated a combination of voxels that are apart therefore space is fragmented.

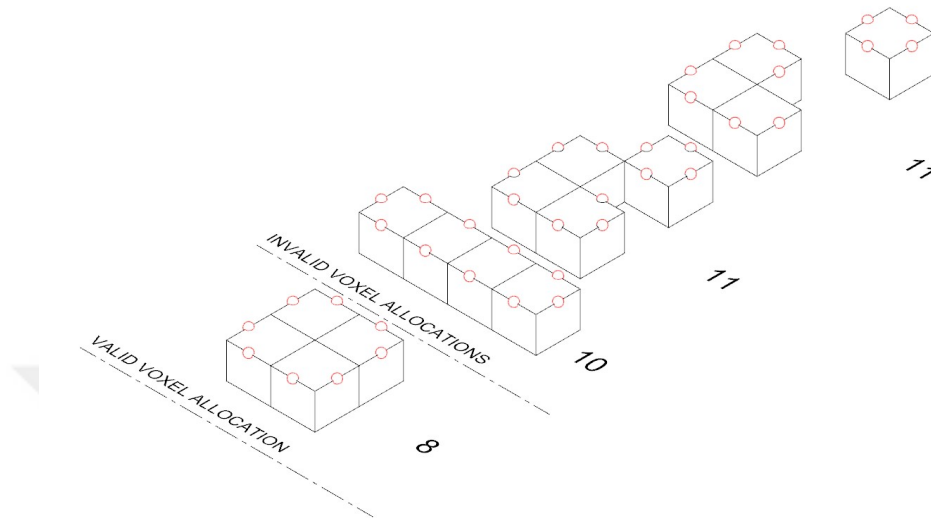


Figure 15 Fragmentation Constraint Visualization with Valid and Invalid Combinations

Not only this constraint prevents spaces to fragment but also it helps to avoid generating whole but oddly shaped spaces.

4.4.7. DOUBLE HEIGHT SPACES

After first allocation process finalize, voxels of building functions that can be double height space check if the above voxel is available if it is available it automatically merges in to building function and allocation process finalize.

Due to how this procedure is constructed it is not certain that at the end of the process, all spaces desired to be double height end up double height.

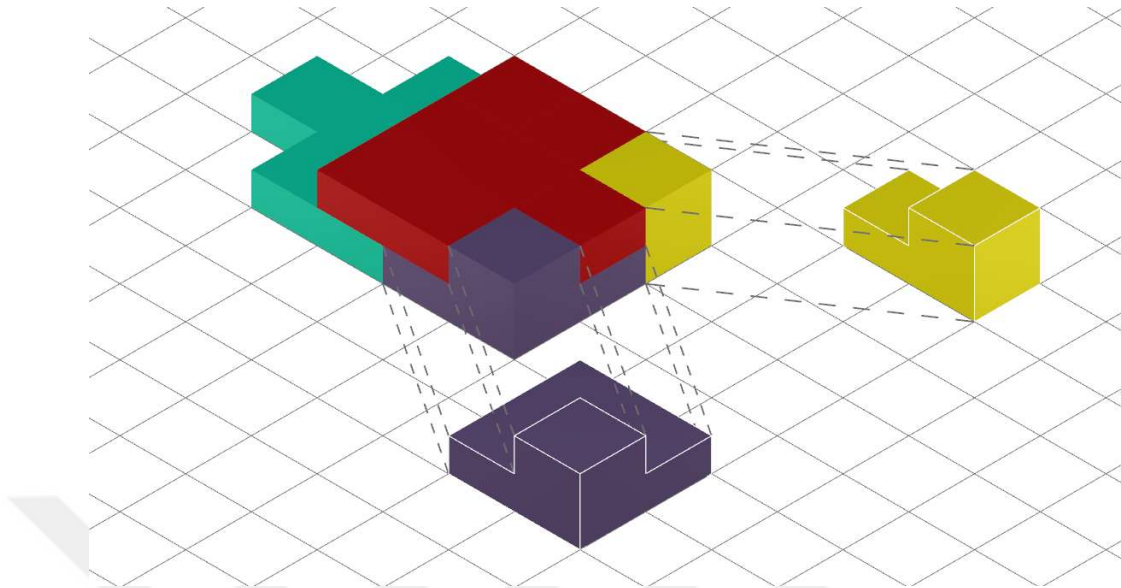


Figure 16 Spatial Configuration with Double Height Spaces (Purple and Yellow)

4.4.8. CLOSENESS CALCULATION

After allocation process is finalized, distances between building functions are calculated. For this there are two different method for calculating the distances. One calculates Euclidian and the other one calculates rectilinear distances.

4.4.8.1. CALCULATING EUCLIDIAN DISTANCE

Distances between spaces in relation both in terms of positive and negative calculated by using Euclidian calculation. Center points of related building function spaces connected by lines then lengths of those lines give distances.

Calculating by using this method is limited therefore not all the spatial combinations that generated based on this method works but it takes less computing time than rectilinear one.

4.4.8.2. CALCULATING RECTILINEAR DISTANCE

Calculating distances between related spaces by using “Shortest Walk GH” algorithm. It utilizes A* search algorithm and much more certain then the calculating Euclidean distances, but it takes much more computing time. It gets bottom faces of merged spaces and find edge of the surfaces then explodes this edge curve in to smaller segment. After this process curve list has multiple duplicates. After cleaning this list by removing duplicates. After this step if there are spaces that can be part of circulation

and marked as pass-through space in Excel file, their inner curves are also inserted in to curve list. For the vertical circulation, vertical edges of building functions marked as core are extracted and inserted in to curve list.

The curves inside this list are creates network of curves where “Shortest Walk” finds shortest paths using this curve network and Euclidean distances between spaces. The shortest paths between spaces exploded in small segments then duplicate segments are removed to calculate total rectilinear distances.

4.4.9. VISUALIZATION

The Space explorer component can visualize some metrics of generated spatial organization such as constraint violation, related spaces of selected building function, and coloring finalized spatial organization with corresponding colors.

4.4.9.1. VISUALIZATION OF CONSTRAINT VIOLATIONS

After allocation process some of the spaces fragment or shaped undesired combinations as discussed before, each building function space end up with a binary value if it violates the constraint or not. Those binary values are checked. if binary value is false in other words, building function space violates the constraint then the space is colored with red color in spatial organization diagram. If binary value is true in other words, building function space does not violate the constraint then the space is colored with blue color in spatial organization diagram.

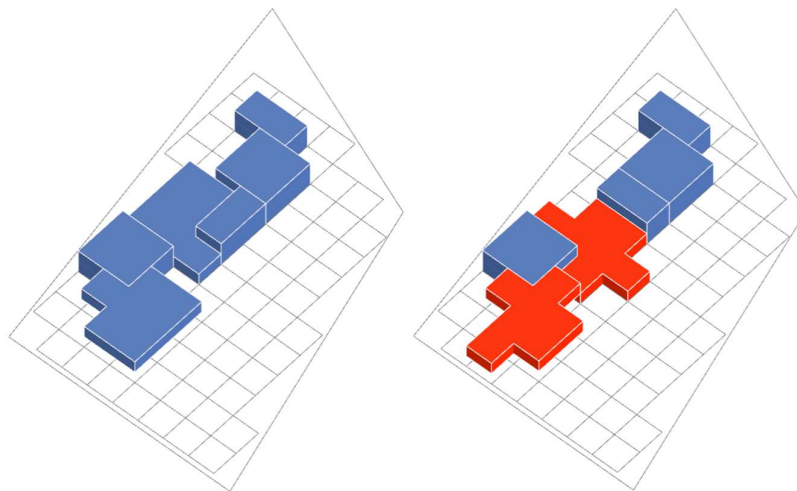


Figure 17 Constraint Violation Visualization, Layout Organization without Violation (Left), Layout Organization with Violation (Right)

4.4.9.2. VISUALIZATION OF RELATED SPACES

Visualization of related spaces of selected building function is implemented to see how building functions that are in relation are distributed. According to the relation value of the selected building function space, it colors the connecting curve between building functions. Red to blue gradient is chosen, red corresponds to -3 and blue corresponds to 3.

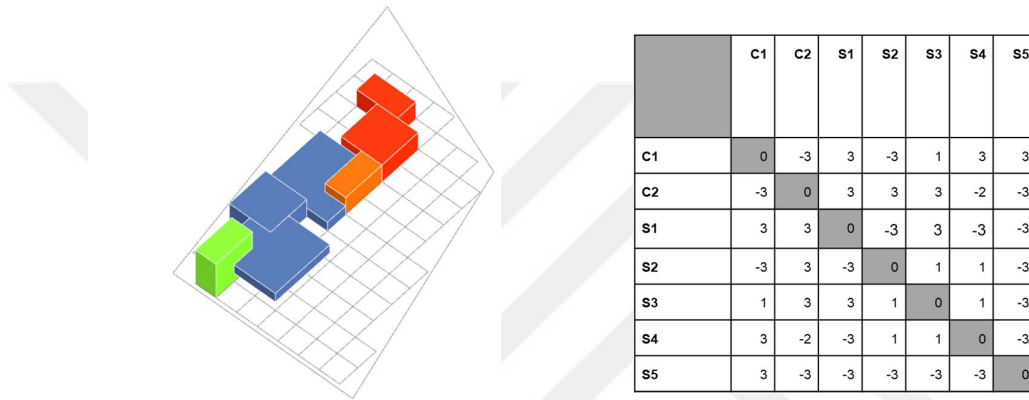


Figure 18 Visualization of Related Spaces (Left), Relations Between Spaces (Right)

4.4.9.3. VISUALIZATION OF FINALIZED SPATIAL ORGANIZATION

Visualization of finalized spatial organization process utilizes the colors user has choose for each building function in Excel during data input. RGB materials composed in grasshopper with values come from Excel data. Then those RGB materials are assigned to corresponding building function space in spatial organization diagram.

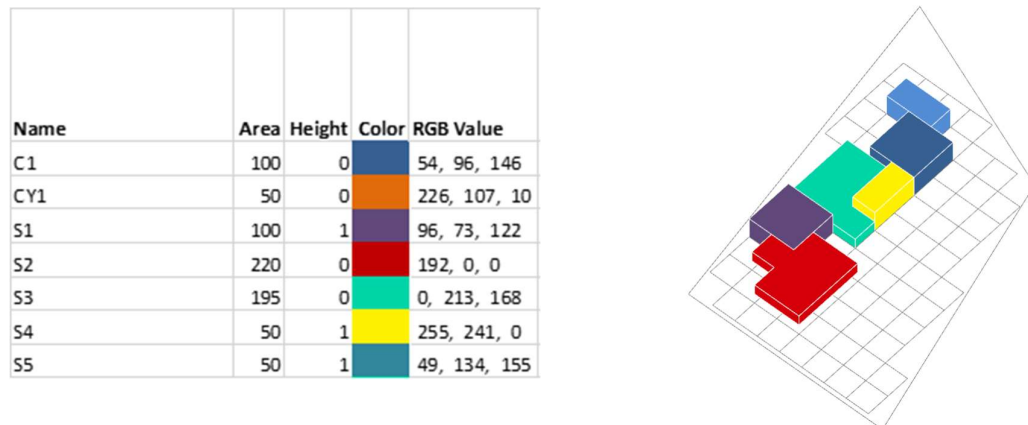


Figure 19 Visualization of Spaces with Corresponding Colors

CHAPTER 5

CASES

In this chapter, results of spaceexplorer are discussed in terms of its limitations and capabilities of providing different multi level architectural layout plans in two different cases. First one consists of seven spaces consist of various types in terms of relationship and geometry such as core, courtyard, single height space, double height and pass through space to present its results in depth. The second one consists of seven spaces with double cores.

