TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

MASTERARBEIT

Optimization-based Site Planning System

ausgeführt zum Zwecke der Erlangung des akademischen Grades einer Diplom-Ingenieurin unter der Leitung

Ao.Univ.Prof. Dipl.-Arch. Dr.phil. Georg Suter

E259

Institut für Architekturwissenschaften

eingereicht an der Technischen Universität Wien

Fakultät für Architektur und Raumplanung

von

Lamprogianni Aikaterini

0925997

Märzstrasse 135/17, 1140 Wien

Wien, am 02.10.2012

Abstract. This thesis explores optimizing the arrangement of buildings on a site according to a set of spatial constraints. Design objectives include solar insolation and building proximity. The difficulty of this task increases considerably with the number of buildings on a site or the complexity of site and building geometries. The solution space associated with such a problem is typically infinitely large and therefore its exploration could be supported by design optimization methods. For that purpose, an optimization-based site planning system was designed and implemented. The system was implemented in an existing geometric design software that provides optimization functionality based on genetic and simulated annealing algorithms. The system was further improved using fine-tuning procedures, involving experiments with the different algorithms and with different implementation decisions. The efficiency of the system and the quality of the results produced is evaluated in two ways: By producing a series of optimization examples according to different system settings for an arbitrary set of buildings and site and by simulating an existent design problem for two case studies. After the simulation is completed, the values of the design criteria of each case are calculated for the actual buildings and compared to the ones produced by the system.

Keywords. massing design, site planning, optimization, generative design, genetic algorithm, simulated annealing algorithm

Table of Contents

`

List of figures

List of charts

1 Introduction

1.1 Motivation

Ever since the first attempts to introduce computational tools into the concept design phase -50 years ago-, an ongoing conversation has started about the role of the computer into the design procedure. Between the two extremes (absolute automation in floor plan design or complete denial of the computational contribution), a balanced way can be found to facilitate the tools available so that they can forward the design procedure, always under a specific scope and with certain limitations. Subjective matters, such as aesthetics or functionality – in its general meaning- cannot be addressed by computational tools since it is not even easy to define such problems. There are though many design aspects that can have a clear definition or at least clear evaluation criteria; these aspects can be computationally optimized in less time and more efficient way.

Even though computational optimization techniques are particularly popular for many years now, they are not widely implemented and used in architectural practices. This is happening on the one hand because of the declination in the variables that are driving the design and on the other hand because of the nature of the implementations, that are not designed to be widely used but mainly for research. The work that is done, attempts to minimize the distance between optimization tools and design everyday-life. Therefore tries to explore the capabilities of software that are already familiar to designers and find out in what extend optimization techniques can be implemented in order to give an answer to specific design problems. In terms of meeting the design criteria of the users, this attempt began with solving shading issues, an aspect considered rather important in the phase of conceptual site planning. Accessibility constraints couldn't be omitted since they almost always affect the design decisions. Another criterion considered necessary was the buildings proximity.

Site planning was chosen as an appropriate field for optimization mainly because it of the desire to study the optimization of criteria that are strongly related to the buildings environmental sustainability than e.g. the minimization of costs, optimization of material-use or structural optimization. The design phase in which the implementation is intended for use, is the conceptual phase; thus it was considered more appropriate to study the actual placement of the building since materials, façade openings and construction details are introduced in a latter phase of the design. Apart from these factors this case of optimization has an appropriate problem definition, since the evaluation criteria are clearly defined (maximizing/minimizing solar insolation and buildings proximity). It is also a case where computational contribution would be highly appreciated since infinite possible combinations should be tested in order to manually verify which ones produce the best results; it is in fact unfeasible to manually test more than a limited number.

1.2 Background

The extensive research done during the last 50 years in the area of automated floor plan design is particularly relevant with site planning optimization, in the sense that the optimal solution for a configuration of different spaces is sought.
In the case of floor plan layout, the design constraints are usually the spaces boundaries and the range of dimensions. The criteria that are being evaluated are mainly the space areas, the proportions and the relations between the spaces, even though in some cases orientation or other criteria are also object of evaluation. In the case of optimizing a building complex configuration on a site, there is a slight alteration on the constraints as well as on the evaluation criteria. The boundaries and the overlaps between the buildings are still an important spatial constraint, but they are defined in a different way. The relations between the buildings are also here an evaluation criterion (building proximity) whereas the relation to the site is also important. Shading efficiency is the key evaluation criterion.

Many approaches and different strategies have been undertaken, while searching for a way to automate floor plan design. The method of evaluation that guides the process is the main feature of the different approaches, according to which they can be distinguished into three categories (Ligget 2000): the ones optimizing a single criterion function, the ones based on graph theoretic approaches where nodes represent activities to be located and edges represent a direct adjacency requirement (Grason 1971, Muther 1973), and the ones trying to find an arrangement that satisfies a diverse set of constraints or relations instead of optimizing a single measure. The various methods followed by the third group, are of particular interest for the implementation of

this system, since the optimization is going to be done according to a set of constraints and considering more than one evaluation criteria. Early such examples are the 'General Space Planner' (Eastman 1973) and the 'Design Problem Solver' (Pfefferkorn 1972) where the goal is to satisfy a set of constraints such as position, orientation, adjacency, path, view or distance. 'SEED' is a software system that supports the early phases in building design by generating rectangular space under various constraints like access, natural light and privacy (Flemming, Coyne, et al. 1994). It was developed at Carnegie Mellon along with 'HeGel' (Akin, Dave and Pithavadian 1992) and 'WRIGHT' (Baykan 1991), two other methods with similar representation but significantly different approaches to constraint satisfaction. 'HeGel' uses a hierarchical generate-and-test method that incrementally constructs solutions by adding one space area (one rectangle) at a time to a partial solution, testing for constraint satisfaction at each step. If no possible addition of a new space meets the constraints, the system backtracks. If one possible new space meets the criteria, the placement is made and if more than one spaces meet the criteria, they are presented to the user for selection. 'WRIGHT' implements a constraint-directed search in which constraints are incrementally satisfied in order to produce a full solution that finds all significantly different solutions. All three methods are based on construction algorithms.

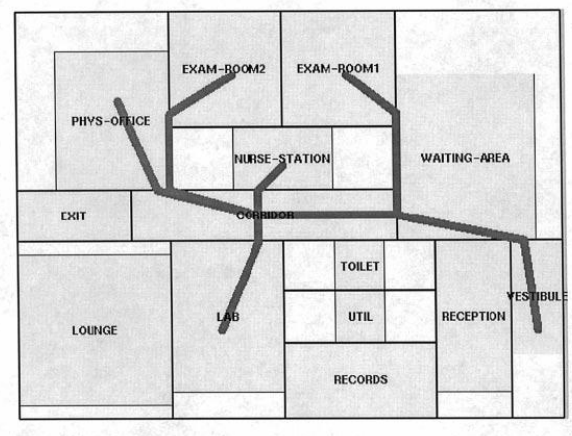


Figure 1.1: Display of access paths in a SEED layout (Flemming, Coyne, et al. 1994)

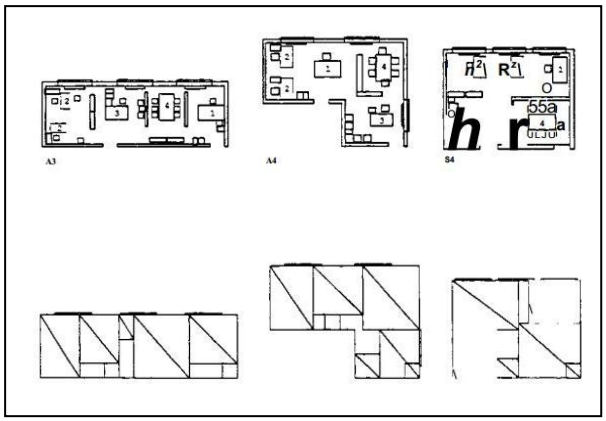


Figure 1.2: Solutions generated by the subjects and by HeGel (Akin, Dave and Pithavadian 1992)

Genetic algorithms and iterative improvement-based algorithms (including simulated annealing) improved the efficiency and the speed of the optimization process. Applications of genetic algorithms show excellent results for the floor plan layout problem (Ligget 2000). A simulated annealing-based method for solving the facility layout problem was applied, by Sharpe and Marksjo (Sharpe and Marksjo 1986) providing a simple but powerful approach to facility layout optimization that was successfully applied to large scale problems with up to 200 locations.

A method of evolutionary computation with the Pareto front based on a weighted function of the objectives used for multi-objective optimization was designed by Ciftcioglu and Bittermann and applied successfully on a case study positioning houses in a residential neighborhood according to the garden performance and visual privacy performance requirements. (Ciftcioglu and Bitterman 2008)

Figure 1.3: Two Pareto-optimal designs found by the genetic algorithm (Ciftcioglu and Bitterman 2008)

Apart from the theoretical approaches, the prototypes and generally the research (academic or not) that has been done in the field, it also worth mentioning the recent efforts that have been done in the direction of making such tools available in architectural practices. In 1998, the German company AcadGraph produced 'Alberti', a software for producing automated floor plans, by giving the input of stories number, name orientation and relations between the rooms. (Lobos and Donath 2010) The software handles the relationships and constraints through the application of the concepts of Neural Networks and produces around a hundred of solutions per second, which are then evaluated according to the criteria. The tool is no longer available.

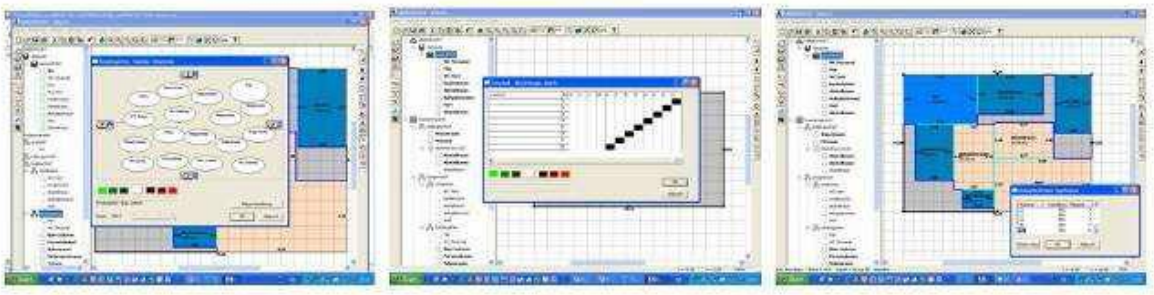
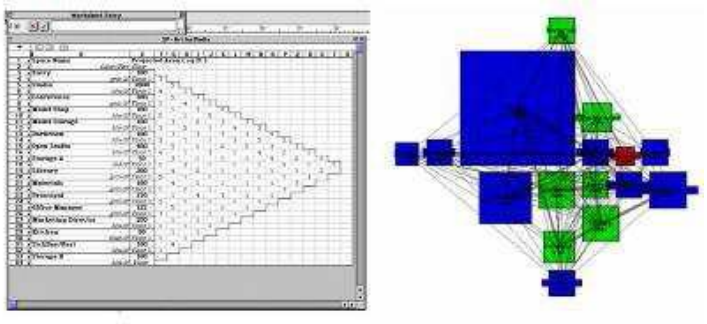


Figure 1.4: Alberti byAcadGraph (Lobos and Donath 2010)



'Vectorworks' 10th version released in 2004 by Nemetcheck, included space planning tools, where the user could see a planar, non-overlapping distribution of the rooms on the screen, after defining the space program. It can be used as basis to create a floor plan, even though it doesn't consider a floor boundary constraint.

Figure 1.5: Vectorworks10 by Nemetscheck (Lobos and Donath 2010)

1.3 Fundamentals of Computational Optimization

Since the implementation is done based on meta-heuristic techniques, this chapter covers some basic concepts of computational optimization techniques and a short description of the ones that are implemented in the system.

Algorithms with stochastic components were often referred to as heuristic in the past, though the recent literature tends to refer to them as metaheuristics. All modern nature-inspired algorithms could conventionally be called metaheuristics (Glover and Kochenberger, Handbook of metaheuristics 2003). Heuristic means –in a more general sense- to find or to discover by trial and error. Here meta- means beyond or higher level, and metaheuristics generally perform better than simple heuristics. The word "metaheuristic" was coined by Fred Glover in his seminal paper (F. Glover 1986), and a metaheuristic can be considered as a "master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality" (Glover and Laguna 1986). In addition, all metaheuristic algorithms use a certain tradeoff of randomization and local search. Quality solutions to difficult optimization problems can be found in a reasonable amount of time, but there is no guarantee that optimal solutions can be reached. It is hoped that these algorithms work most of the time, but not all the time. Almost all metaheuristic algorithms tend to be suitable for global optimization. (Yang 2010)

Many problem-solving processes tend to be heuristic throughout the human history; however heuristic as a scientific method for optimization is a modern phenomenon. From the 1940s to 1960s, heuristic methods have been used in various applications, but the first landmark came with the advent of evolutionary algorithms. In 1963 Ingo Rechenberg and Hans-Paul Schwefel, then both at the Technical University of Berlin, developed evolutionary strategies while L. J. Fogel et al. developed evolutionary programming in 1966. Genetic algorithms were developed by J. Holland in the 1960s and 1970s, though his seminal book on genetic algorithms was published in 1975 (Holland 1975).
The 1980s and 1990s were the most exciting time for metaheuristic algorithms. One big step was the development of simulated annealing (SA) in 1983, an optimization technique, pioneered by S. Kirkpatrick et al., and inspired by the annealing process of metals.
The two subfields of metaheuristics optimization that are going to be used for the implementation of the system are the genetic algorithms and the simulated annealing algorithm. There are many optimization algorithms which can be classified in many ways, depending on the focus and characteristics. We will mention the ones according to which genetic and simulated annealing algorithms can be classified (Figure 1.6).