5.1. CASE1: SEVEN SPACES: CORE AND COURTYARD

In this case following data is used in which spaces with different characteristics exists. Space named C1 is a core where vertical circulation happens. CY1 is courtyard in other words it is a void space in relation with other spaces. S1 is a double height space which is constraint to the ground level. S2 and S3 is single height spaces which can be in upper levels. S4 is double height space in addition to that circulation can passthrough it. S5 is double height space.








Name	Area	Height	Color	RGB Value	Ground Level Access	Core(1) Courtyard(2)	Pass-Through
C1	100	0		54, 96, 146	TRUE	1	1
CY1	50	0		226, 107, 10	TRUE	2	1
S1	100	1		96, 73, 122	TRUE	0	0
S2	220	0		192, 0, 0	FALSE	0	0
S3	195	0		0, 213, 168	FALSE	0	0
S4	50	1		255, 241, 0	TRUE	0	1
S5	50	1		49, 134, 155	TRUE	0	0

Figure 20 Example Building Program for Seven Space Case

Relations between the seven spaces can be seen in the following figure. In this figure spaces with -3 value are not desired to be in close vicinity whereas spaces with +3 value are desired to be close vicinity.

	C1	CY1	S1	S2	S3	S4	S5
C1	0	3	3	1	2	3	-3
CY1	3	0	3	3	3	1	-3
S1	3	3	0	-3	3	-3	-3
S2	1	3	-3	0	1	1	-3
S3	2	3	3	1	0	1	-3
S4	3	1	-3	1	1	0	-3
S5	-3	-3	-3	-3	-3	-3	0

Figure 21 Example Relation Matrix for Seven Space Case

5.2. CASE2: SEVEN SPACES: DOUBLE CORE

In this case following data is used in which spaces with different characteristics exists. Space named C1 and C2 are cores where vertical circulation happens. S1 is a double height space which is constraint to the ground level. S2 and S3 is single height spaces which can be in upper levels. S4 is double height space in addition to that circulation can pass through it. S5 is double height space.








Name	Area	Height	Color	RGB Value	Ground Level Access	Core(1) Courtyard(2)	Pass-Through
C1	100	0		54, 96, 146	TRUE	1	1
C2	50	0		226, 107, 10	TRUE	1	1
S1	100	1		96, 73, 122	TRUE	0	0
S2	220	0		192, 0, 0	FALSE	0	0
S3	195	0		0, 213, 168	FALSE	0	0
S4	50	1		255, 241, 0	TRUE	0	1
S5	50	1		49, 134, 155	TRUE	0	0

Figure 22 Example Building Program for Seven Space Case

Relations between the seven spaces can be seen in the following figure. In this figure spaces with -3 value are not desired to be in close vicinity whereas spaces with +3 value are desired to be close vicinity.

	C1	C2	S1	S2	S3	S4	S5
C1	0	-3	3	-3	1	3	3
C2	-3	0	3	3	3	-2	-3
S1	3	3	0	-3	3	-3	-3
S2	-3	3	-3	0	1	1	-3
S3	1	3	3	1	0	1	-3
S4	3	-2	-3	1	1	0	-3
S5	3	-3	-3	-3	-3	-3	0

Figure 23 Example Relations Matrix for Seven Space Case

CHAPTER 6

RESULTS AND DISCUSSION

In this chapter results of spaceexplorer are discussed in terms of its limitations and capabilities of providing different multi level architectural layout plans in two different cases. First one consists of seven spaces consist of various types in terms of relationship and geometry such as core, courtyard, single height space, double height and pass through space to present its results in depth. The second one consists of seven spaces with double cores.

Results of each optimization approach compared against each others in terms of how much they reduce the objective functions after 30 generations. Changes of objective function values and constraint violation count can be seen

6.1. CASE1: SEVEN SPACES: CORE AND COURTYARD

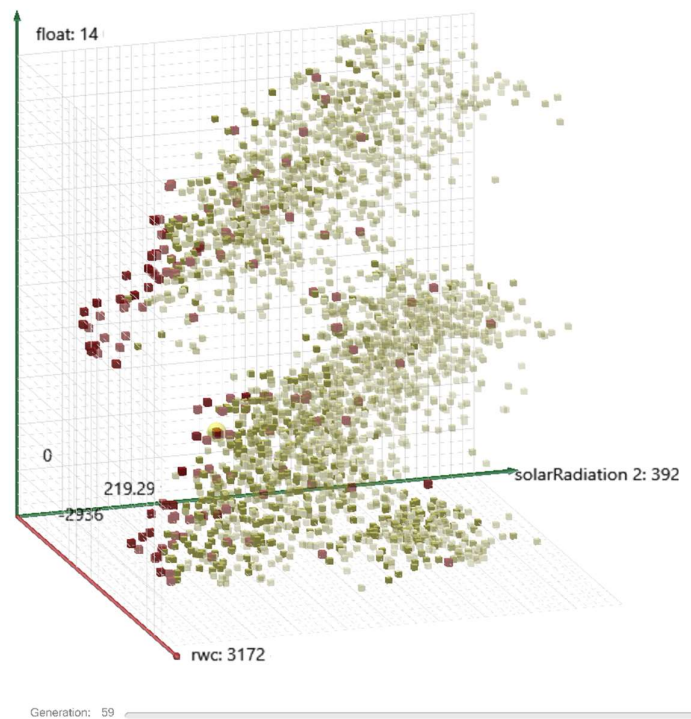


Figure 24 Solution Cluster After 60 Generation with pareto-front (dark red), first front (red) and history (yellow)

After running the algorithm for 60 generation, in total 163 solutions are found 24 of these solutions are non dominated solutions. The cluster of solutions provide design alternatives with different fitness function values. Range of the three fitness function values can be found in the following table.

Table 6 Fitness Function Values of Selected Solutions from The Cluster

Fitness Function	Min Value	Max Value
RelationsWeightedCirculation	-2936	3192
Total Solar Radiation	219.29	392
Floating Voxel Count	0	14

Some example solutions are selected from the cluster of 163 and can be found in the following figure

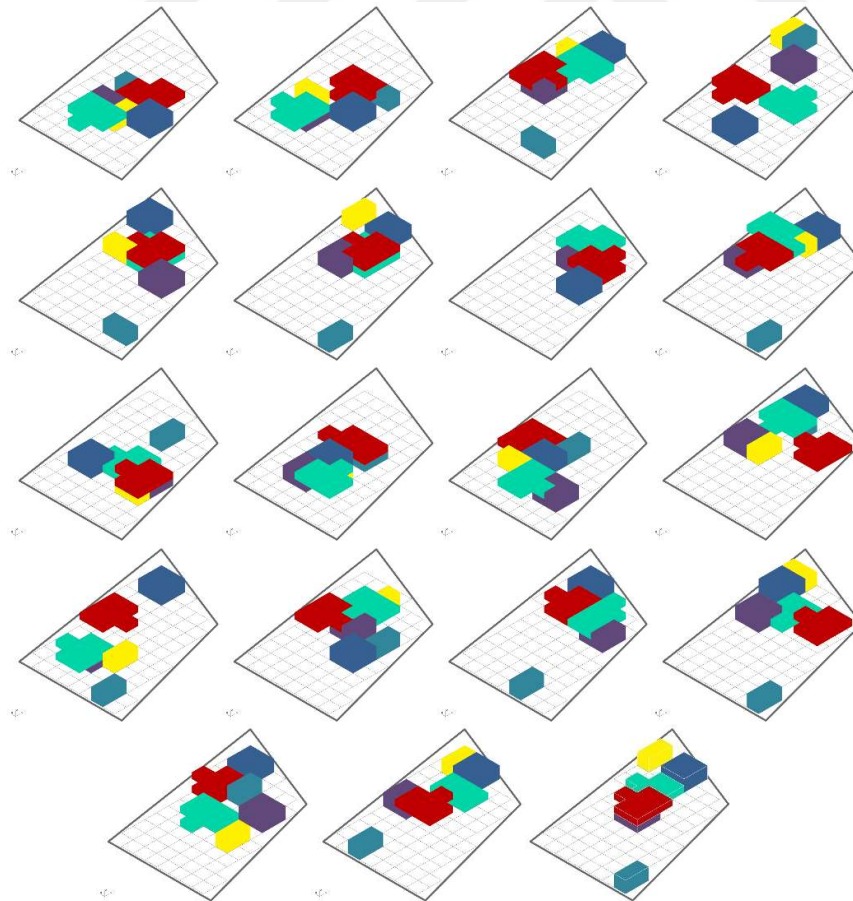


Figure 25 Samples from Solution Cluster

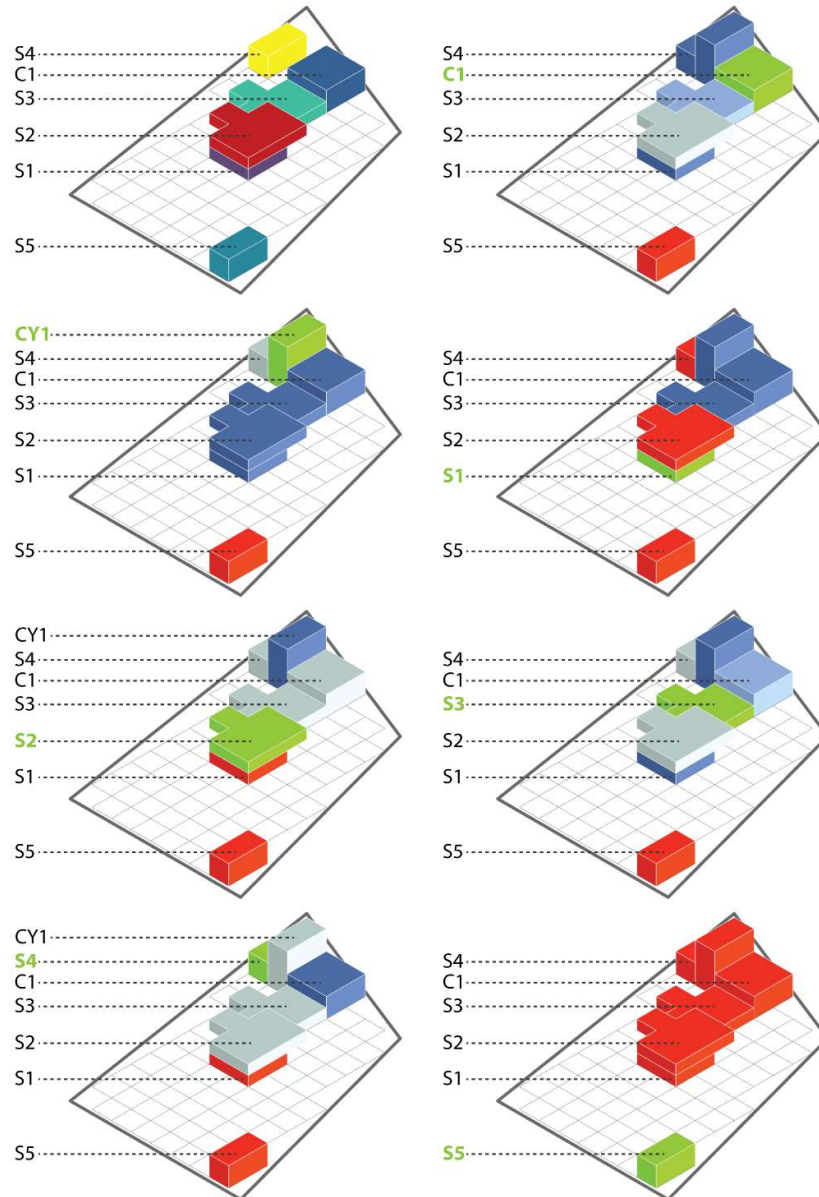


Figure 26 Visualization of Relations Between spaces of The Selected Solution

In the following figure spaces are marked with green and the rest of the spaces marked accordingly the relation to the marked space, red means the space desired to be apart from the green space and if it gets blue color the space is desired to be near green space. Shades of the colors represents the scale of desire. As a result of the relations matrices its clearly shown that S5 is located apart from the main mass of the building since it is not desired to be in near vicinity of none of the spaces. The distances between spaces

shows also similar pattern as relation matrices of spaces to satisfy desired relations as shown in figure 25