If the derivative or gradient of a function is the focus, optimization can be classified into gradient-based algorithms and gradient-free algorithms. Gradient-free algorithms do not use any derivative information but the values of the function itself. From a different perspective, optimization algorithms can be classified into trajectory-based and population-based. A trajectory-based algorithm typically uses a single agent or one solution at a time, which will trace out a path as the iterations continue, while population-based algorithms such as genetic algorithms use multiple agents which will interact and trace out multiple paths. Optimization algorithms can also be classified as deterministic or stochastic. If an algorithm works in a mechanical deterministic manner without any random nature, it is called deterministic. For such an algorithm, it will reach the same final solution if we start with the same initial point. On the other hand, if there is some randomness in the algorithm, the algorithm will usually reach a different point every time the algorithm is executed, even though the same initial point is used. Search capability can also be a basis for algorithm classification. In this case, algorithms can be divided into local and global search algorithms. Local search algorithms typically converge towards a local optimum, not necessarily the global optimum. Modern metaheuristic algorithms in most cases tend to be suitable for global optimization, though not always successful or efficient.

	METAHEURISTIC OPTIMIZATION ALGORITHMS							
	CLASSIFICATION 1		CLASSIFICATION 2		CLASSIFICATION 3		CLASSIFICATION 4	
	GRADIENT -BASED	GRADIENT -FREE	TRAJECTORY-BASED	POPULATION-BASED	DETERMINISTIC	STOCHASTIC	LOCAL SEARCH	GLOBAL SEARCH
GENETIC ALGORITHM		x		x		x		x
SIMULATED ANNEALING ALGORITHM	x		x			x		x

Figure 1.6: Classification of Genetic and Simulated-Annealing Algorithms

Genetic Algorithms

Genetic algorithms (GAs) are probably the most popular evolutionary algorithms with a diverse range of applications. A vast majority of well-known optimization problems have been solved by genetic algorithms. In addition, genetic algorithms are population-based and many modern evolutionary algorithms are directly based on, or have strong similarities to, genetic algorithms.
Genetic algorithms, developed by John Holland and his collaborators in the 1960s and 1970s, are a model or abstraction of biological evolution based on Charles Darwin's theory of natural selection. Holland was the first to use crossover, recombination, mutation and selection in the study of adaptive and artificial systems (Holland 1975). These genetic operators are the essential components of genetic algorithms as a problem-solving strategy and are the minimum set of operators, that distinguish GAs from other evolutionary computation

methods even though there is no rigorous definition of 'genetic algorithm' accepted by all in the evolutionary-computation community. (Mitchell 1996)
The goal while running a genetic algorithm is to optimize an evaluation criterion, which in terms of evolutionary computing is the fitness function. This fitness function is the evaluation of every solution variation. Each solution is created after combining a set of variables: the genes -and thus each solution can also be called a chromosome or a genome (a combination of genes). The encoding of the genomes was done in its original version as arrays of bits or character strings, even though that doesn't always have to be the case.

Therefore the two first steps, in order to initiate a genetic algorithm are :

- the definition of the solution space (the genes) and
- the definition of the fitness function

The following steps are:

- creation of a population of genomes (generation)
- evaluation of the fitness of every genome in the population
- creation of a new population by performing fitness-proportionate selection, crossover and mutation
- replacement of the old population by the new one

The last three steps are then repeated for a number of generations. Each iteration, which leads to a new population of genomes, is called a generation. The algorithm always starts with a random-selected generation: generation0. In every iteration of the algorithm every new generation is an improved version of the previous one. The best genome of the last generation is decoded to obtain a solution to the problem.

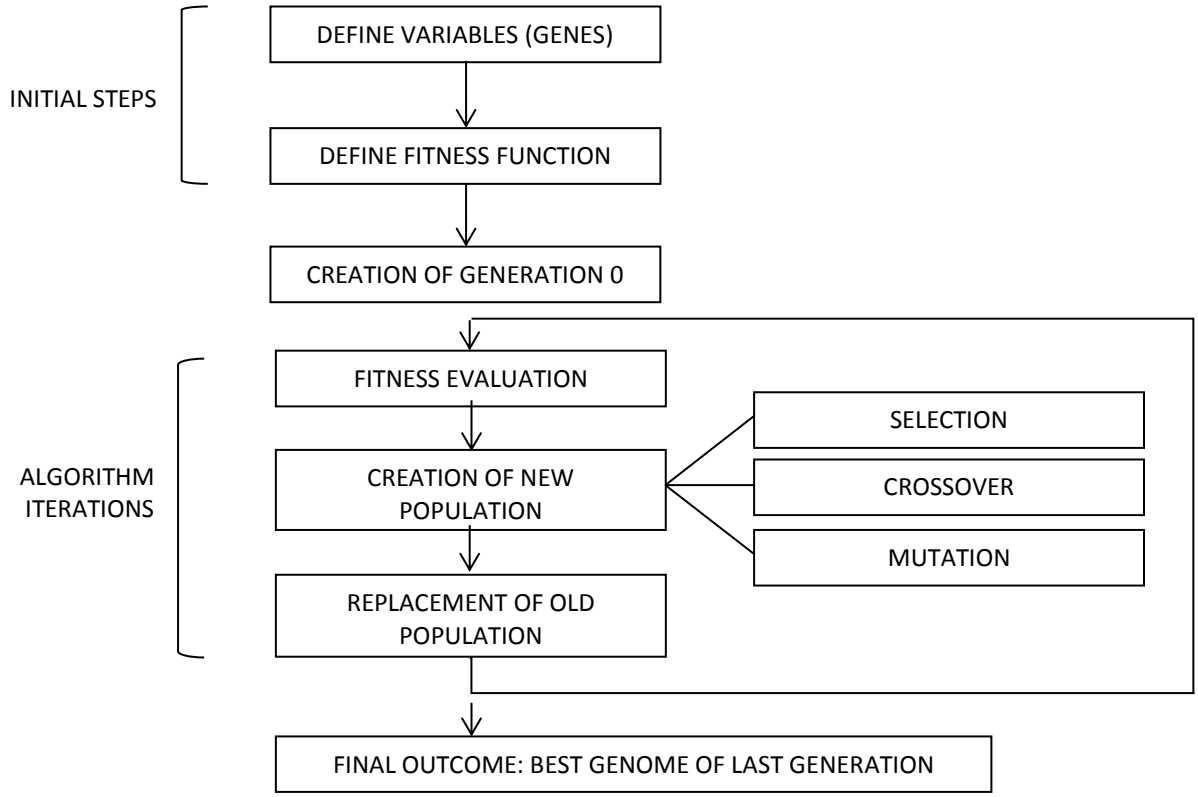


Figure 1.7 : Evolutionary algorithm structure

The selection operator selects the genomes for reproduction. The fitter the genome, the more times it is likely to be selected to reproduce. Sometimes, in order to make sure that the best genomes remain in the population, they are transferred to the next generation without much change, which is called elitism. (Koziel and Yang 2011)

Crossover operator randomly chooses a locus and exchanges the subsequences of genes before and after that locus between two genomes to create the offspring. The crossover operator roughly mimics biological recombination between two single-chromosome organisms. The mutation operator randomly flips some of the genes is a genome, according to some probability factor that is usually very small. (Mitchell 1996)

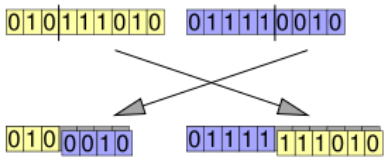


Figure 1.8: Crossover Operator

An important issue is the formulation or choice of an appropriate fitness function that determines the selection criterion in a particular problem. A proper criterion for selecting the best chromosomes is important, because it determines how chromosomes with higher fitness are preserved and transferred to the next generation. This is often carried out in association with a certain form of elitism. The basic form is to select the best chromosome (in each generation) which will be carried over to the new generation without being modified by the genetic operators. This ensures that a good solution is attained more quickly. Another important issue is the choice of various parameter values. The crossover probability pc is usually very high, typically in the interval $[0.7,1.0]$. On the other hand, the mutation probability pm is usually small (typically, in the interval $[0.001,0.05]$). If pc is too small then crossover is applied sparsely, which is not desirable. If the mutation probability is too high, the algorithm could still `jump around' even if the optimal solution is close.

Other issues include multiple sites for mutation and the use of various population sizes. The mutation at a single site is not very efficient, so mutation at multiple sites typically increases the evolution of the search. On the other hand, too many mutants will make it difficult for the system to converge or even lead the system toward wrong solutions. In real ecological systems, if the mutation rate is too high under high selection pressure, then the whole population might become extinct. The choice of the right population size is very important, because if the population size is too small, there will not be enough evolution, and there is a risk for the whole population to converge prematurely. In the real world, ecological theory suggests that a species with a small population is in real danger of extinction. In a small population, if a genome with a fitness substantially larger than the fitness of the other chromosomes in the population appears too early, it may produce enough offspring to overwhelm the whole (small) population. This will eventually drive the system to a local optimum (not the global optimum). On the other hand, if the population is too large, more evaluations of the objective function are needed, which will require extensive computing time.

As a simple example, an initial population is generated and its final solution locations aggregate towards optimal solutions (Figure 1.10, Figure 1.9)

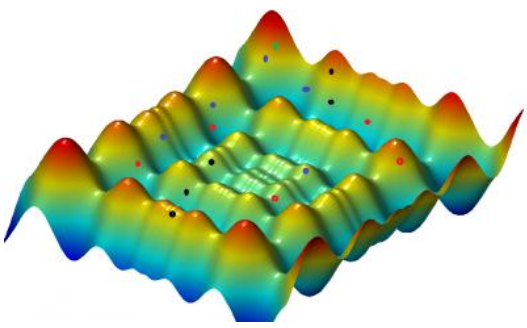


Figure 1.10: Genetic algorithm -initial population and solutions (Yang 2010)

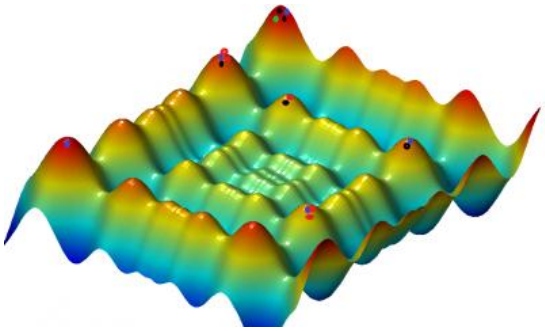


Figure 1.9: Genetic algorithm -final solutions (Yang 2010)

Simulated Annealing Algorithm

Similarly to the theoretical approach of the evolutionary algorithm, the optimization problem is structured in the following way: We are searching a set of solution candidates in order to find the optimal solution that minimizes the cost function. The cost function is similar to the fitness function in GAs, with the difference that in Simulated Annealing theory we regard the optimization as a minimization function. This originates from the physical inspiration of the algorithm, where the objective is to bring a system to a state of minimum energy. The simulated annealing algorithm is a technique that belongs to a class of search algorithms called threshold algorithms. A starting point is initially selected from the solution candidates. For every step of the algorithm a new candidate is selected. The cost of the original solution is abstracted from the cost of the new one. If the cost difference between the two solutions is below the threshold (t) the new solution replaces the previous one, otherwise the original one remains as current. This procedure is repeated until the termination of the algorithm. The algorithm terminates either because of a time limit or because it reached a specific result. There are three types of threshold algorithms (Aarts, Korst and van Laarhoven 2003):

- Iterative improvement: The cost function of the new solution must always be less than the previous one (t=0). This is a greedy local search variant.
- Threshold accepting: The threshold (t) can have values bigger than or equal to zero, which are decreasing on every comparison. As a result other solutions with larger costs are accepted but in a limited way, mostly at the beginning of each run. Gradually the algorithm reaches a point where the cost difference must be zero and only improvements are accepted.
- Simulated annealing: The threshold (t) is a random variable with values between zero and infinity that follows a probability distribution function formulated in such a way that solutions that correspond to large increases in cost have small probability of being accepted, whereas solutions that correspond in small increases in cost have larger probability. In simulated annealing each new solution can be chosen to replace the current one.

  The probability distribution function is the negative exponential distribution with parameter $1/c_k$ (Kirkpatrick, Gelatt and Vecchi 1983).The parameter c_k is used in the simulated annealing algorithm as a control parameter, and it plays an important role since it leads to the selection of larger increases in cost at the beginning of the algorithm and rejects them while the algorithms is getting closer to the solution.

$$P_{c_k}\{accept\ j\} = \begin{cases} 1 & ,\ if\ f(j) \le f(i) \\ \exp\left(\frac{f(i)-f(j)}{c_k}\right), & if\ f(j) > f(i) \end{cases}$$

  Where i,j are solutions that belong to the solution set of the problem, f is the cost function of (i) and (j), k the number of iteration (k=0,1,2,….) and P_{c_k} is the probability of accepting j from i at the k^{th} iteration.