Relation Matrices								Distances Between Spaces							
	C1	CY1	S1	S2	S3	S4	S5		C1	CY1	S1	S2	S3	S4	S5
C1	0	3	3	1	2	3	-3	C1	0	1	24	20	1	8	96
CY1	3	0	3	3	3	1	-3	CY1	1	0	32	28	1	1	112
S1	3	3	0	-3	3	-3	-3	S1	24	32	0	84	1	40	48
S2	1	3	-3	0	1	1	-3	S2	20	28	84	0	52	32	48
S3	2	3	3	1	0	1	-3	S3	1	1	1	36	0	8	72
S4	3	1	-3	1	1	0	-3	S4	8	1	40	32	8	0	120
S5	-3	-3	-3	-3	-3	-3	0	S5	96	112	48	48	72	120	0

Figure 27 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)

6.2. CASE2: SEVEN SPACES: DOUBLE CORE

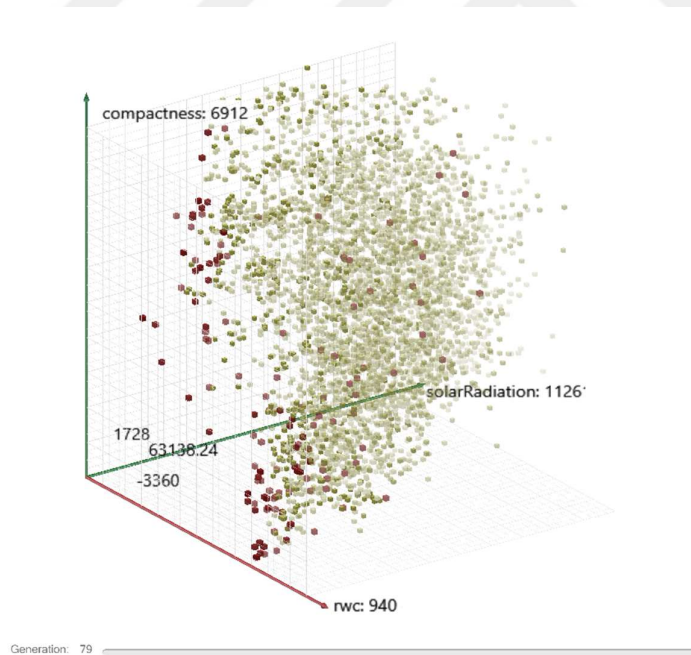


Figure 28 Solution Cluster After 80 Generation with pareto-front (dark red), first front (red) and history (yellow)

After running the algorithm for 80 generation, in total 154 solutions are found 65 of these solutions are non dominated solutions. The cluster of solutions provide design alternatives with different fitness function values. Range of the three fitness function values can be found in the following table.

Table 7 Fitness Function Values of Given Solution Cluster

Fitness Function	Min Value	Max Value
RelationsWeightedCirculation	-3360	-70
Total Solar Radiation	63138	86410
Compactness	310	633

Two design solution is selected and compared against each other in terms of compactness and relation weighted circulation value. Fitness function values of selected design alternatives as follows

Table 8 Fitness Function Values of Selected Solutions from The Cluster

Designation	Relations Weighted Circulation	Total Solar Radiation	Compactness
Alt1	-1652	86058	330
Alt2	-222	66310	280

In the following figure spaces are marked with green and the rest of the spaces marked accordingly the relation to the marked space, red means the space desired to be apart from the green space and if it gets blue color the space is desired to be near green space. Shades of the colors represents the scale of desire. As a result of the relations matrix its clearly shown that S5 is located apart from the main mass of the building since it is not desired to be in near vicinity of none of the spaces. The distances between spaces shows also similar pattern as relation matrices of spaces to satisfy desired relations as shown in figure 30 and figure 32

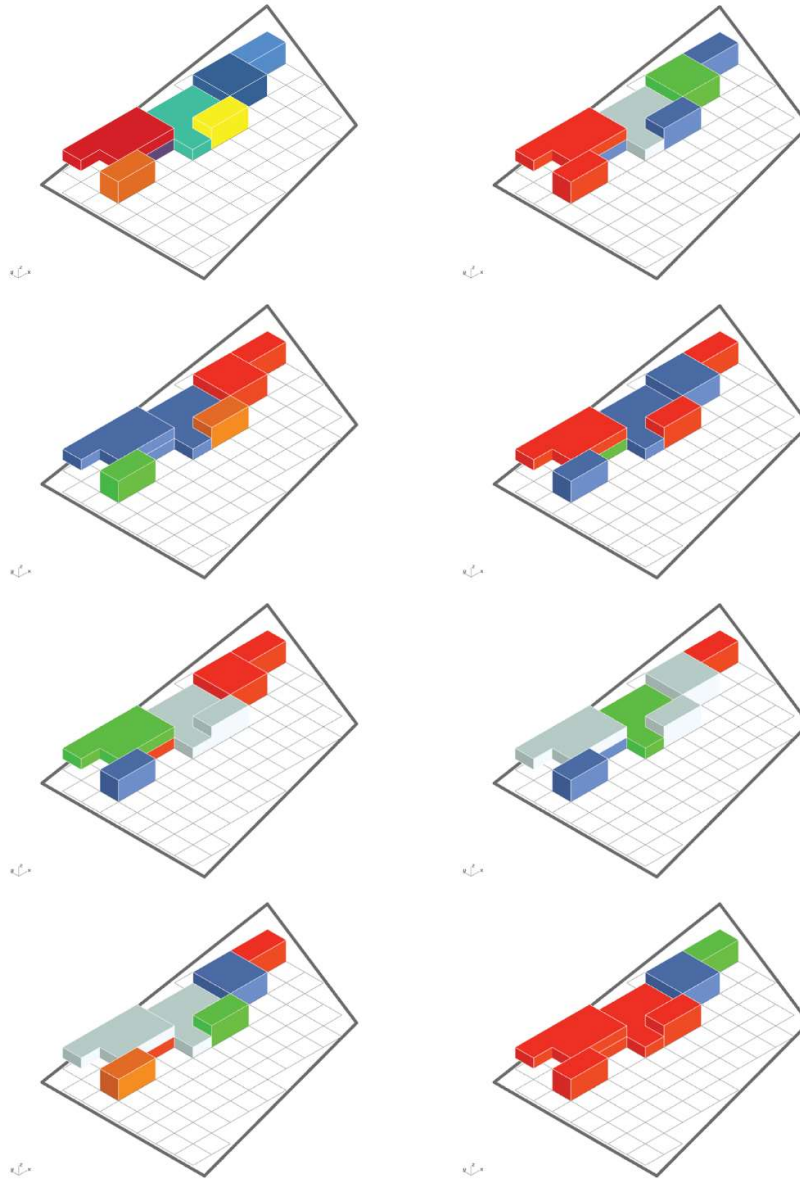


Figure 29 Visualization of Relations Between spaces of The Selected Solution

Relation Matrix								Distances Between Spaces							
	C1	C2	S1	S2	S3	S4	S5		C1	CY1	S1	S2	S3	S4	S5
C1	0	-3	3	-3	1	3	3	C1	0	48	24	28	1	1	1
C2	-3	0	3	3	3	-2	-3	CY1	48	0	1	4	16	24	72
S1	3	3	0	-3	3	-3	-3	S1	24	1	0	4	1	24	40
S2	-3	3	-3	0	1	1	-3	S2	28	4	52	0	52	8	40
S3	1	3	3	1	0	1	-3	S3	1	16	1	36	0	1	16
S4	3	-2	-3	1	1	0	-3	S4	1	40	24	8	1	0	24
S5	3	-3	-3	-3	-3	-3	0	S5	1	72	40	40	16	24	0

Figure 30 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)



Figure 31 Visualization of Relations Between spaces of The Selected Solution

Relation Matrix								Distances Between Spaces							
	C1	C2	S1	S2	S3	S4	S5		C1	CY1	S1	S2	S3	S4	S5
C1	0	-3	3	-3	1	3	3	C1	0	1	1	4	4	1	1
C2	-3	0	3	3	3	-2	-3	CY1	1	0	1	4	4	1	8
S1	3	3	0	-3	3	-3	-3	S1	1	1	0	12	4	1	16
S2	-3	3	-3	0	1	1	-3	S2	4	4	12	0	8	12	1
S3	1	3	3	1	0	1	-3	S3	4	4	20	8	0	20	16
S4	3	-2	-3	1	1	0	-3	S4	1	1	1	12	20	0	16
S5	3	-3	-3	-3	-3	-3	0	S5	1	8	16	1	16	16	0

Figure 32 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)

In this chapter results of spaceexplorer are discussed in terms of its limitations and capabilities of providing different multi level architectural layout plans in two different cases. First one consists of seven spaces consist of various types in terms of relationship and geometry such as core, courtyard, single height space, double height and pass through space to present its results in depth. The second one consists of seven spaces with double cores.

Results of each optimization approach compared against each others in terms of how much they reduce the objective functions after 30 generations. Changes of objective function values and constraint violation count can be seen

6.3. CASE1: SEVEN SPACES: CORE AND COURTYARD

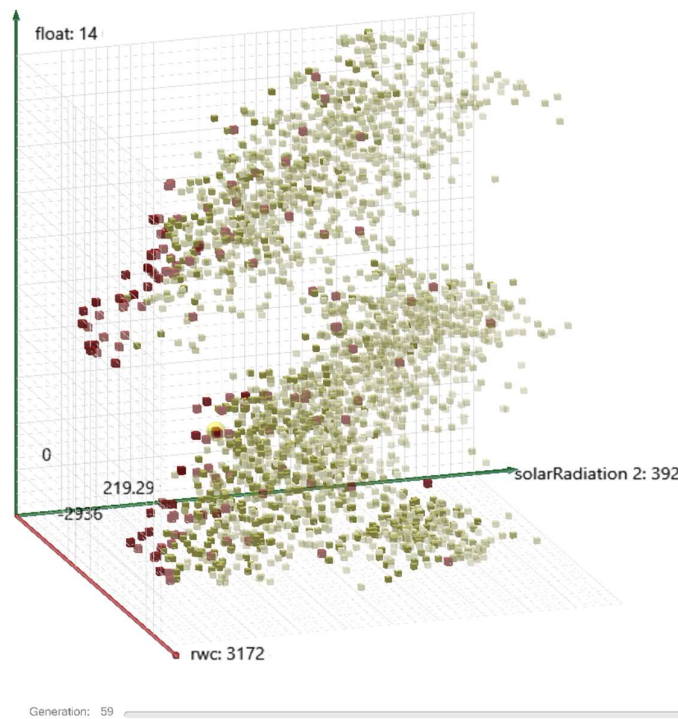


Figure 33 Solution Cluster After 60 Generation with pareto-front (dark red), first front (red) and history (yellow)

After running the algorithm for 60 generation, in total 163 solutions are found 24 of these solutions are non dominated solutions. The cluster of solutions provide design alternatives with different fitness function values. Range of the three fitness function values can be found in the following table.

Table 9 Fitness Function Values of Selected Solutions from The Cluster

Fitness Function	Min Value	Max Value
RelationsWeightedCirculation	-2936	3192
Total Solar Radiation	219.29	392
Floating Voxel Count	0	14

Some example solutions are selected from the cluster of 163 and can be found in the following figure

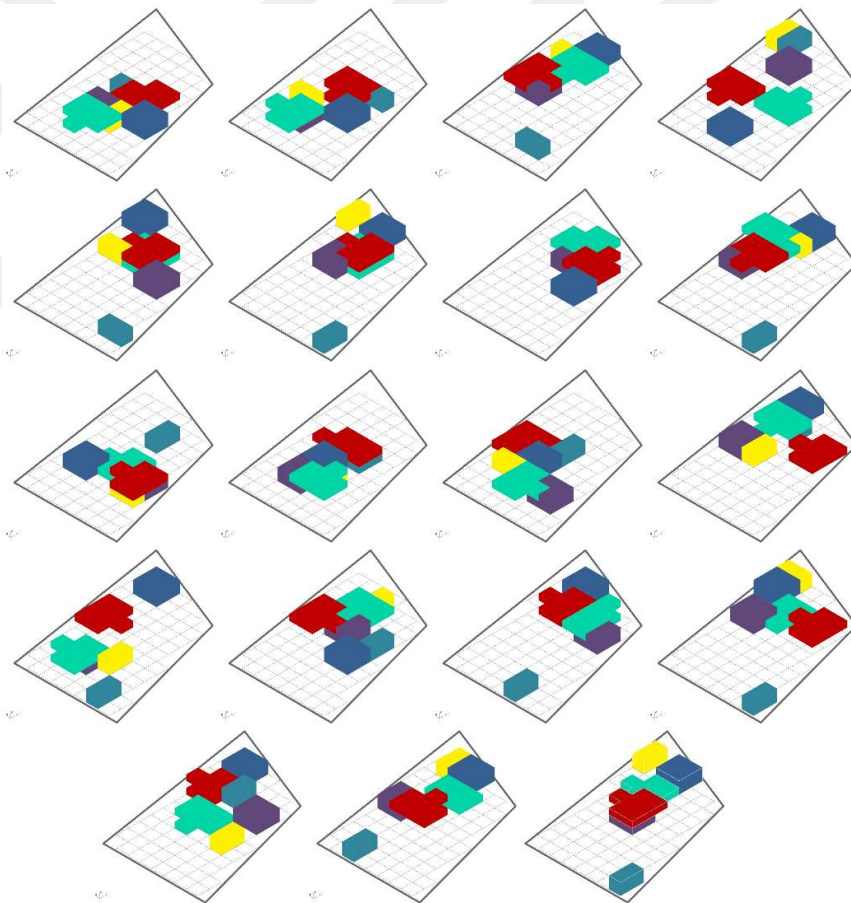


Figure 34 Samples from Solution Cluster

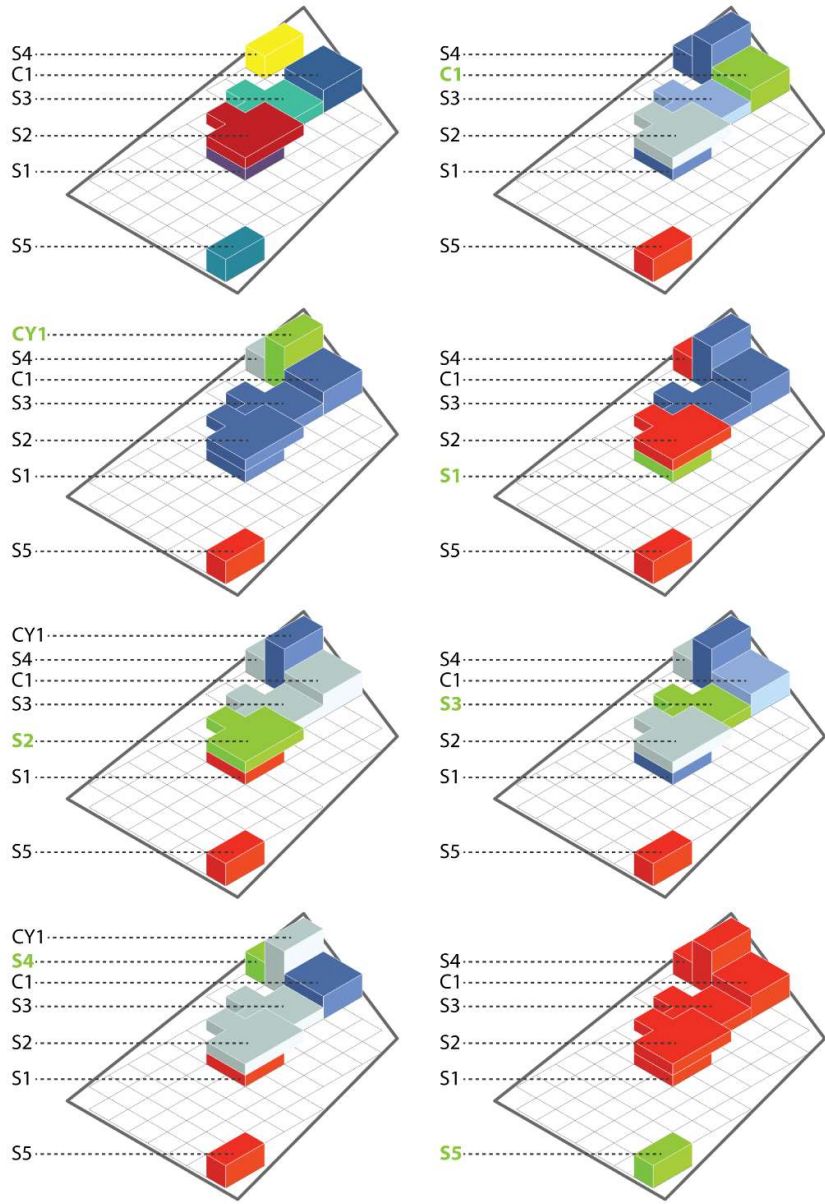


Figure 35 Visualization of Relations Between spaces of The Selected Solution

In the following figure spaces are marked with green and the rest of the spaces marked accordingly the relation to the marked space, red means the space desired to be apart from the green space and if it gets blue color the space is desired to be near green space. Shades of the colors represents the scale of desire. As a result of the relations matrices its clearly shown that S5 is located apart from the main mass of the building since it is not desired to be in near vicinity of none of the spaces. The distances between spaces

shows also similar pattern as relation matrices of spaces to satisfy desired relations as shown in figure 36

Relation Matrices								Distances Between Spaces							
	C1	CY1	S1	S2	S3	S4	S5		C1	CY1	S1	S2	S3	S4	S5
C1	0	3	3	1	2	3	-3	C1	0	1	24	20	1	8	96
CY1	3	0	3	3	3	1	-3	CY1	1	0	32	28	1	1	112
S1	3	3	0	-3	3	-3	-3	S1	24	32	0	84	1	40	48
S2	1	3	-3	0	1	1	-3	S2	20	28	84	0	52	32	48
S3	2	3	3	1	0	1	-3	S3	1	1	1	36	0	8	72
S4	3	1	-3	1	1	0	-3	S4	8	1	40	32	8	0	120
S5	-3	-3	-3	-3	-3	-3	0	S5	96	112	48	48	72	120	0

Figure 36 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)

6.4. CASE2: SEVEN SPACES: DOUBLE CORE

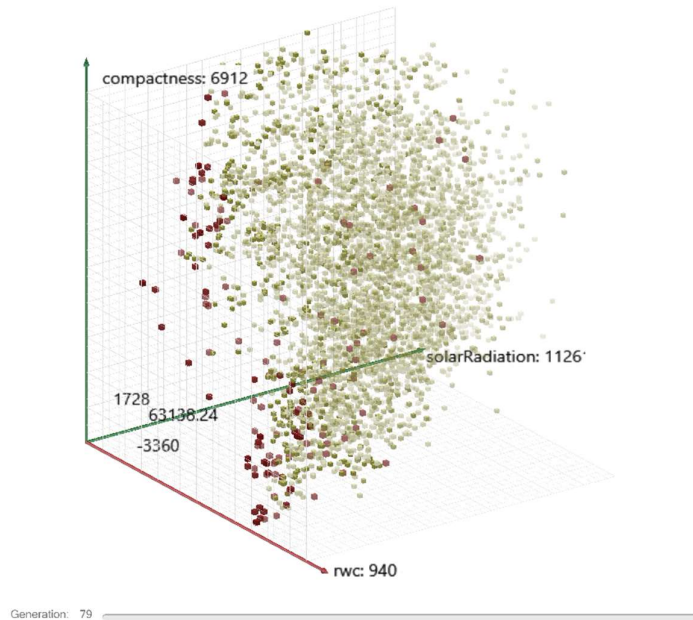


Figure 37 Solution Cluster After 80 Generation with pareto-front (dark red), first front (red) and history (yellow)

After running the algorithm for 80 generation, in total 154 solutions are found 65 of these solutions are non dominated solutions. The cluster of solutions provide design alternatives with different fitness function values. Range of the three fitness function values can be found in the following table.

Table 10 Fitness Function Values of Given Solution Cluster

Fitness Function	Min Value	Max Value
RelationsWeightedCirculation	-3360	-70
Total Solar Radiation	63138	86410
Compactness	310	633

Two design solution is selected and compared against each other in terms of compactness and relation weighted circulation value. Fitness function values of selected design alternatives as follows

Table 11 Fitness Function Values of Selected Solutions from The Cluster

Designation	Relations Weighted Circulation	Total Solar Radiation	Compactness
Alt1	-1652	86058	330
Alt2	-222	66310	280

In the following figure spaces are marked with green and the rest of the spaces marked accordingly the relation to the marked space, red means the space desired to be apart from the green space and if it gets blue color the space is desired to be near green space. Shades of the colors represents the scale of desire. As a result of the relations matrix its clearly shown that S5 is located apart from the main mass of the building since it is not desired to be in near vicinity of none of the spaces. The distances between spaces shows also similar pattern as relation matrices of spaces to satisfy desired relations as shown in figure 39 and 41



Figure 38 Visualization of Relations Between spaces of The Selected Solution

Relation Matrix								Distances Between Spaces							
	C1	C2	S1	S2	S3	S4	S5		C1	CY1	S1	S2	S3	S4	S5
C1	0	-3	3	-3	1	3	3	C1	0	48	24	28	1	1	1
C2	-3	0	3	3	3	-2	-3	CY1	48	0	1	4	16	24	72
S1	3	3	0	-3	3	-3	-3	S1	24	1	0	4	1	24	40
S2	-3	3	-3	0	1	1	-3	S2	28	4	52	0	52	8	40
S3	1	3	3	1	0	1	-3	S3	1	16	1	36	0	1	16
S4	3	-2	-3	1	1	0	-3	S4	1	40	24	8	1	0	24
S5	3	-3	-3	-3	-3	-3	0	S5	1	72	40	40	16	24	0

Figure 39 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)