The procedure that is followed in simulated annealing is inspired by statistical mechanics and is very similar to annealing in metallurgy, a technique where heating and controlled cooling of a material are used in order to increase the size of its crystals and reduce its defects. Statistical mechanics is the central discipline of condensed matter physics that applies a probability theory for dealing with large populations, while analyzing the atoms' properties found in samples of liquids or solids. The state of the system in such samples can only be described by average behaviors, because of the large number of atoms per cubic centimeter, when the average is taken over the ensemble of identical system introduced by Gibbs (Kirkpatrick, Gelatt and Vecchi 1983). In this ensemble each configuration defined by the atomic positions r{i} of the system is weighted by its Boltzmann probability factor exp (-E({r_i})/k_BT), where E({r_i}) is the energy of the configuration, k_B is Boltzmann's constant and T is temperature. The Boltzmann factor is a weighting factor that determines the relative probability of a particle to be in a state i in a multi-state system in thermodynamic equilibrium at temperature T. As T is lowered the Boltzmann distribution (sum of all the Boltzmann factors for all the states of the system) collapses into the lowest energy state or states and therefore ground state configurations of a macroscopic body dominate its properties at low temperatures, even though they are extremely rare among all the configurations. A Boltzmann factor drastically increases the efficiency of the system's ground state search.

In practical contexts, low temperature is not a sufficient condition for finding ground states of matter. In order to determine the low-temperature state of a material (for example by growing a single crystal from a melt) experiments are done by careful annealing: first melting the substance and then slowly lowering the temperature while spending a lot of time in the vicinity of the freezing point. Otherwise the substance is allowed to get out of equilibrium and the resulting crystal displays many defects.

The Metropolis algorithm for sampling from multi-dimensional distributions is used in order to utilize the techniques of statistical mechanics for solving optimization problems.

1.4 Methodology

In order to fulfill the task of designing the optimization system for site planning, the following steps are followed:

System Design. The structure of the system is initially designed schematically, independently of the implementation and the tools that are going to be used, by defining how the input, the output and the optimization task should be done. The input is the design input that the user gives i.e. the buildings, the site, local data and the desired optimization setting that will define the weights that each criterion has and the type of optimization (maximize/minimize). The output should be a proper documentation of the results, so that they are in later time accessible and an instant visualization of the ranked and evaluated solutions. The design of the optimization task is mainly about the definition of the constraints, the evaluation criteria and the relations between them. The constraints are actually the spatial constraints of site planning i.e. the appropriate placement of the buildings (elimination of building overlaps, containment in the site boundary, ease of access).

Implementation. As soon as the appropriate tools are found, the system should be implemented, in a way that exploits the tools capacities and explores the possibilities and the limitations. The way the user interacts with the system should also be considered, while finding the proper balance between user-interaction and the ability of the system to be 'user-friendly' and not too technical.

Examples. A set of examples based on an arbitrary set of buildings and site, displays how the system works for different input and different optimization settings.

Case studies. The most appropriate way of verifying the efficiency of the implementation or discovering possible limitations, is to test it on case studies i.e. cases that are already built and whose design objectives can be translated to the evaluation criteria of the optimization system. This way the results of the evaluation values between built-examples and the variations produced by the system can be compared.

2 System Design

The system consists of the following three parts: input, optimization and output.
During the input the user must provide the system with the design parameters and the desired optimization settings.
The optimization is the technical part of the system and is done mainly in two steps: the creation of different solutions (solution synthesis) and their evaluation. In order to create a solution, the solution space and the way according to which the solutions are generated have to be defined. In order to evaluate a solution we have to define the spatial constraints and the evaluation criteria.
The output is given in the form of documenting the results and visualizing them on the screen.

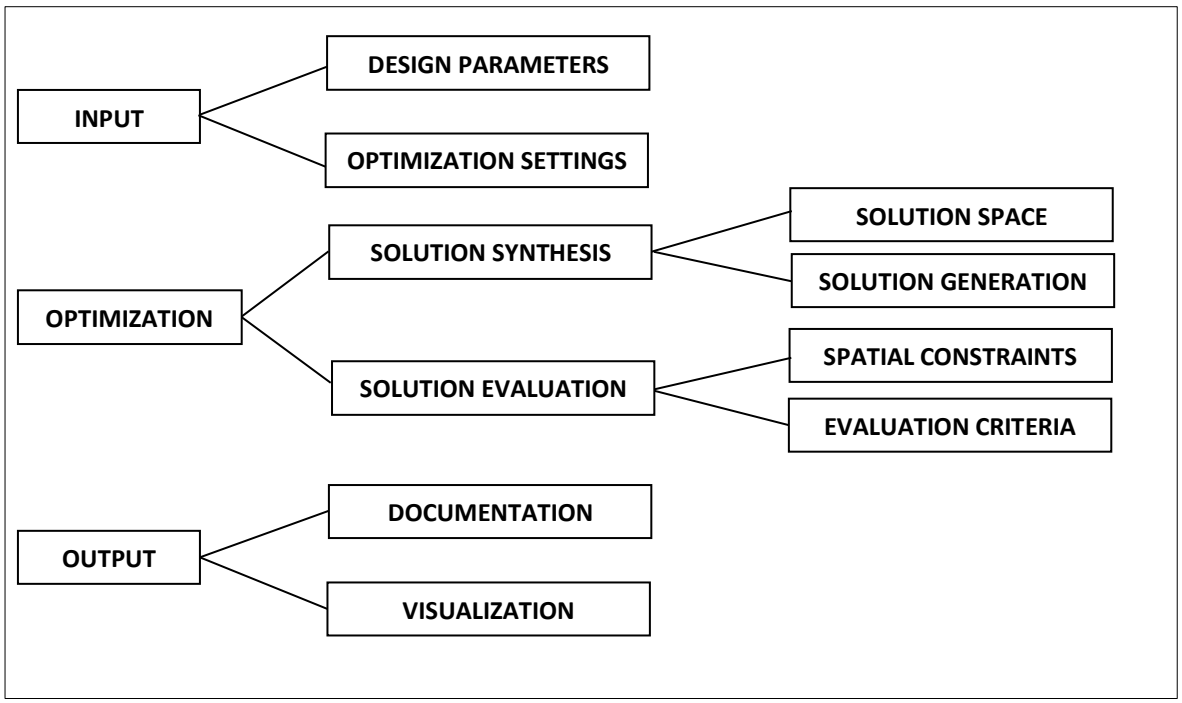


Figure 2.1: System Structure

2.1 System Terminology

In order to avoid confusion during the next chapters, we should clarify all the terms and concepts to which we are going to refer to. The site access area (SA) is the area on the site that is defined as the desired access area. It could be a street or a square; in general it is a boundary where nothing built is allowed to be on and that serves as circulation area. The buildings of the site are attracted to it and should be adjacent to its boundaries. The non-constructible zone (NCZ), is also an area where nothing should be built, but doesn't have any accessibility features; it's irrelevant whether the buildings are near or far from it. It could be an area where the site isn't suitable for a building's placement like a water-area, for example, or an area of very steep slope. It could also represent an existing building or be used to create zones that let the wind pass through. Thus, the constructible area of the site (CS) is the site area minus the site access area and the non-constructible zone, while considering the desired offset from the site boundary that expresses the minimum distance the buildings should keep from it. The building access area (BA) is the area inside a building boundary that should be adjacent to the site access area. It could be the lobby of the building, the entrance area. We will call overlap zone (OZ) an area of certain width, surrounding the site access area. This is the zone where the entrances of the buildings should be located in. The buildings are described by the letter "B" and number i, for $i \in \{1,2,\ldots,n\}$ and n number of buildings on site. "d" is the distance of a building from the site access area

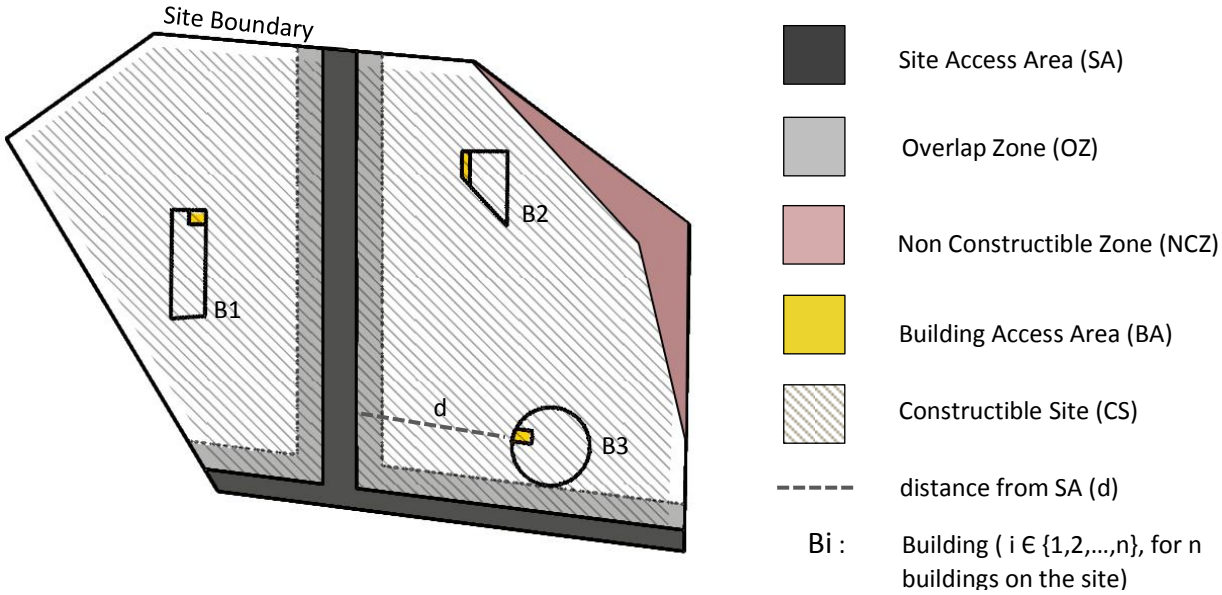


Figure 2.2: Site Planning Terms

2.2 Input

The design parameters that the user has to provide to the system are a) a set of objects i.e. the site, the site access area, the non-constructible zone, the buildings and the access area of the buildings and b) some planning parameters i.e. the minimum distance between the buildings, the minimum distance from the site boundary, the north angle and the site location (latitude and longitude). The input of the site access area and the non-constructible zone is optional; if they do not exist the system should still have the capacity to work efficiently. Regarding the optimization settings, the user must enter the weight that represents the level of importance during the optimization procedure for each evaluation criterion, and select whether the criteria should be maximized or minimized.

2.3 Optimization

2.3.1 Solution Synthesis

Definition of Solution Space

Each possible solution to the problem of placing a given set of buildings onto a given site can be accurately expressed with the help of two variables per building:

- The building location parameter, which is in fact a two-dimensional point:

$$P(x,y)$$

- The orientation, which is a numerical value of the rotation angle in respect to the positive y axis:

$$a \in \{0,1,2,\ldots,360\}$$

The possible points (P) are given after creating a grid of points that is contained into the boundary of the constructible site area (Figure 2.3,Figure 2.4). The possible angle values (a) are the integers between 0 and 360.

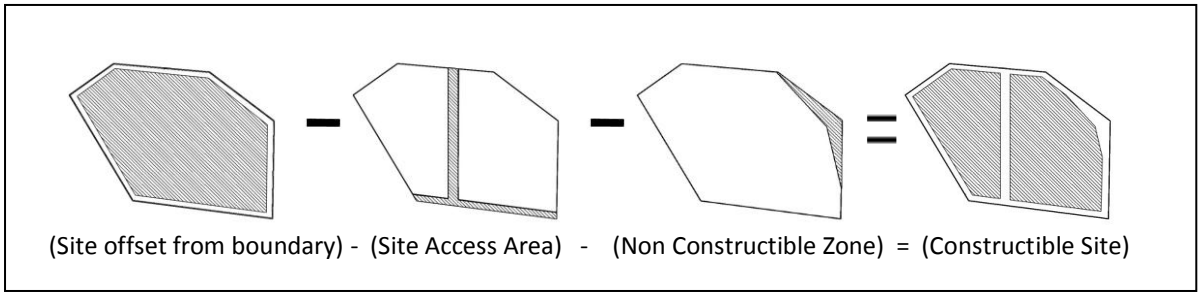


Figure 2.3: Constructible Site Definition

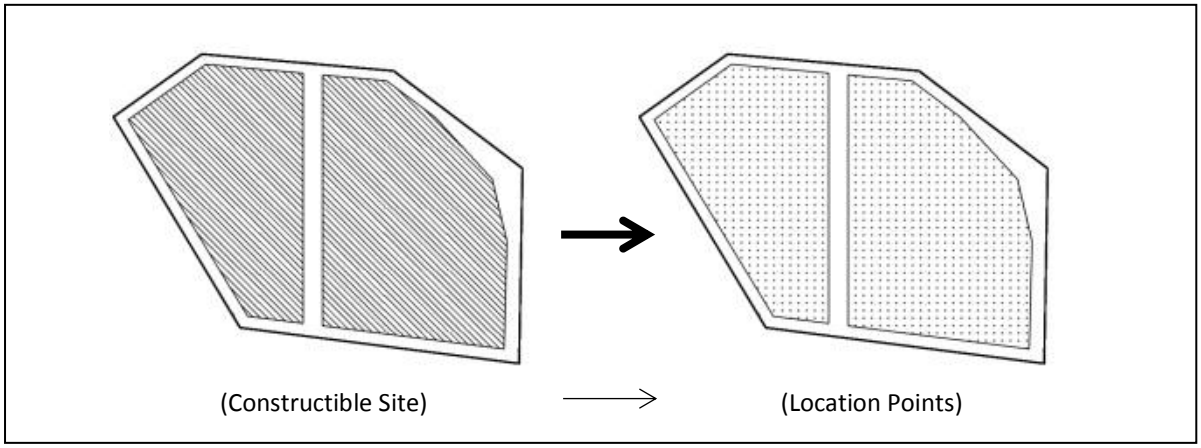


Figure 2.4: Definition of Location Points

Thus the problem's solution space is the set of all possible combinations of the two variables for all buildings (equations (2.1),(2.2)). For N buildings we would be searching for the best solutions in a 2*N-dimensional solution landscape. The number of possible solutions (k) is given if we multiply the number of buildings with the number of possible points and the number of discrete angle values. If we assume for example that the contained-in-constructible-site grid of a given site consists of 5000 points and there are 10 buildings on the site, then the number of possible solutions is $36*10^6$.

$$SolutionSpace = \{S_1, S_2, \ldots, S_k\} \quad (2.1)$$

$$Si = \{P_1, a_1, P_2, a_2, \ldots, P_N, a_N\} \quad (2.2)$$

for $k=N*N_P*N_a$, the number of all possible solutions,

N: the number of buildings on the site,

N_P: the number of points on the site,

N_a: the number of discrete angle values,

$i \in \{1,2,\ldots,k\}$

Solution Generation

In order to generate a solution each building is moved from its starting point to one of the points of the constructible-site-grid and rotated around it for some angle. The starting point is one point generated from the building's boundary given by the user.

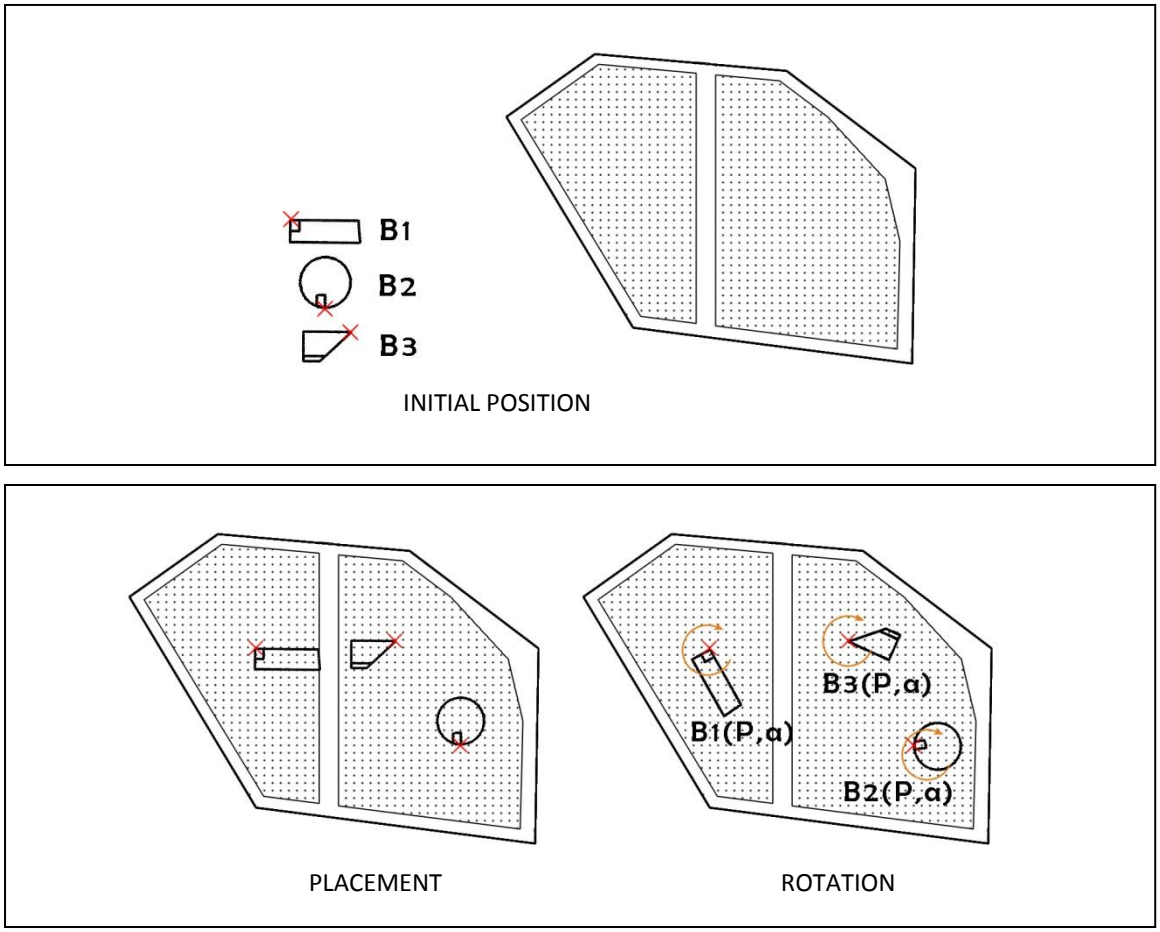


Figure 2.5: Generation of a solution

2.3.2 Solution Evaluation

Spatial Constraints

In order to control the quality of the solutions produced and avoid solutions that make no sense, there is a set of spatial constraints that the solutions should always meet. If they don't then they should be considered invalid and not provided in the final output.

Spatial Constraints:

1. All the buildings must be contained in the constructible site
2. The buildings shouldn't overlap with each other
3. The buildings access area should be at least in some extend contained in the overlap zone

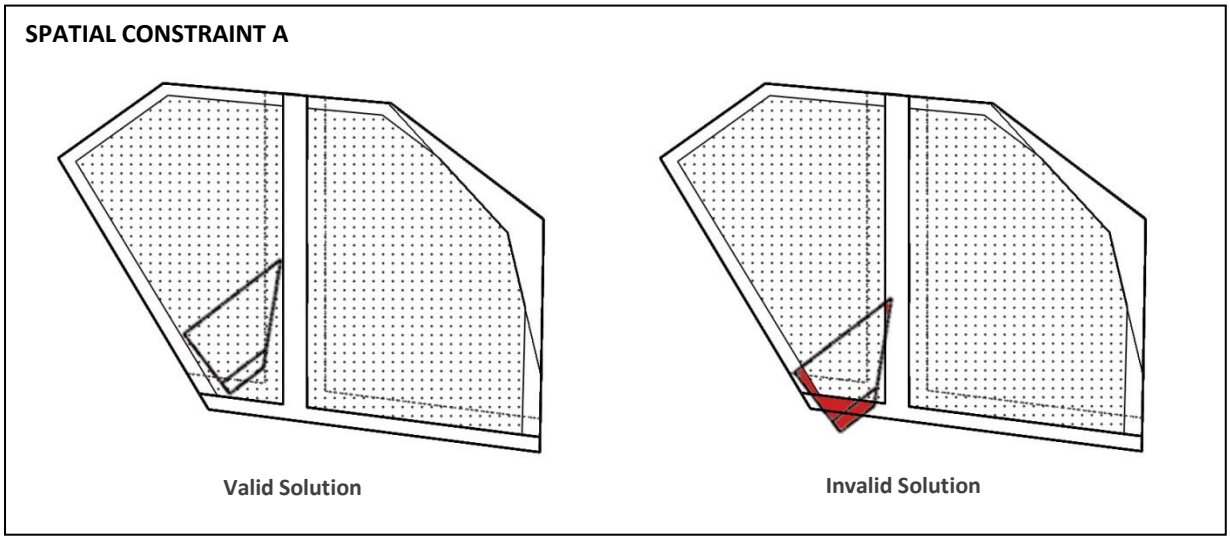


Figure 2.6: Spatial Constraint A: Building Containment in Constructible Site

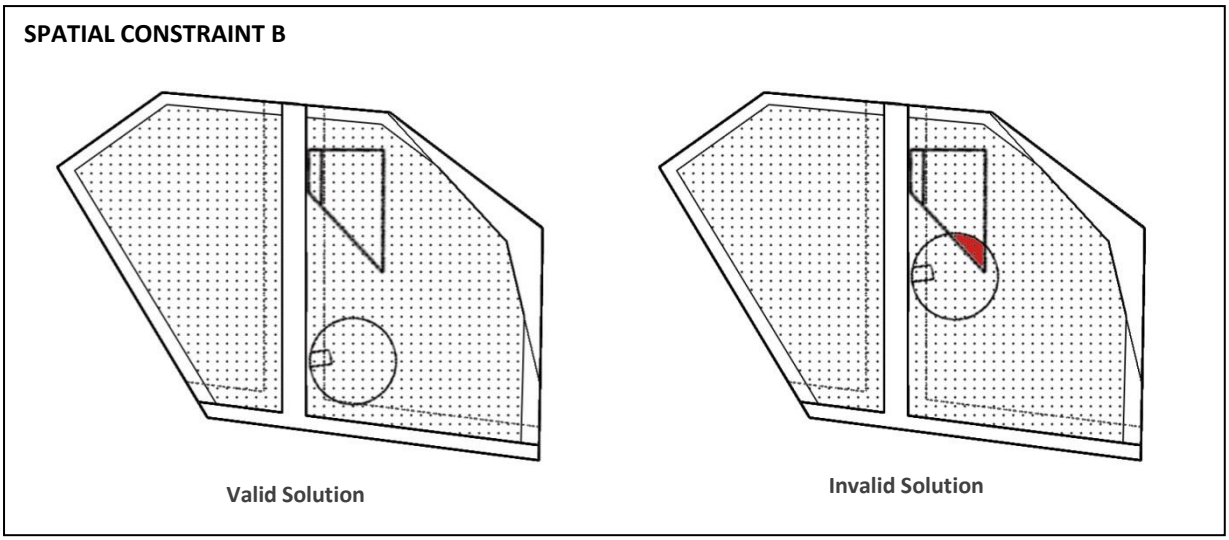


Figure 2.7: Spatial Constraint B: Buildings Overlap with each other

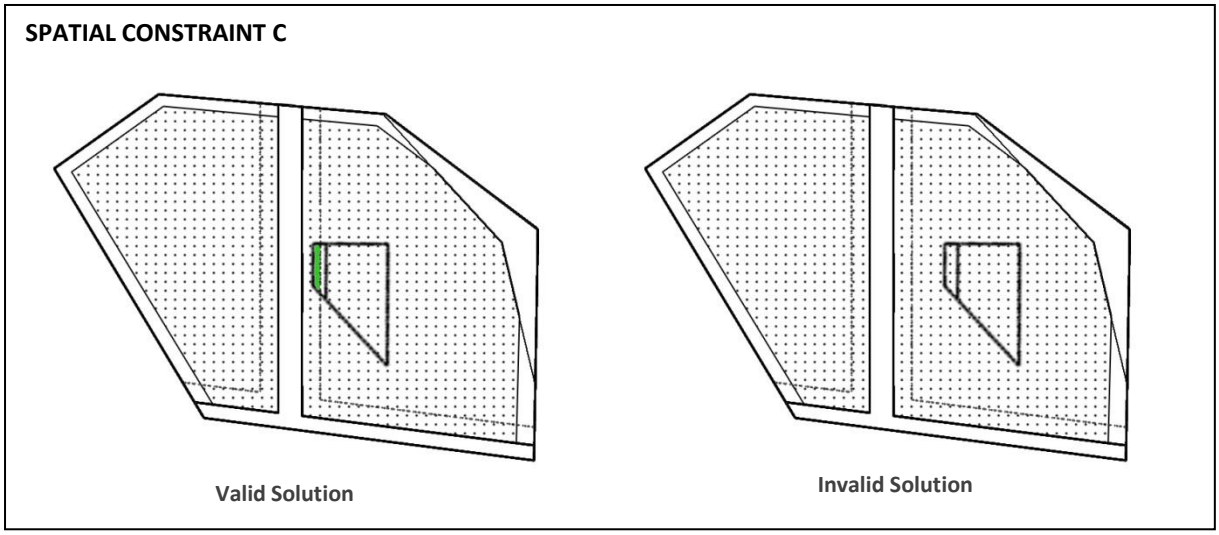


Figure 2.8: Spatial Constraint C: Accessibility Requirements

Evaluation Criteria

The optimization of the solar insolation is in fact the maximization/minimization of the sum of solar insolation values on the building envelope. Depending on the desired result and factors like local climate, the user can choose whether maximum or minimum values should be guiding the algorithm. The optimization of the building proximity is performed by either minimizing or maximizing the distances between the buildings.
It is obvious that these two criteria can either cooperate (when solar insolation is being maximized and building proximity is being minimized and vice versa) or conflict (when both criteria are being maximized or minimized). Thus the weights of each criterion can define the orientation of the optimization. The final evaluation values should be a sum of each evaluation criterion value for each building, while considering whether the spatial constraints are fulfilled or not.