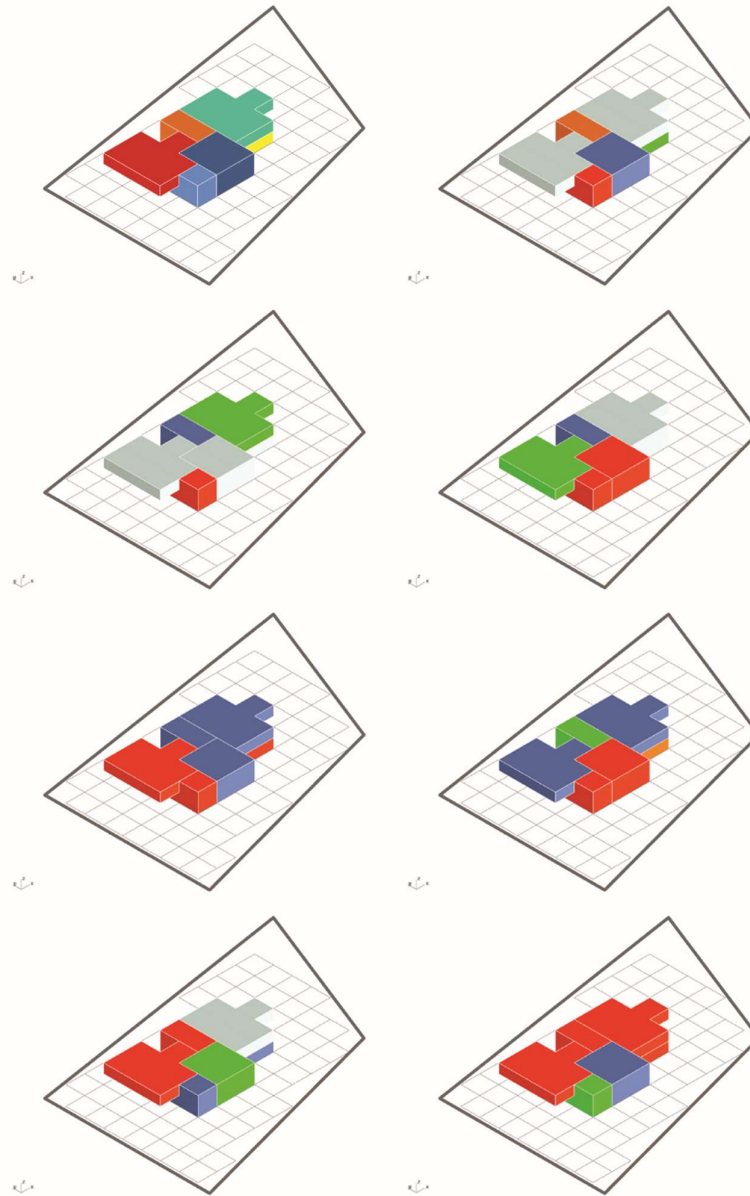


Figure 40 Visualization of Relations Between spaces of The Selected Solution

Relation Matrix								Distances Between Spaces							
	C1	C2	S1	S2	S3	S4	S5		C1	CY1	S1	S2	S3	S4	S5
C1	0	-3	3	-3	1	3	3	C1	0	1	1	4	4	1	1
C2	-3	0	3	3	3	-2	-3	CY1	1	0	1	4	4	1	8
S1	3	3	0	-3	3	-3	-3	S1	1	1	0	12	4	1	16
S2	-3	3	-3	0	1	1	-3	S2	4	4	12	0	8	12	1
S3	1	3	3	1	0	1	-3	S3	4	4	20	8	0	20	16
S4	3	-2	-3	1	1	0	-3	S4	1	1	1	12	20	0	16
S5	3	-3	-3	-3	-3	-3	0	S5	1	8	16	1	16	16	0

Figure 41 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)

In this chapter results of spaceexplorer are discussed in terms of its limitations and capabilities of providing different multi level architectural layout plans in two different cases. First one consists of seven spaces consist of various types in terms of relationship and geometry such as core, courtyard, single height space, double height and pass through space to present its results in depth. The second one consists of seven spaces with double cores.

Results of each optimization approach compared against each others in terms of how much they reduce the objective functions after 30 generations. Changes of objective function values and constraint violation count can be seen

6.5. CASE1: SEVEN SPACES: CORE AND COURTYARD

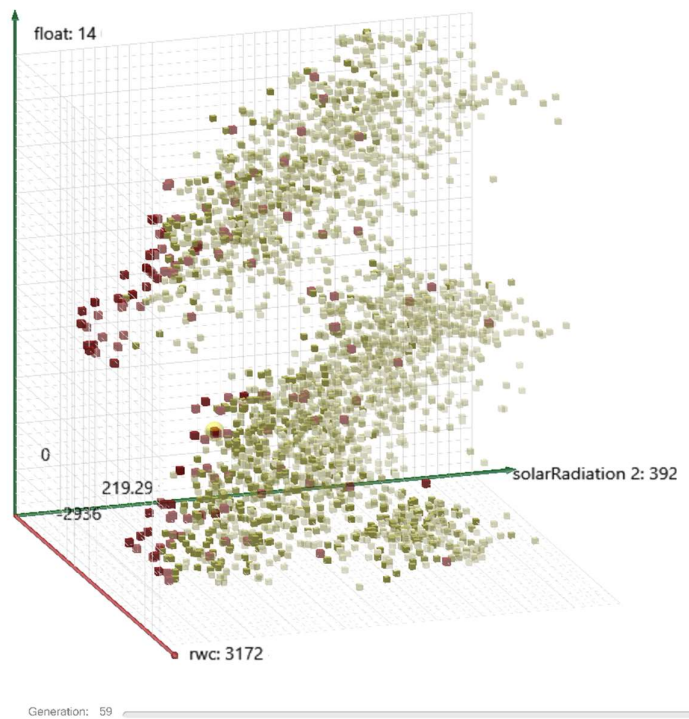


Figure 42 Solution Cluster After 60 Generation with pareto-front (dark red), first front (red) and history (yellow)

After running the algorithm for 60 generation, in total 163 solutions are found 24 of these solutions are non dominated solutions. The cluster of solutions provide deisgn alternatives with different fitness function values. Range of the three fitness function values can be found in the following table.

Table 12 Fitness Function Values of Selected Solutions from The Cluster

Fitness Function	Min Value	Max Value
RelationsWeightedCirculation	-2936	3192
Total Solar Radiation	219.29	392
Floating Voxel Count	0	14

Some example solutions are selected from the cluster of 163 and can be found in the following figure

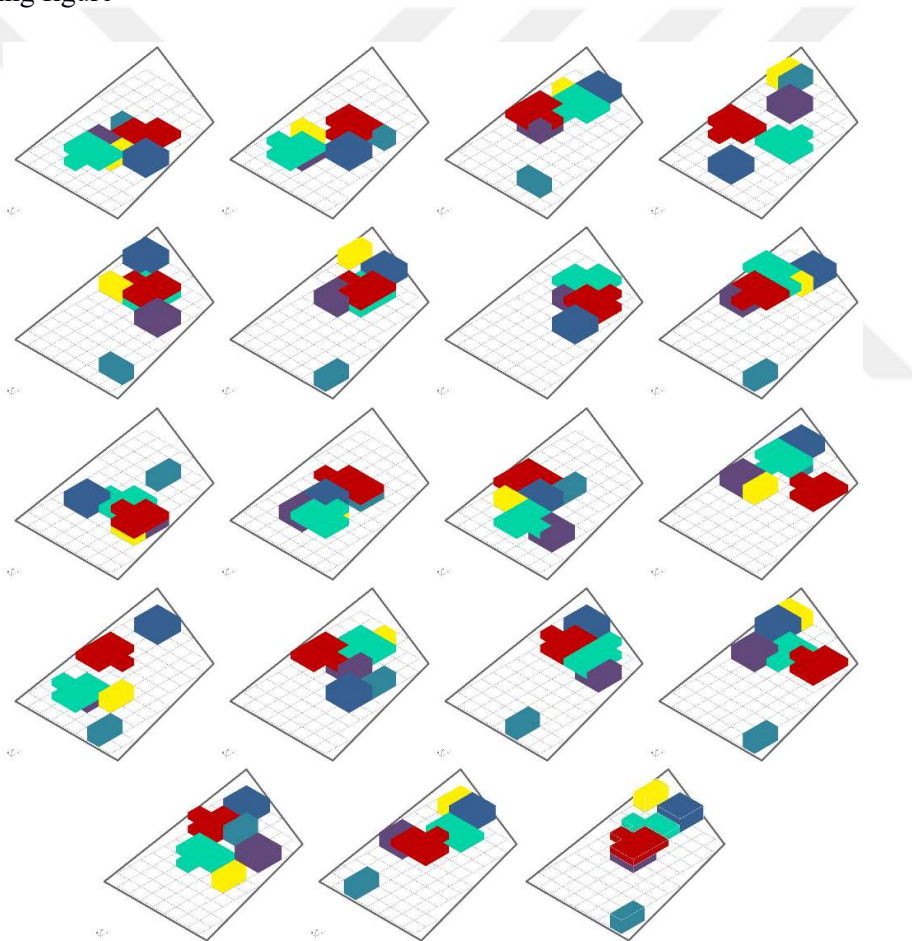


Figure 43 Samples from Solution Cluster

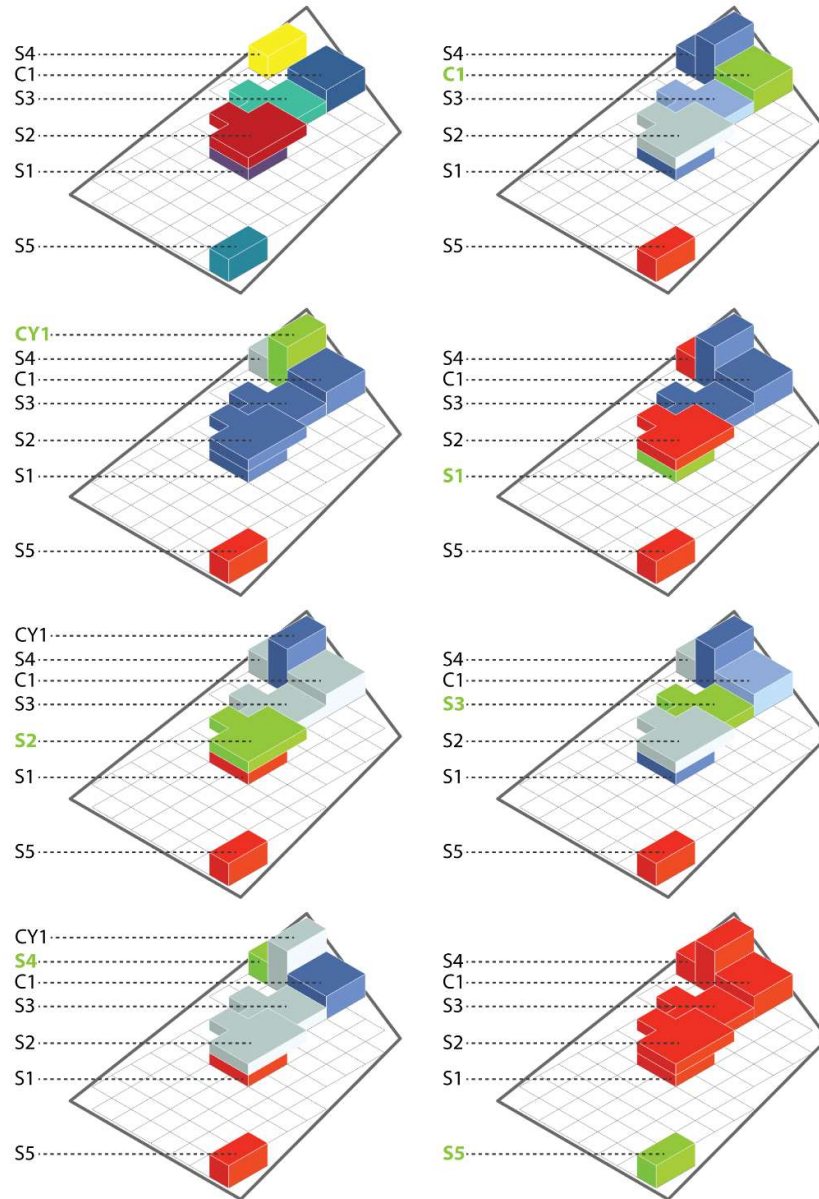


Figure 44 Visualization of Relations Between spaces of The Selected Solution

In the following figure spaces are marked with green and the rest of the spaces marked accordingly the relation to the marked space, red means the space desired to be apart from the green space and if it gets blue color the space is desired to be near green space. Shades of the colors represents the scale of desire. As a result of the relations matrices its clearly shown that S5 is located apart from the main mass of the building since it is not desired to be in near vicinity of none of the spaces. The distances between spaces

shows also similar pattern as relation matrices of spaces to satisfy desired relations as shown in figure XX

Relation Matrices								Distances Between Spaces							
	C1	CY1	S1	S2	S3	S4	S5		C1	CY1	S1	S2	S3	S4	S5
C1	0	3	3	1	2	3	-3	C1	0	1	24	20	1	8	96
CY1	3	0	3	3	3	1	-3	CY1	1	0	32	28	1	1	112
S1	3	3	0	-3	3	-3	-3	S1	24	32	0	84	1	40	48
S2	1	3	-3	0	1	1	-3	S2	20	28	84	0	52	32	48
S3	2	3	3	1	0	1	-3	S3	1	1	1	36	0	8	72
S4	3	1	-3	1	1	0	-3	S4	8	1	40	32	8	0	120
S5	-3	-3	-3	-3	-3	-3	0	S5	96	112	48	48	72	120	0

Figure 45 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)

6.6. CASE2: SEVEN SPACES: DOUBLE CORE

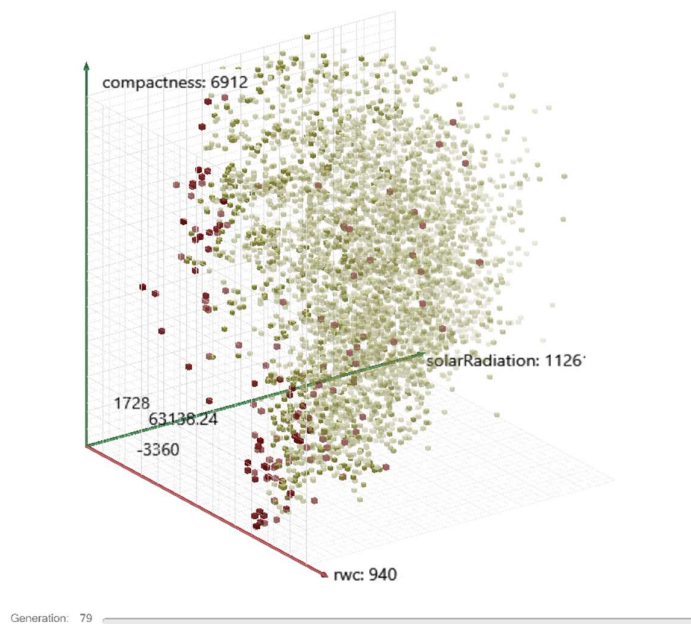


Figure 46 Solution Cluster After 80 Generation with pareto-front (dark red), first front (red) and history (yellow)

After running the algorithm for 80 generation, in total 154 solutions are found 65 of these solutions are non dominated solutions. The cluster of solutions provide design alternatives with different fitness function values. Range of the three fitness function values can be found in the following table.

Table 13 Fitness Function Values of Given Solution Cluster

Fitness Function	Min Value	Max Value
RelationsWeightedCirculation	-3360	-70
Total Solar Radiation	63138	86410
Compactness	310	633

Two design solution is selected and compared against each other in terms of compactness and relation weighted circulation value. Fitness function values of selected design alternatives as follows

Table 14 Fitness Function Values of Selected Solutions from The Cluster

Designation	Relations Weighted Circulation	Total Solar Radiation	Compactness
Alt1	-1652	86058	330
Alt2	-222	66310	280

In the following figure spaces are marked with green and the rest of the spaces marked accordingly the relation to the marked space, red means the space desired to be apart from the green space and if it gets blue color the space is desired to be near green space. Shades of the colors represents the scale of desire. As a result of the relations matrix its clearly shown that S5 is located apart from the main mass of the building since it is not desired to be in near vicinity of none of the spaces. The distances between spaces shows also similar pattern as relation matrices of spaces to satisfy desired relations as shown in figure 48 and figure 50

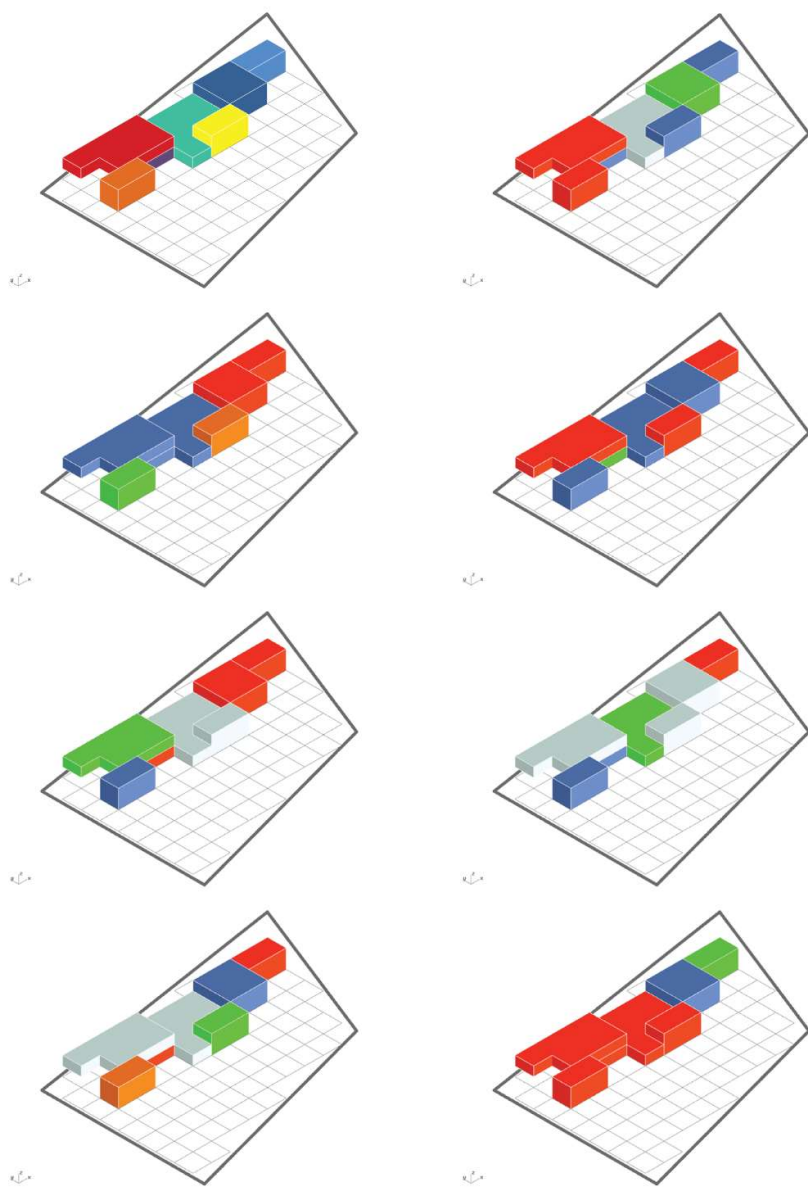


Figure 47 Visualization of Relations Between spaces of The Selected Solution

Relation Matrix								Distances Between Spaces							
	C1	C2	S1	S2	S3	S4	S5		C1	CY1	S1	S2	S3	S4	S5
C1	0	-3	3	-3	1	3	3	C1	0	48	24	28	1	1	1
C2	-3	0	3	3	3	-2	-3	CY1	48	0	1	4	16	24	72
S1	3	3	0	-3	3	-3	-3	S1	24	1	0	4	1	24	40
S2	-3	3	-3	0	1	1	-3	S2	28	4	52	0	52	8	40
S3	1	3	3	1	0	1	-3	S3	1	16	1	36	0	1	16
S4	3	-2	-3	1	1	0	-3	S4	1	40	24	8	1	0	24
S5	3	-3	-3	-3	-3	-3	0	S5	1	72	40	40	16	24	0

Figure 48 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)



Figure 49 Visualization of Relations Between spaces of The Selected Solution

Relation Matrix								Distances Between Spaces							
	C1	C2	S1	S2	S3	S4	S5		C1	CY1	S1	S2	S3	S4	S5
C1	0	-3	3	-3	1	3	3	C1	0	1	1	4	4	1	1
C2	-3	0	3	3	3	-2	-3	CY1	1	0	1	4	4	1	8
S1	3	3	0	-3	3	-3	-3	S1	1	1	0	12	4	1	16
S2	-3	3	-3	0	1	1	-3	S2	4	4	12	0	8	12	1
S3	1	3	3	1	0	1	-3	S3	4	4	20	8	0	20	16
S4	3	-2	-3	1	1	0	-3	S4	1	1	1	12	20	0	16
S5	3	-3	-3	-3	-3	-3	0	S5	1	8	16	1	16	16	0

Figure 50 Relation Matrices(left), Distances Between Spaces of the Selected Solution(right)

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH

According to the result of this study it is shown that genetic algorithm based multi-level architectural layout planning can be achieved in Rhino Grasshopper parametric design environment to guide architect to find design alternatives with in huge search space in early design phase which may be developed fully by using spaceexplorer.

Due to complexity of layout design task it is not possible to configure an ultimate generative model for all architectural layout problems. Therefore, configuring generative model in modular fashion let the architect mix and match related modules and only configure the lacking modules instead of modeling whole model from scratch for specific architectural design problem.

It is shown that space explorer is capable of solving multi level architectural layout design problems with irregular size for building functions, single or multiple cores and courtyard.

Space explorer's ability to generate and evaluate design alternatives in a reasonable time may let architect explore and consider design alternatives which may not be for seen with different spatial qualities. Such as producing separate building masses if the relations matrix requires

While calculating distance between spaces, circulation-based metrics produce much more reasonable results than Euclidian based distances since circulation-based metric is calculated according to the possible circulation paths that spatial organization allows, rather than calculating direct theoretical distances between center points of spaces.

Total solar gain objective helps to find optimum placement of spaces in building plot in relation to the site object that may cast shadow and affect the thermal performance of the building.

Having double height spaces also let more sophisticated architectural configurations that may increase spatial quality.

For the further studies as mentioned other aspect of architectural spatial qualities can be objectified to be used as optimization goals. Site conditions can be defined such as topography, which also affects how building masses can be placed and create additional problems in terms of circulation, underground and semi underground spaces.

It is possible to utilize this approach while making spatial organization of a mix use high rise building. By trying to minimize vertical circulation requirement by placing related spaces close vicinity not only that but also it can guide architect in terms sizing of circulation elements.

In addition to that, this approach can also be applied for adaptive reuse of an existing building with small modifications to how voxel space is created by considering the existing structural system of the building while creating voxels and selection points. Therefore, generated adapted design has less conflicts with the existing conditions.