2.4 Output

The results that should be documented are the total evaluation value of the solution, the evaluation values of each criterion and each building, the variables of the solutions (point and angle for each building) so that every solution can be later reproduced the solution iteration number and the solution efficiency number (number of solution in the set of ranked according to the efficiency solutions).
The visualization of the results should be a display of a number of desired solutions on the screen, according to their total efficiency, or their efficiency according to one of the criteria. It should also be possible that the user selects a number of the above mentioned characteristics to be displayed along with each solution.

3 Implementation

3.1 Environment

The selection of the tools for the implementation was done while having in mind three main goals:

- The tools should already be widely spread and familiar to a great number of users. The users should be able to find solutions to the site planning problem without having advanced knowledge of computational techniques, programming etc.
- They should facilitate an easy workflow so that interoperability problems can be avoided and the designer can work quickly without having to confront problems met while importing and exporting data between different software.
- They should have the necessary capacities to forward the system implementation.

The most suitable software meeting the above objectives are: Rhinoceros which is a modeling software and Grasshopper, a graphical algorithm editor integrated in Rhinoceros' modeling tools. The ability to work with genetic algorithms and simulated annealing algorithms is available inside Grasshopper.

In the Figure 3.1 we can see the interface of Grasshopper and an example showing the creation of a curve based on the user input. The tabs located on the top contain the tools available, which are mainly of two types: the parameters and the components. The parameters contain data and can either have an input from Rhinoceros, Grasshopper or directly from the user. The components are functions which require a specific input and result in an output. Every set of parameters and components is, in terms of Grasshopper, called a 'definition' and produces a specific design or solves a specific problem, in the same way and logic that an algorithm would do.

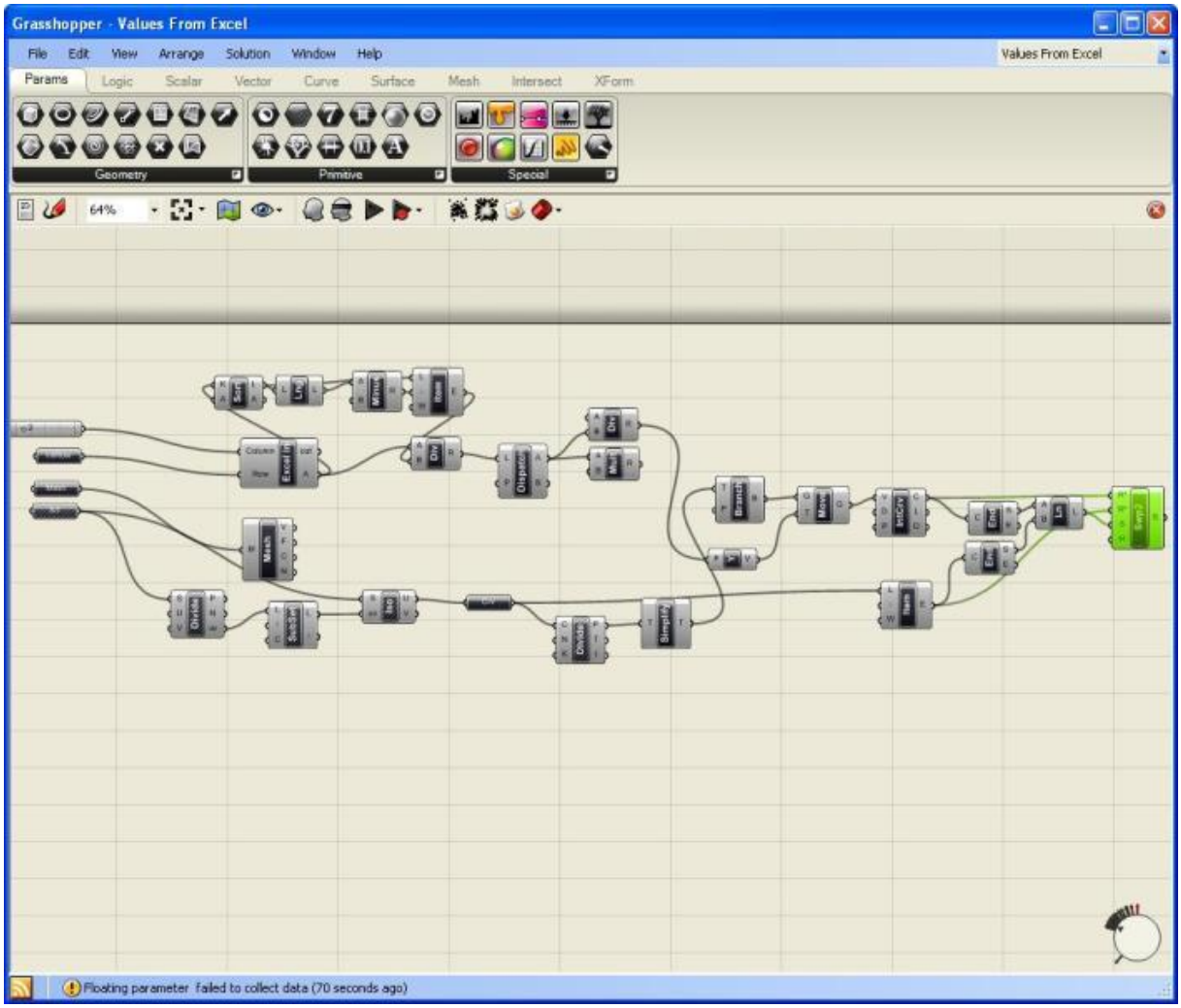


Figure 3.1: Grasshopper 'Definition'

In Figure 3.2 we can see the interface of 'Galapagos', which is a grasshopper component applying optimization solvers using a Genetic Algorithm (GA) and a Simulated annealing Algorithm (SA). The detailed description of both solvers as described in the Galapagos documentation is provided in Appendix C and Appendix D. Both solvers can both function, requiring exactly the same input. The input for galapagos is the range of value of some variables i.e. the solution space of the problem (in terms of GA the 'genes'), that describe a solution and the value of the evaluation criterion of this solution (in terms of GA the 'fitness') which should be optimized. Provided with this range of values the solver searches the solution space through several iterations and tries to optimize the evaluation criterion by searching different combinations of values.

The variables can be multiple, whereas the evaluation criterion is single. Therefore the optimization of multiple criteria must be implemented by the user. The output of Galapagos, is the combination of variables values (in terms of GA the 'genomes') that produce the most efficient - according to the evaluation criterion - results. The output of the process can be seen directly on the screen but a user-defined way to record it in order to reuse it has to be found. There is also a possibility to change some of the solvers' settings in the options menu. These settings are related to the structure of these solvers and the type of the optimization problem. That means that the user should experiment a little bit with the settings and their effect on the quality of the solution and the time the solver needs to find it.

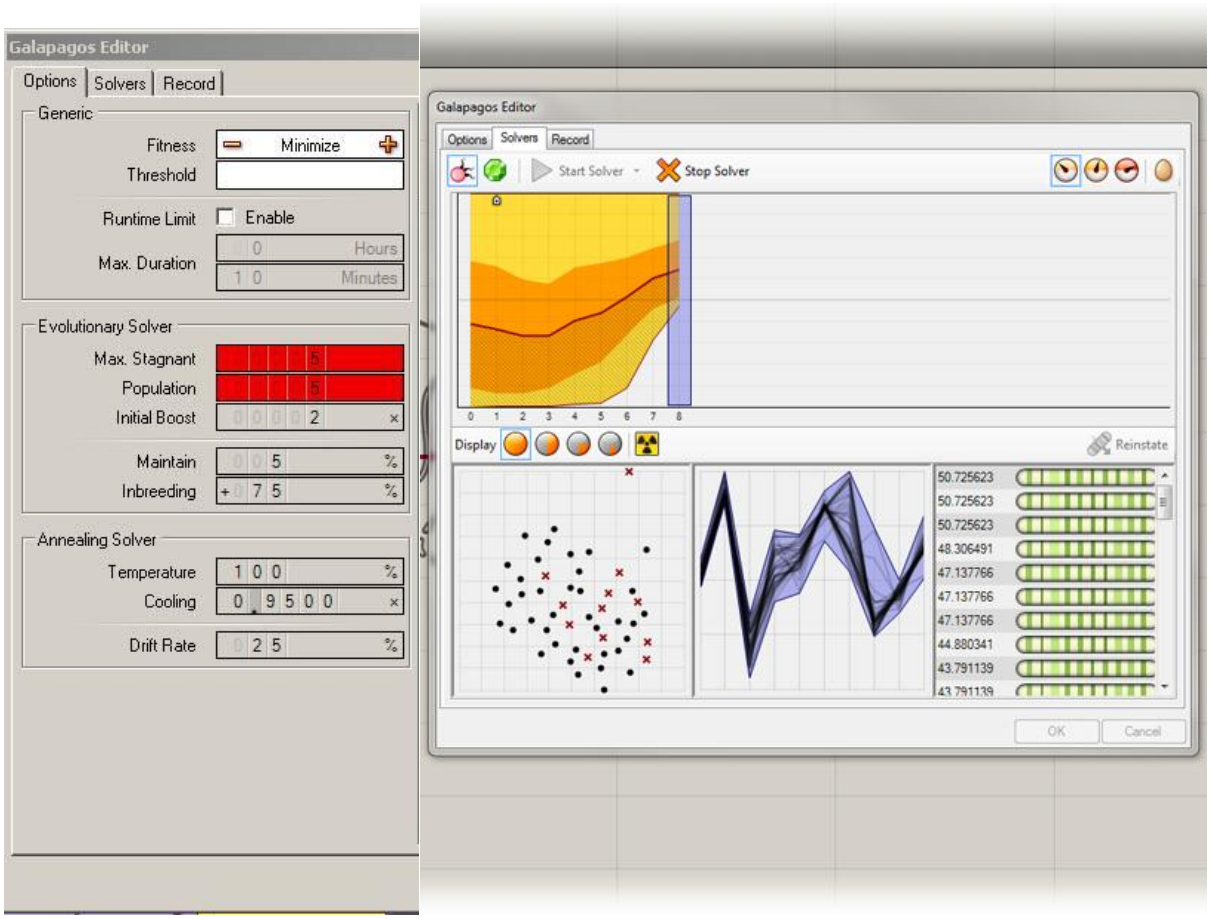


Figure 3.2: Genetic Solver Available in Grasshopper

3.2 Site Planner

In
Figure 3.3 we can see the general structure of the implementation which follows the system structure.

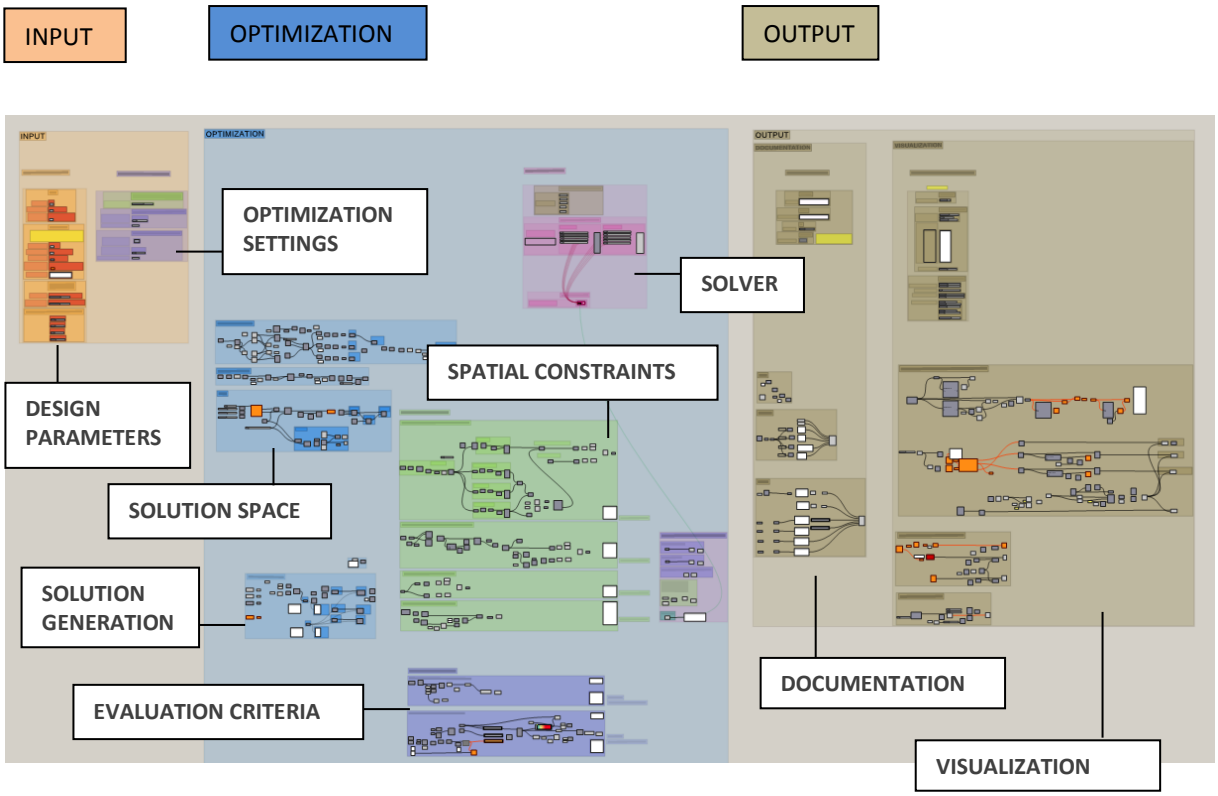


Figure 3.3: 'Site Planner' in Grasshopper

3.2.1 Input

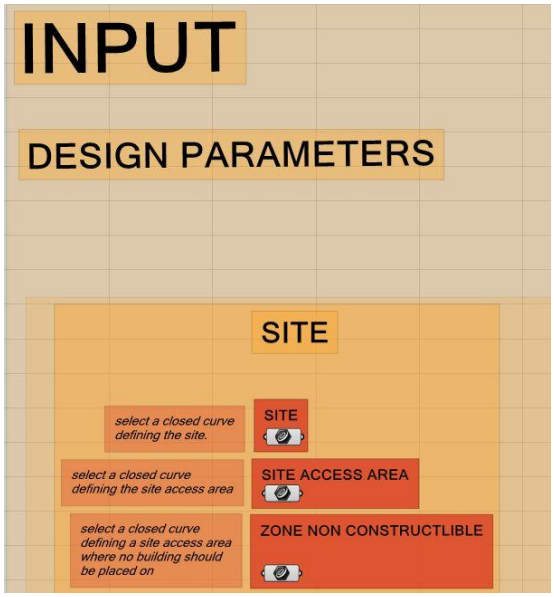


Figure 3.4: Design Parametes Input

In order to initiate the process the user gives an input of the design objects in grasshopper interface. This input is concerning four categories: the site, the buildings, the layout preferences and the sun calculation parameters.