REFERENCES

- Bahreghmand, A., Batard, T., Marques, R., Evans, A., & Blat, J. (2017). Optimizing layout using spatial quality metrics and user preferences. *Graphical Models*, 93, 25–38. <https://doi.org/10.1016/j.gmod.2017.08.003>
- Baušys, R., & Pankrašovaitė, I. (2005). Optimization of architectural layout by the improved genetic algorithm. *Journal of Civil Engineering and Management*, 11(1), 13–21.
- Baykan, C. (2003). Spatial Relations and Architectural Plans. *E-Activities in Design and Education*. Retrieved from <http://www.metu.edu.tr/~baykan/publications-pdf/baykan-eia9.pdf>
- Bader, J., & Zitzler, E. (2010). HypE: An Algorithm for Fast Hypervolume-Based Many-Objective Optimization. *Evolutionary Computation*, 19(1), 45 – 76. https://doi.org/10.1162/EVCO_a_00009
- Chatzikonstantinou, I. (2014). A 3-Dimensional Architectural Layout Generation Procedure for Optimization Applications : DC-RVD. *Proceedings of the 2014 ECAADe Conference*, 1, 287–296. Retrieved from http://cumincad.scix.net/cgi-bin/works/Show?_id=ecaade2014_163&sort=DEFAULT&search=ioannis&hits=7
- Choudhary, R., & Michalek, J. (2005). Design optimization in computer-aided architectural design. *CAADRIA 2005 - The Association for Computer-Aided Architectural Design Research in Asia: Digital Opportunities*, 149–159.
- Dino, I. G. (2016). An evolutionary approach for 3D architectural space layout design exploration. *Automation in Construction*, 69, 131–150. <https://doi.org/10.1016/j.autcon.2016.05.020>
- Drira, A., Pierreval, H., & Hajri-Gabouj, S. (2007). Facility layout problems: A survey. *Annual Reviews in Control*, 31(2), 255–267. <https://doi.org/10.1016/j.arcontrol.2007.04.001>
- Du, T., Turrin, M., Jansen, S., Dobbelsteen, A. Van Den, & Bioria, N. (2018). A Review on Automatic Generation of Architectural Space Layouts with Energy Performance Optimization A Review on Automatic Generation of Architectural Space Layouts with Energy Performance Optimization. *International Conference On Building Energy, Environment*, (February), 856–861. Retrieved from <http://www.cobee2018.net/assets/pdf/p/283.pdf>
- Hathhorn, J., Sisikoglu, E., & Sir, M. Y. (2013). A multi-objective mixed-integer programming model for a multi-floor facility layout. *International Journal of Production Research*, 51(14), 4223–4239. <https://doi.org/10.1080/00207543.2012.753486>
- Hornby, G., Globus, A., Linden, D., & Lohn, J. (2012). *Automated Antenna Design with Evolutionary Algorithms*. 5, 1–8. <https://doi.org/10.2514/6.2006-7242>
- Ismail, F. S., Yusof, R., & Khalid, M. (2012). Optimization of electronics component placement design on PCB using self organizing genetic algorithm (SOGA). *Journal of Intelligent Manufacturing*, 23(3), 883–895. <https://doi.org/10.1007/s10845-010-0444-x>
- Jo, J. H., & Gero, J. S. (1998). Space layout planning using an evolutionary approach. *Artificial Intelligence in Engineering*, 12(3), 149–162. [https://doi.org/10.1016/S0954-1810\(97\)00037-X](https://doi.org/10.1016/S0954-1810(97)00037-X)

- Korde, M. R., & Shahare, A. (2017). Design and Development of Simulation Model for Plant Layout. *IJSTE - International Journal of Science Technology & Engineering* |, 3(09), 446–449.
- Lin, C. J. (2005). Space Layout Game: An interactive game of space layout for teaching and representing design knowledge. *CAADRIA 2005 - The Association for Computer-Aided Architectural Design Research in Asia: Digital Opportunities*, 130–141.
- Lobos, D., & Donath, D. (2011). The problem of space layout in architecture: A survey and reflections. *Arquitectura Revista*, 6(2), 136–161.
<https://doi.org/10.4013/arq.2010.62.05>
- Menges, A., & Ahlquist, S. (2011). *Computational design thinking: computation design thinking*. John Wiley & Sons.
- Michalek, J. J., Choudhary, R., & Papalambros, P. Y. (2002). Architectural layout design optimization. *Engineering Optimization*, 34(5), 461–484.
<https://doi.org/10.1080/03052150214016>
- Nagy, D., Lau, D., Locke, J., Stoddart, J., Villaggi, L., Wang, R., ... Benjamin, D. (2018). *Project Discover: An Application of Generative Design for Architectural Space Planning*. 59–66.
<https://doi.org/10.22360/simaud.2017.simaud.007>
- Peckiene, A., & Ustinovičius, L. (2017). Possibilities for Building Spatial Planning using BIM Methodology. *Procedia Engineering*, 172, 851–858.
<https://doi.org/10.1016/j.proeng.2017.02.085>
- Preisinger, C., & Heimrath, M. (2014). Karamba - A toolkit for parametric structural design. *Structural Engineering International: Journal of the International Association for Bridge and Structural Engineering (IABSE)*.
<https://doi.org/10.2749/101686614X13830790993483>
- Rodrigues, E., Gaspar, A. R., & Gomes, Á. (2013). An approach to the multi-level space allocation problem in architecture using a hybrid evolutionary technique. *Automation in Construction*, 35, 482–498.
<https://doi.org/10.1016/j.autcon.2013.06.005>
- Sadeghipour Roudsari, M., Pak, M., & Smith, A. (2013). Ladybug: a Parametric Environmental Plugin for Grasshopper To Help Designers Create an Environmentally-Conscious Design. *13th Conference of International Building Performance Simulation Association*, 3129–3135.
- Saraswat, A., Venkatadri, U., & Castillo, I. (2015). A framework for multi-objective facility layout design. *Computers and Industrial Engineering*, 90, 167–176.
<https://doi.org/10.1016/j.cie.2015.09.006>
- Sharma, D., Chattopadhyay, C., & Harit, G. (2016). A unified framework for semantic matching of architectural floorplans. *Proceedings - International Conference on Pattern Recognition*, 0, 2422–2427. <https://doi.org/10.1109/ICPR.2016.7899999>
- Skandhakumar, N., Salim, F., Reid, J., Drogemuller, R., & Dawson, E. (2016). Graph theory based representation of building information models for access control applications. *Automation in Construction*, 68, 44–51.
<https://doi.org/10.1016/J.AUTCON.2016.04.001>

- Stage, E. D. (2009). the Architect As Performer of Energy Simulation in the Performance Based Design. *Building Simulation 2009*.
- Sönmez, N. O. (2015). Architectural layout evolution through similarity-based evaluation. *International Journal of Architectural Computing*, 13(3–4), 271–297. <https://doi.org/10.1260/1478-0771.13.3-4.271>
- Verma, M., & Thakur, M. K. (2010). Architectural space planning using genetic algorithms. *2010 The 2nd International Conference on Computer and Automation Engineering, ICCAE 2010*, 2, 268–275. <https://doi.org/10.1109/ICCAE.2010.5451497>
- Yeh, I. C. (2006). Architectural layout optimization using annealed neural network. *Automation in Construction*, 15(4), 531–539. <https://doi.org/10.1016/j.autcon.2005.07.002>
- Yi, Y. K., & Malkawi, A. M. (2009). Optimizing building form for energy performance based on hierarchical geometry relation. *Automation in Construction*, 18(6), 825–833. <https://doi.org/10.1016/j.autcon.2009.03.006>
- Yi, H., & Yi, Y. K. (2014). Performance based architectural design optimization: Automated 3D space layout using simulated annealing. *2014 ASHRAE/IBPSA-USA Building Simulation Conference*, (October), 292–299.
- Žerovnik, J. (2015). Heuristics for NP-hard optimization problems - simpler is better!?. *Logistics & Sustainable Transport*, 6(1), 1–10. <https://doi.org/10.1515/jlst-2015-0006>
- Zitzler, E., Laumanns, M., & Thiele, L. (2014). SPEA2: Improving the Strength Pareto Evolutionary Algorithm. *Igarss 2014*. <https://doi.org/10.1007/s13398-014-0173-7.2>

APPENDIX 1 – FLOOR SELECTION COMPONENT C# CODE

```
using Rhino;
using Rhino.Geometry;
using Rhino.DocObjects;
using Rhino.Collections;

using GH_IO;
using GH_IO.Serialization;
using Grasshopper;
using Grasshopper.Kernel;
using Grasshopper.Kernel.Data;
using Grasshopper.Kernel.Types;

using System;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using System.Linq;
using System.Data;
using System.Drawing;
using System.Reflection;
using System.Collections;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Runtime.InteropServices;

/// <summary>
/// This class will be instantiated on demand by the Script
/// component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
    #region Utility functions
        /// <summary>Print a String to the [Out] Parameter of the Script
        /// component.</summary>
        /// <param name="text">String to print.</param>
        private void Print(string text) { /* Implementation hidden. */ }
        /// <summary>Print a formatted String to the [Out] Parameter of
        /// the Script component.</summary>
        /// <param name="format">String format.</param>
        /// <param name="args">Formatting parameters.</param>
        private void Print(string format, params object[] args) { /*
        Implementation hidden. */ }
        /// <summary>Print useful information about an object instance to
        /// the [Out] Parameter of the Script component. </summary>
        /// <param name="obj">Object instance to parse.</param>
        private void Reflect(object obj) { /* Implementation hidden. */ }
        /// <summary>Print the signatures of all the overloads of a
        /// specific method to the [Out] Parameter of the Script component.
        </summary>
        /// <param name="obj">Object instance to parse.</param>
        private void Reflect(object obj, string method_name) { /*
        Implementation hidden. */ }
    #endregion
}
```

```

#region Members
    /// <summary>Gets the current Rhino document.</summary>
    private readonly RhinoDoc RhinoDocument;
    /// <summary>Gets the Grasshopper document that owns this
script.</summary>
    private readonly GH_Document GrasshopperDocument;
    /// <summary>Gets the Grasshopper script component that owns this
script.</summary>
    private readonly IGH_Component Component;
    /// <summary>
    /// Gets the current iteration count. The first call to
RunScript() is associated with Iteration==0.
    /// Any subsequent call within the same solution will increment
the Iteration count.
    /// </summary>
    private readonly int Iteration;
#endregion

    /// <summary>
    /// This procedure contains the user code. Input parameters are
provided as regular arguments,
    /// Output parameters as ref arguments. You don't have to assign
output parameters,
    /// they will have a default value.
    /// </summary>
    private void RunScript(bool create, bool modify, int min, int max,
int count, List<Point3d> data, List<int> floor, List<object>
spacenames, List<int> x, ref object A, ref object B)
    {

        Random rnd = new Random();

        if(modify)
        { var input = Component.Params.Input[8].Sources[0];

            List<System.Guid> guids = new List<System.Guid>();
            Grasshopper.Kernel.IGH_Param selSlidersInput =
Component.Params.Input[8];
            IList<Grasshopper.Kernel.IGH_Param> sources =
selSlidersInput.Sources;
            bool isMySlidersEmpty = !sources.Any();

            if (!isMySlidersEmpty) {
                foreach (var source in sources)
                {
                    IGH_DocumentObject component =
source.Attributes.GetTopLevel.DocObject;
                    Grasshopper.Kernel.Special.GH_NumberSlider mySlider =
component as Grasshopper.Kernel.Special.GH_NumberSlider;
                    if (mySlider == null)
                        continue;
                    guids.Add(mySlider.InstanceGuid);
                }
            }
            // Find all sliders.
            List<Grasshopper.Kernel.Special.GH_NumberSlider> sliders = new
List<Grasshopper.Kernel.Special.GH_NumberSlider>();
            foreach (IGH_DocumentObject docObject in
GrasshopperDocument.Objects)
            {

```

```

        Grasshopper.Kernel.Special.GH_NumberSlider slider =
docObject as Grasshopper.Kernel.Special.GH_NumberSlider;
        if (slider != null)
        {
            // check if the slider is in the selected list
            if (!isMySlidersEmpty)
            {
                if (guids.Contains(slider.InstanceGuid))
sliders.Add(slider);
            }
            else
            {
                sliders.Add(slider);
            }
        }
    }
    if (sliders.Count == 0)
    {
        System.Windows.Forms.MessageBox.Show("No sliders could be
found, create sliders first!", "erinc<3oykù", MessageBoxButtons.OK);
        return;
    }
    foreach (Grasshopper.Kernel.Special.GH_NumberSlider slider in
sliders)
        slider.Slider.Minimum = min;

    foreach (Grasshopper.Kernel.Special.GH_NumberSlider slider in
sliders)
        slider.Slider.Maximum = max;

    foreach (Grasshopper.Kernel.Special.GH_NumberSlider slider in
sliders)
        slider.SetSliderValue(max);

}

if(create)
{
    for (int i = 0; i < count; i++)
    {

        //instantiate new slider
        Grasshopper.Kernel.Special.GH_NumberSlider slid = new
Grasshopper.Kernel.Special.GH_NumberSlider();
        slid.CreateAttributes(); //sets up default values, and makes
sure your slider doesn't crash rhino

        //customise slider (position, ranges etc)
        int inputcount = this.Component.Params.Input[0].SourceCount;
        slid.Attributes.Pivot = new PointF((float)
this.Component.Attributes.DocObject.Attributes.Bounds.Left -
slid.Attributes.Bounds.Width - 30, (float)
this.Component.Params.Input[8].Attributes.Bounds.Y + 30 + i *
30); //inputcount * 30);
        slid.NickName = spacenames[i] + "_XY";
        slid.Slider.Maximum = max;
    }
}

```

```

    slid.Slider.Minimum = min;
    slid.Slider.DecimalPlaces = 0;
    slid.SetSliderValue((decimal) min);

    //Until now, the slider is a hypothetical object.
    // This command makes it 'real' and adds it to the canvas.
    GrasshopperDocument.AddObject(slid, false);

    //Connect the new slider to this component
    this.Component.Params.Input[8].AddSource(slid);
}
}

List<int> spaceindexlist = new List<int>();

for (int i = 0; i < count; i++)
{
    int spaceindex = x[i];
    spaceindexlist.Add(spaceindex);

    // A = spaceindexlist;
}

List<Point3d> selectedpointslist = new List<Point3d>();
int floorcount;

for (int i = 0; i < count; i++)
{
    floorcount = floor[i];
    Point3d selectedpoints = data[spaceindexlist[i] + ((max + 1) *
floorcount)];
    selectedpointslist.Add(selectedpoints);
}
// Point3d selectedpoints = data[23];
//

A = selectedpointslist;
B = floor[2];

}

// <Custom additional code>

// </Custom additional code>
}

```

APPENDIX 2 – LOCATION SELECTION COMPONENT C# CODE

```
using Rhino;
using Rhino.Geometry;
using Rhino.DocObjects;
using Rhino.Collections;

using GH_IO;
using GH_IO.Serialization;
using Grasshopper;
using Grasshopper.Kernel;
using Grasshopper.Kernel.Data;
using Grasshopper.Kernel.Types;

using System;
using System.IO;
using System.Xml;
using System.Xml.Linq;
using System.Linq;
using System.Data;
using System.Drawing;
using System.Reflection;
using System.Collections;
using System.Windows.Forms;
using System.Collections.Generic;
using System.Runtime.InteropServices;

/// <summary>
/// This class will be instantiated on demand by the Script
component.
/// </summary>
public class Script_Instance : GH_ScriptInstance
{
    #region Utility functions
        /// <summary>Print a String to the [Out] Parameter of the Script
component.</summary>
        /// <param name="text">String to print.</param>
        private void Print(string text) { /* Implementation hidden. */ }
        /// <summary>Print a formatted String to the [Out] Parameter of
the Script component.</summary>
        /// <param name="format">String format.</param>
        /// <param name="args">Formatting parameters.</param>
        private void Print(string format, params object[] args) { /*
Implementation hidden. */ }
        /// <summary>Print useful information about an object instance to
the [Out] Parameter of the Script component. </summary>
        /// <param name="obj">Object instance to parse.</param>
        private void Reflect(object obj) { /* Implementation hidden. */ }
        /// <summary>Print the signatures of all the overloads of a
specific method to the [Out] Parameter of the Script component.
</summary>
        /// <param name="obj">Object instance to parse.</param>
        private void Reflect(object obj, string method_name) { /*
Implementation hidden. */ }
    #endregion
}
```

```

#region Members
    /// <summary>Gets the current Rhino document.</summary>
    private readonly RhinoDoc RhinoDocument;
    /// <summary>Gets the Grasshopper document that owns this
script.</summary>
    private readonly GH_Document GrasshopperDocument;
    /// <summary>Gets the Grasshopper script component that owns this
script.</summary>
    private readonly IGH_Component Component;
    /// <summary>
    /// Gets the current iteration count. The first call to
RunScript() is associated with Iteration==0.
    /// Any subsequent call within the same solution will increment
the Iteration count.
    /// </summary>
    private readonly int Iteration;
#endregion

    /// <summary>
    /// This procedure contains the user code. Input parameters are
provided as regular arguments,
    /// Output parameters as ref arguments. You don't have to assign
output parameters,
    /// they will have a default value.
    /// </summary>
    private void RunScript(bool create, bool modify, int min, int
floorcount, int count, List<Point3d> data, List<string> spacenames,
List<bool> groundconstraint, List<int> x, ref object A)
    {
        Random rnd = new Random();

        if(modify)
        { var input = Component.Params.Input[5].Sources[0];

            List<System.Guid> guids = new List<System.Guid>();
            Grasshopper.Kernel.IGH_Param selSlidersInput =
Component.Params.Input[5];
            IList<Grasshopper.Kernel.IGH_Param> sources =
selSlidersInput.Sources;
            bool isMySlidersEmpty = !sources.Any();

            if (!isMySlidersEmpty) {
                foreach (var source in sources)
                {
                    IGH_DocumentObject component =
source.Attributes.GetTopLevel.DocObject;
                    Grasshopper.Kernel.Special.GH_NumberSlider mySlider =
component as Grasshopper.Kernel.Special.GH_NumberSlider;
                    if (mySlider == null)
                        continue;
                    guids.Add(mySlider.InstanceGuid);
                }
            }
            // Find all sliders.
            List<Grasshopper.Kernel.Special.GH_NumberSlider> sliders = new
List<Grasshopper.Kernel.Special.GH_NumberSlider>();
            foreach (IGH_DocumentObject docObject in
GrasshopperDocument.Objects)
            {
                Grasshopper.Kernel.Special.GH_NumberSlider slider =

```



```

docObject as Grasshopper.Kernel.Special.GH_NumberSlider;
    if (slider != null)
    {
        // check if the slider is in the selected list
        if (!isMySlidersEmpty)
        {
            if (guids.Contains(slider.InstanceGuid))
sliders.Add(slider);
        }
        else
        {
            sliders.Add(slider);
        }
    }
}
if (sliders.Count == 0)
{
    System.Windows.Forms.MessageBox.Show("No sliders could be
found, create sliders first!", "erinc<3oykù", MessageBoxButtons.OK);
    return;
}
foreach (Grasshopper.Kernel.Special.GH_NumberSlider slider in
sliders)
    slider.Slider.Minimum = min;

    foreach (Grasshopper.Kernel.Special.GH_NumberSlider slider in
sliders)
        slider.Slider.Maximum = floorcount - 1;

    foreach (Grasshopper.Kernel.Special.GH_NumberSlider slider in
sliders)
        slider.SetSliderValue(floorcount - 1);

}

if(create)
{
    for (int i = 0; i < count; i++)
    {

        //instantiate new slider

        //customise slider (position, ranges etc)

        //slid.Slider.GripWidth = 200;

        if(groundconstraint[i])
        {
            Grasshopper.Kernel.Special.GH_Panel pan = new
Grasshopper.Kernel.Special.GH_Panel();
            pan.CreateAttributes();
            pan.Attributes.Pivot = new PointF((float)
this.Component.Attributes.DocObject.Attributes.Bounds.Right +
pan.Attributes.Bounds.Width / 4, (float)
this.Component.Params.Input[7].Attributes.Bounds.Bottom + (i + 1) *

```

```

20);//inputcount * 30);
    pan.NickName = spacenames[i] + "_Z";
    pan.SetUserText("0");

    //Grasshopper.Kernel.Parameters.Param_Number num = new
Grasshopper.Kernel.Parameters.Param_Number();
    //num.CreateAttributes();
    //num.Attributes.Pivot = new PointF((float)
this.Component.Attributes.DocObject.Attributes.Bounds.Left -
num.Attributes.Bounds.Width, (float)
this.Component.Params.Input[7].Attributes.Bounds.Bottom + (i + 1) *
20);//inputcount * 30);
    //num.NickName = "zemin";
    //num.Access.Equals(1);
    GrasshopperDocument.AddObject(pan, false);
    this.Component.Params.Input[8].AddSource(pan);
}

else{
    Grasshopper.Kernel.Special.GH_NumberSlider slid1 = new
Grasshopper.Kernel.Special.GH_NumberSlider();
    slid1.CreateAttributes(); //sets up default values, and
makes sure your slider doesn't crash rhino
    int inputcount =
this.Component.Params.Input[0].SourceCount;
    slid1.NickName = spacenames[i] + "_Z";
    slid1.Attributes.Pivot = new PointF((float)
this.Component.Attributes.DocObject.Attributes.Bounds.Left -
slid1.Attributes.Bounds.Width, (float)
this.Component.Params.Input[8].Attributes.Bounds.Bottom + (i + 1) *
20);//inputcount * 30);

    slid1.Slider.Maximum = floorcount - 1;
    slid1.Slider.Minimum = min;
    slid1.Slider.DecimalPlaces = 0;
    slid1.SetSliderValue((decimal) floorcount - 1);
    GrasshopperDocument.AddObject(slid1, false);
    this.Component.Params.Input[8].AddSource(slid1);
}

//Until now, the slider is a hypothetical object.
// This command makes it 'real' and adds it to the canvas.
// GrasshopperDocument.AddObject(slid, false);

//Connect the new slider to this component
// this.Component.Params.Input[5].AddSource(slid);
}
}

List<int> spaceindexlist = new List<int>();

for (int i = 0; i < count; i++)
{

```

```

    if(groundconstraint[i])
    {
        spaceindexlist.Add(0);
    }

    else
    {
        int spaceindex = x[i];

        spaceindexlist.Add(spaceindex);
    }

    A = spaceindexlist;

}

// List<Point3d> selectedpointslist = new List<Point3d>();
// for (int i = 0;i < count; i++)
// {
//     Point3d selectedpoints = data[spaceindexlist[i]];
//     selectedpointslist.Add(selectedpoints);
// }
// Point3d selectedpoints = data[23];
//

// A = selectedpointslist;

}

// <Custom additional code>

// </Custom additional code>

```