SITE INPUT

The user must provide one closed polyline defining the site and one defining the access area in the site (streets, paths, e.t.c). Furthermore the user can provide a closed polyline defining area that shouldn't be built; the non-constructible area. The site access area is also optional. In case a parameter has no input, its component should remain empty.

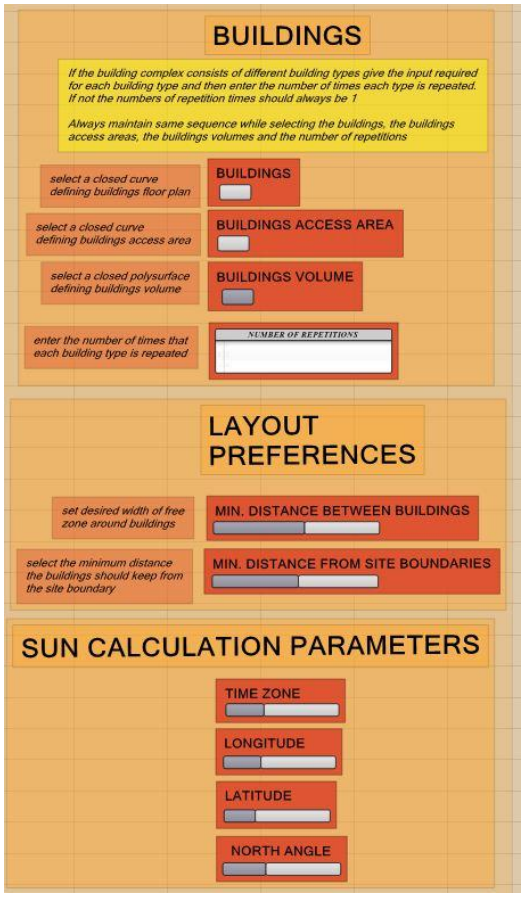


Figure 3.5: Design Parametes Input

BUILDINGS

Regarding the buildings, the user must select a polyline defining the floor plan and one defining the entrance plus the volume defined by a closed polysurface for each building.
It is important that during the selection all input concerning one building is one object, for example the floor plan is only one polyline, the building volume only one polysurface, etc. It is also important that a specific order is maintained i.e if the selection of the floor plans is following the order Building 1- Building 2 – Building 3 -…. then all other inputs (entrances, volumes etc) should also keep the same order.
When several building types are involved, the user should give the above mentioned input once per building type and then fill the list by entering the number that each building type is repeated. When building types do not exist and the planning concerns individual buildings only, each list item should have a value of 1.

LAYOUT PREFERENCES

The user provides the value of the distance that should be kept between the different buildings on the site and the distance that the buildings should keep from the site boundaries.

SUN CALCULATION PARAMETERS

The parameters needed for the sun calculation (longitude, latitude, time zone, north angle)

3.2.2 Optimization

Solution Definition

Solution Space

In order to define the solution space we must as described during the system design, produce a grid of points contained in the constructible area of the site.

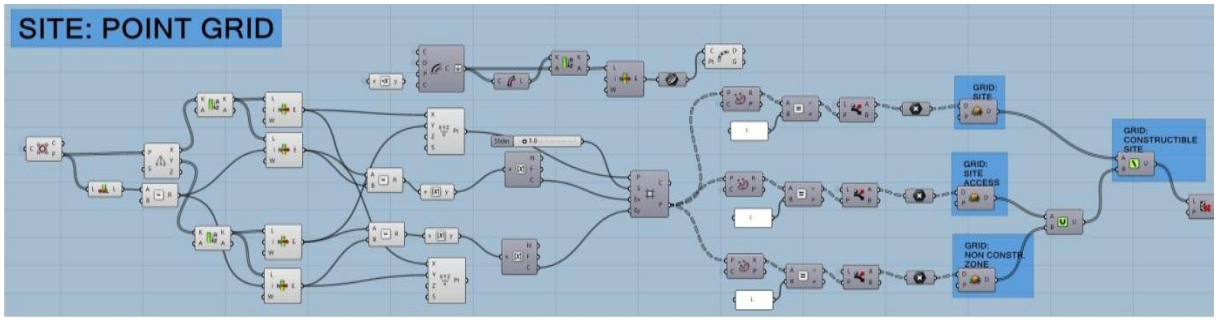


Figure 3.6 : Calculation of Grid points

The final grid points located on the constructible area of the site are produced, as we can see in Figure 3.6 and Figure 3.7, by finding which points of the initial grid are contained in the site boundary, the access area and the non-constructible zone and then subtracting the last two from the first one.

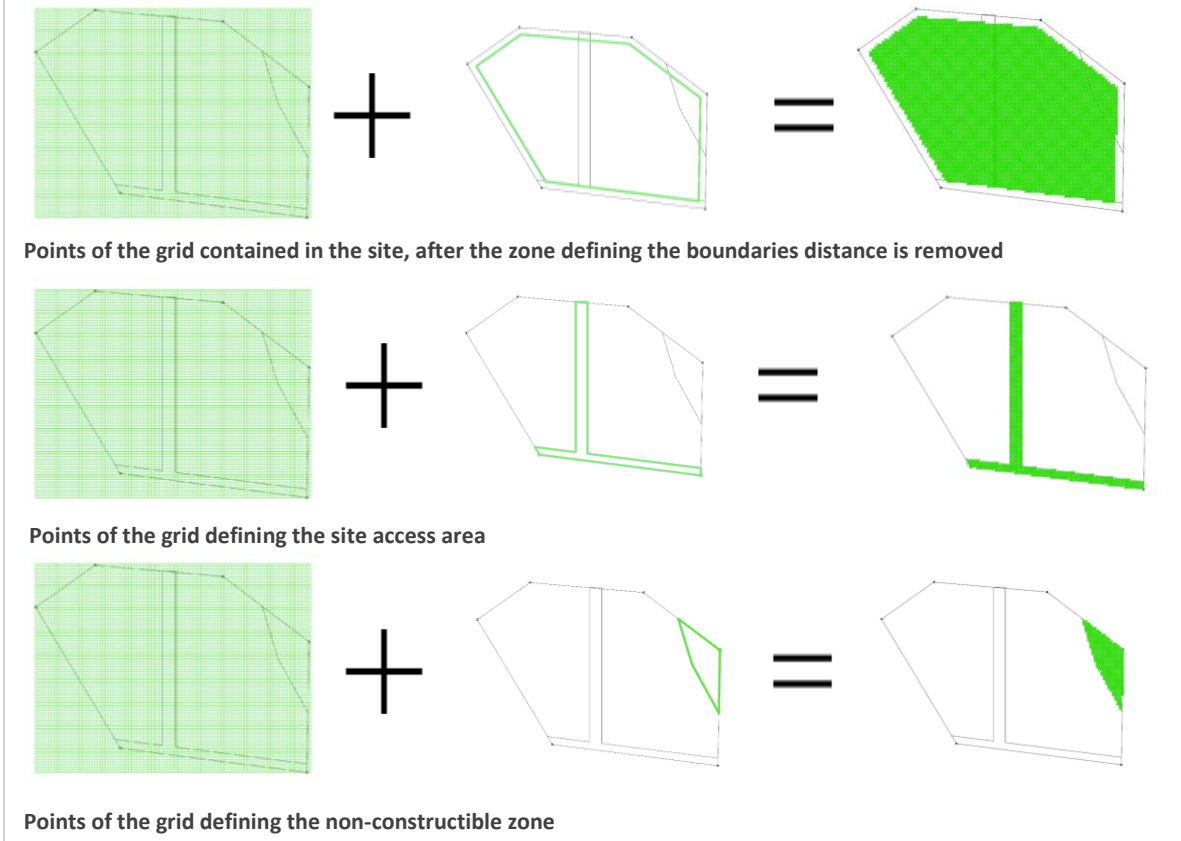


Figure 3.7: Grid Operations

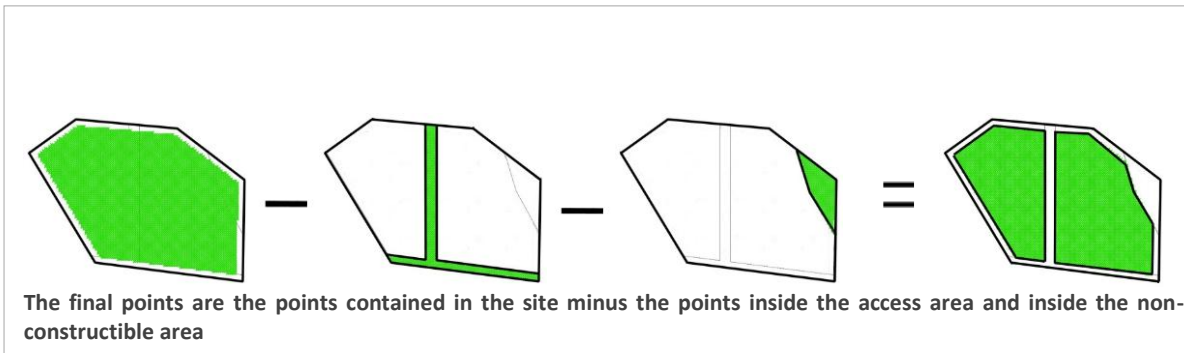


Figure 3.8: Calculation of grid points contained in constructible site

The points generated by this process are the ones that are going to be used for the buildings' placement on the site. Each one of them has an ID number - the Point ID -, which is used together with the angle to characterize the parameters (Bi(P,A)) of each building in every solution. The angle values range from 0 to 360. All the different combinations of location points and angles of each building produce the solution space.

The range of the parameter values defining the solution space is then provided to the solver (Galapagos) so that it can explore the solution space in order to find the optimal solutions. This range is given through the 'slider' component, a component that after setting a lower and upper limit produces a range of numbers. As the optimization runs the solver tries different values of this range and in this way explores the solution space (Figure 3.9).

For a specific input of e.g. a site containing 1000 location points and 3 buildings that should be located on it, the solver would need three sliders with a range from 1 to 1000 to locate the buildings and three sliders with a range from 0 to 360 to rotate them. The production of these sliders and the connection to the solver has to be done by the user, since their number is not predefined; it depends on the number of buildings.

In order to avoid the fact that the user would have to set manually the upper limit for the point slider, since the number of the points generated depends on the site input, the point list is mapped to a new list with a value range from 0 to 1 in a way that every value from 0 to 1 corresponds to one point ID. The angle slider has a fix upper limit independent of the specific problem at 360 degrees.

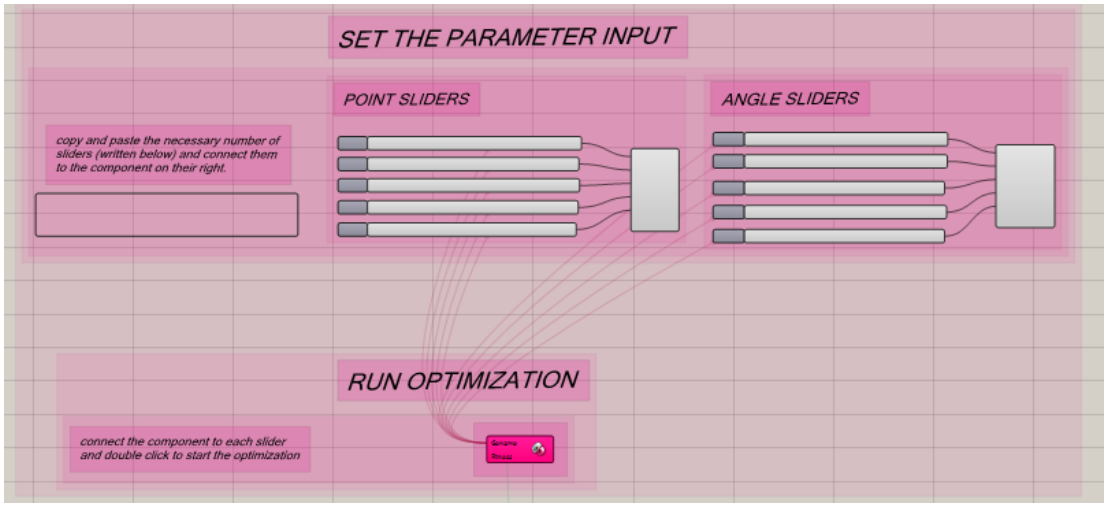


Figure 3.9: The variables that produce each solution

For every optimization iteration, these variables are the input to the solution generation and in this way produce one unique solution which is then subject to examination of whether or not it is successful enough.

Solution Generation

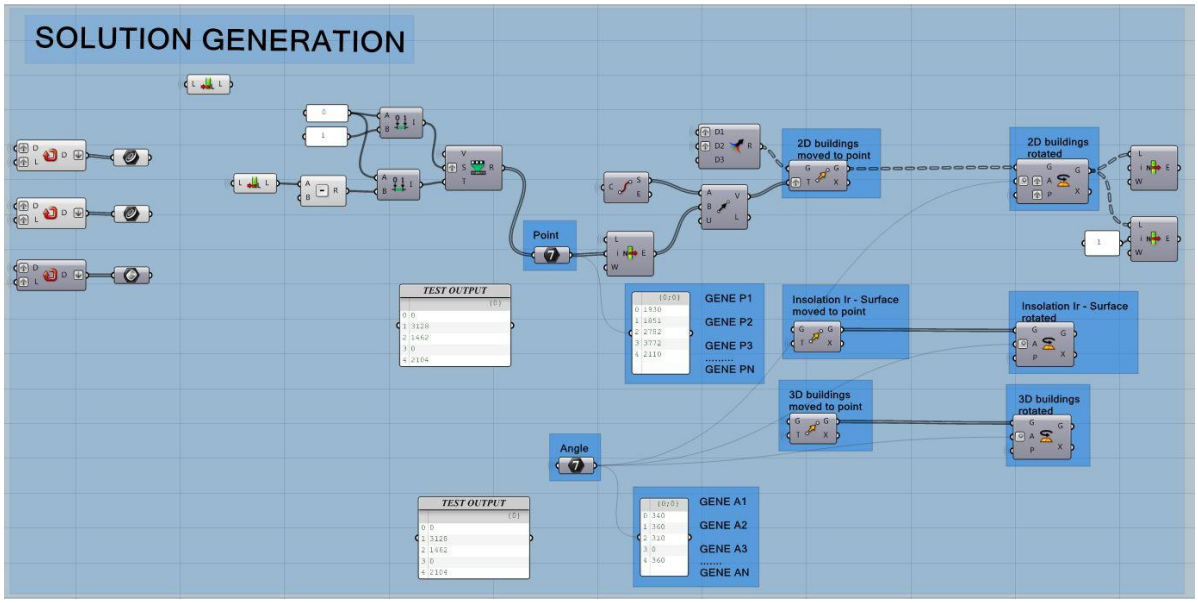


Figure 3.10: Generation of a solution

In order to produce one solution we need each building relocated from its input location to one of the points of the constructible site and rotated according to angle between 0 and 360 degrees, according to the values that the solver selects on every iteration while it is running.

Solution Evaluation

In order for a solution to be valid, it has to fit the spatial constraints as described in the system design.

Spatial Constraints

In order to ensure that the spatial constraints (constructible site containment, building zero overlap, accessibility requirements) are met, three functions are formulated (SC_A, SC_B, SC_C) that are resulting in a TRUE/FALSE condition and are later used to turn the final evaluation of the solution to zero when the constraints are not met.

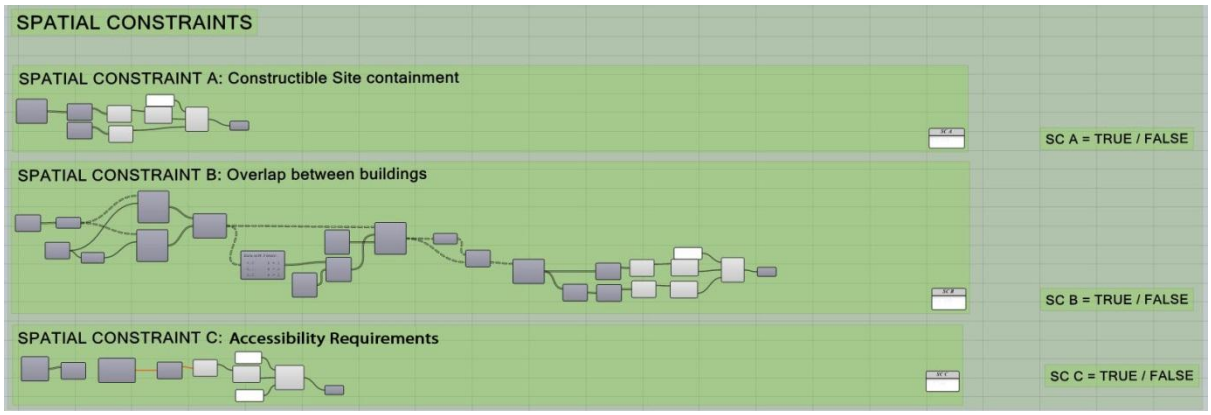


Figure 3.11: Spatial constraints definition

The operators needed for this functions are the boundary of the site constructible area (A_{SC}), the boundary of the site access area (A_{SA}), the boundaries of the buildings (A_B) and the boundaries of the buildings'access area (A_{BA}). Given these operators and the minimum distances between the buildings and the buildings and site, we can produce the rest of the operators required.
The boundary of the site constructible area (A_{CS}), is calculated by producing an offset boundary of the site ($A_{S'}$), with an offset value y, set by the user (Figure 3.12) and then subtracting from it the boundaries of the site access area and the non-constructible zone (A_{NCZ}). (Figure 3.13).

While calculating the buildings overlap with each other, the buildings' boundary is replaced by a new boundary ($A_{B'}$) created with an offset value set from the user. If x is the minimum distance that should be kept between the buildings, then x/2 is the offset distance of the boundary (Figure 3.12).

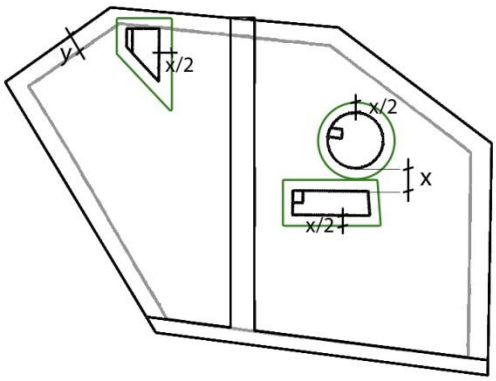


Figure 3.12: Minimum distances

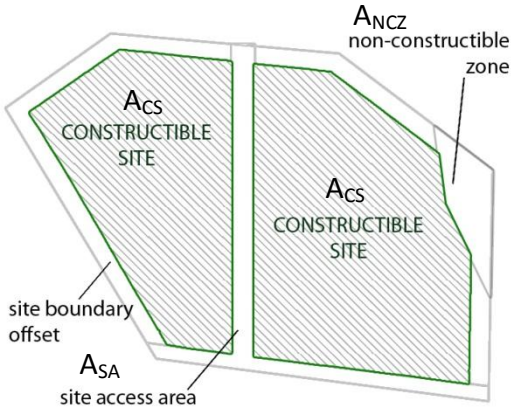


Figure 3.13: Constructible site

- Constraint A: Constructible site containment

The sum of the buildings boundaries (A_{Bi}) is compared with the intersection of their union with the constructible site. When these two values are equal then the function results in a TRUE value (equation (3.1)).TRUE always equals 1, while FALSE equals zero.

$$SC_A = \text{TRUE, when } A_{B1} + A_{B2} + \ldots + A_{BN} = (A_{B1} \cup A_{B2} \cup \ldots \cup A_{BN}) \cap A_{CS} \quad (3.1)$$

where N the number of buildings on site

A_{Bi} the area of building(i) floor plan, $i \in \{1,2,\ldots,N\}$

A_{CS} the site constructible area

$A_{CS} = A_{S'} \setminus (A_{SA} \cup A_{NCZ})$

$A_{S'}$ the area included in the boundary offset from the site to y,

y the minimum distance that the buildings have to keep from the site,

A_{SA} the area of the site access area,

A_{NCZ} the area of the non-constructible zone

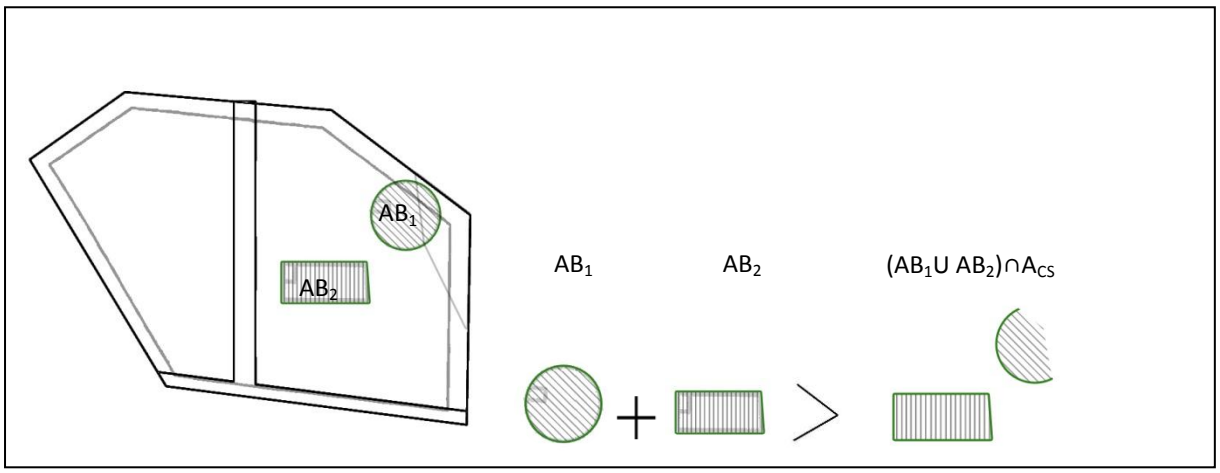


Figure 3.14: SC_A = FALSE

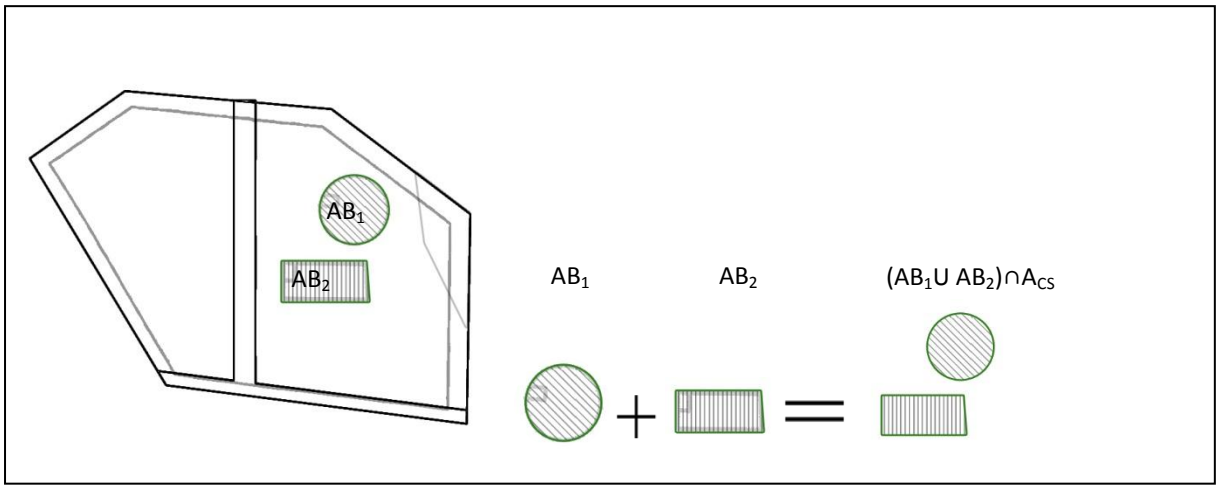


Figure 3.15: SC_A = TRUE

- Constraint B: Overlap between buildings

Even though the buildings overlap with each other is already controlled through Constraint A, the minimum distance between the buildings is not yet considered. Therefore Constraint B is implemented in order to ensure that minimum distance is always kept. The sum of the new buildings boundaries ($A_{Bi'}$) according to offset x/2, is compared with their union. When these two values are equal the function results in a TRUE value (equation (3.2)).

SC_B = TRUE, when $A_{B1'} + A_{B2'} + ... + A_{BN'} = A_{B1'} \cup A_{B2'} \cup ... \cup A_{BN'}$ **(3.2)**

where $A_{Bi'}$ is the area included in the boundary offset from the buildings boundary to x/2,
x is the minimum distance between two buildings,

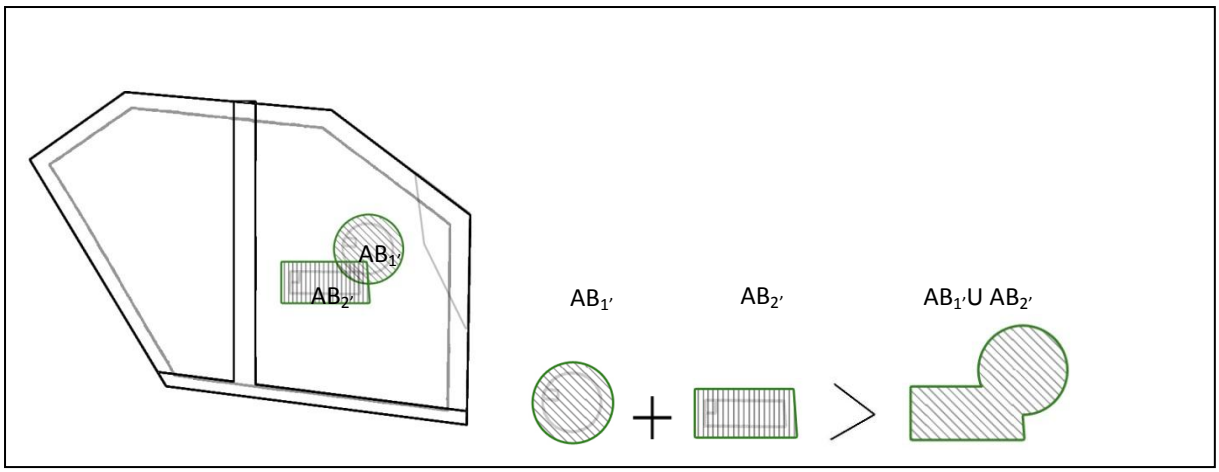


Figure 3.16: SC_B = FALSE

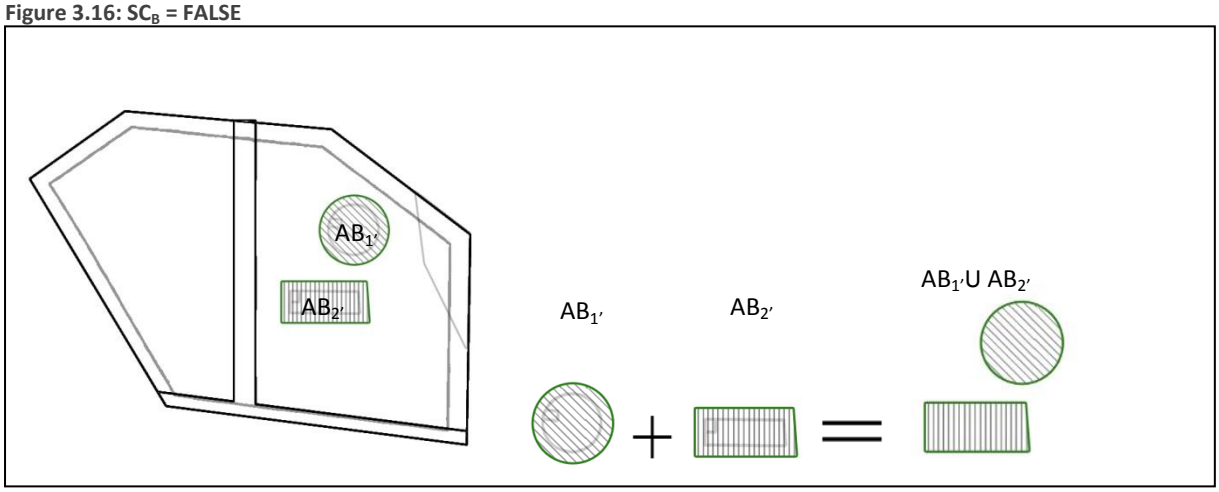


Figure 3.17: SC_B = TRUE

- Constraint C: Accessibility Requirements

The overlap between the buildings access area and the overlap zone should be at least 5% of the buildings access area, in order to certify that the accessibility requirements of the design are met.

SC_C = TRUE, when $A_{BAi} \cap A_{OZ} > 0{,}05 * A_{BAi}$

where A_{BAi} the area of a building(i) access area and A_{OZ} the area of the site overlap zone

(3.3)

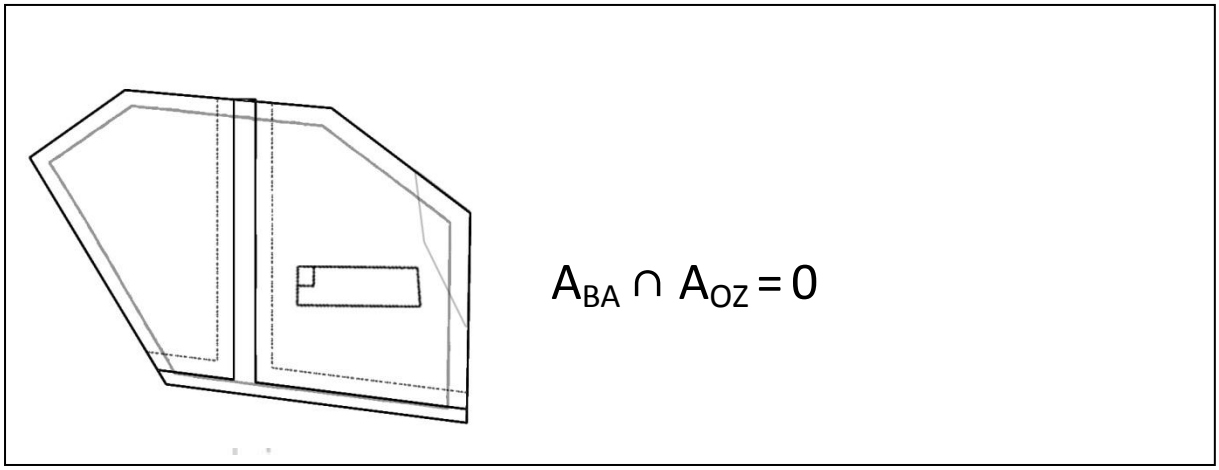


Figure 3.18: SC_C = FALSE

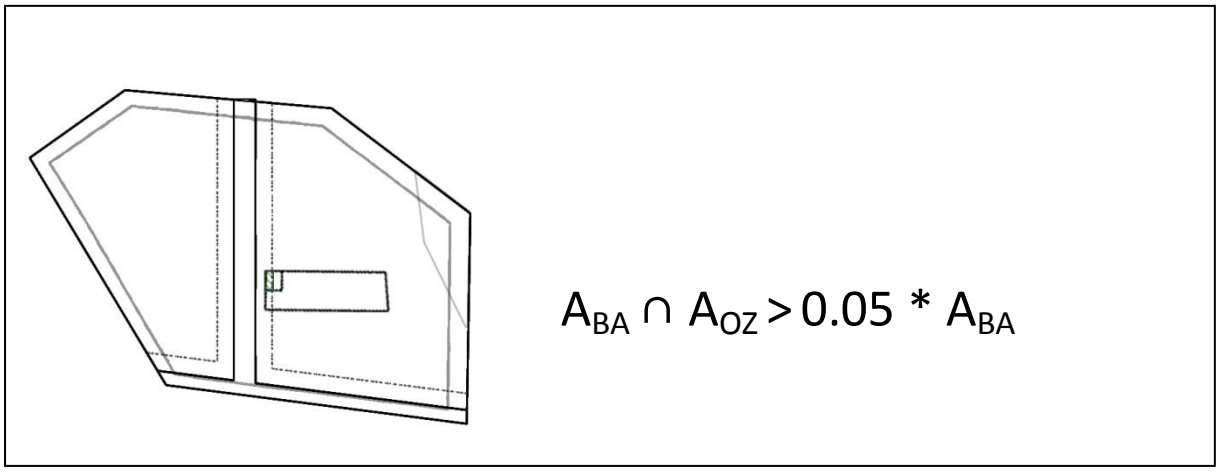


Figure 3.19: SC_C = TRUE

Evaluation Criteria

The value of each evaluation criterion (∑F) is a sum of the value of all buildings evaluation (F_i), as seen in equations (3.7) and (3.11). Each F_i should have a maximum value of 1 so that all criteria can be equally rated (equations (3.6),(3.10)). Thus the maximum value each ∑F will be N, where N is the number of buildings. The criteria can either be maximized or minimized according to the user's option settings.

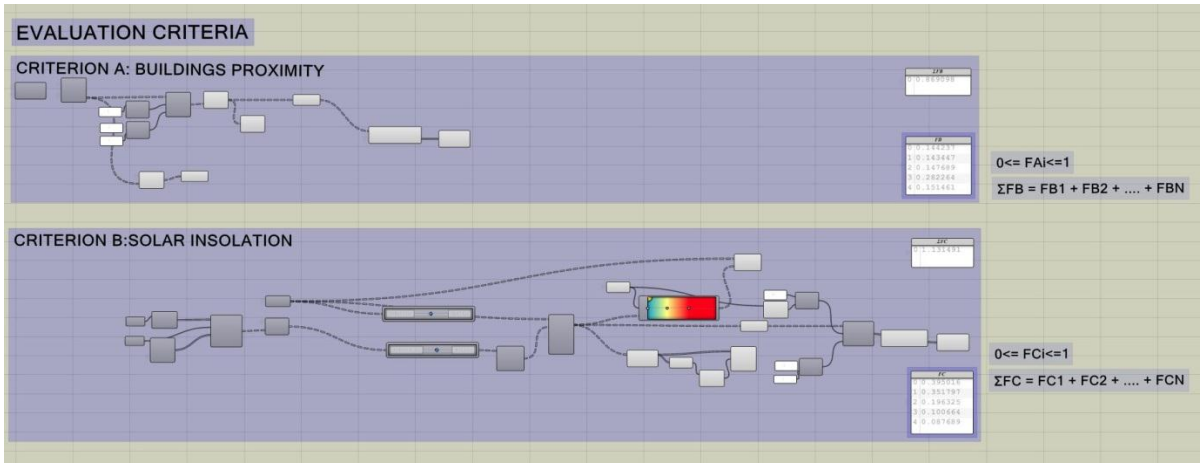


Figure 3.20: Evaluation Criteria Definition

- Criterion A: Buildings Proximity

The building proximity efficiency of each solution is evaluated for each building by measuring the distances between itself and the other buildings on the site. This sum of distances is then divided by the number of the other buildings to result in an average value of distance. This average distance is then further divided with the maximum distance between two points on the site so that it can always be expressed as a value between 0 and 1. The resulting value is the value of the evaluation criterion A for each building (FAi) when we want to minimize the buildings proximity, and is subtracted from 1 to give the final value in the case that we want to maximize the buildings proximity.
The maximum value is accomplished when all the buildings have an average distance from each other that equals the distance between two further points, when minimizing buildings proximity. When maximizing it then the maximum value appears for zero distances between the buildings (equation (3.5)).

$$FA_i = \begin{cases} 1-[((DB_1B_i + DB_2B_i + \ldots\ldots + DB_NB_i)/(N-1)) / DP_oP_M], \text{ when maximizing proximity} \\ ((DB_1B_i + DB_2B_i + \ldots\ldots + DB_NB_i)/(N-1)) / DP_oP_M, \text{ when minimizing proximity} \end{cases} \quad (3.4)$$

Where DB_iB_j the distance between building (i) and building (j), N the number of buildings and i, j Є (1, 2, …, N) and Po, Pm the two more distant points on the site

$$FA_{i\,max} = \begin{cases} 1-(0/(N-1))/ DP_oP_M=1, \text{ when maximizing proximity} \\ ((N-1)* DP_oP_M /(N-1))/ DP_oP_M=1, \text{ when minimizing proximity} \end{cases} \quad (3.5)$$

$$0 \le FA_i \le 1 \quad (3.6)$$

$$\sum FA=FA_1 + FA_2 + \ldots\ldots + FA_N \quad \text{, for N number of buildings} \quad (3.7)$$

- Criterion B: Solar Insolation

The solar insolation is evaluated for each building in a very rough way that is not result of proper solar analysis but can give some rough basis for the evaluation. The sun rays vectors are produced by calculating the sun position on an annual basis and according to the user's input of longitude, latitude, time zone and north angle. The calculation algorithm is produced by Ted Ngai (http://www.tedngai.net/experiments/incident-solar-analemma.html) and is based on the algorithm published by the National Oceanic and Atmospheric Administration (NOAA).The values in the system do not consider the scattering and absorption effects such as water vapor, ozone and aerosol; therefore the numbers are not accurate and can only provide a rough approximation.
Each building's exposure to the sun is then calculated as the sum of sines of the angles between the sun ray vectors and the building. The maximum value would be accomplished for a horizontal surface, while 0 would be the number of rays passing through a north-facing façade, or a façade behind another building, in the case that we are trying to maximize solar insolation, and vice versa in the case we are trying to minimize solar insolation.

$$FB_i = \begin{cases} \Sigma\sin(a)/N_R, \text{ when maximizing solar insolation} \\ 1 - \Sigma\sin(a)/N_R, \text{ when minimizing solar insolation} \end{cases} \quad (3.8)$$

,where a the angle between the sun ray vector and the building and N_R the total vectors of the sun analemma produced

$$FB_{i\,max} = \begin{cases} N_R / N_R = 1, \text{ when maximizing solar insolation} \\ 1\text{-}0/N_R = 1, \text{ when minimizing solar insolation} \end{cases} \quad (3.9)$$

$$0 \leq FB_i \leq 1 \quad (3.10)$$

$$\sum FB = FB_1 + FB_2 + \ldots\ldots + FB_N \text{ , for N number of buildings} \quad (3.11)$$

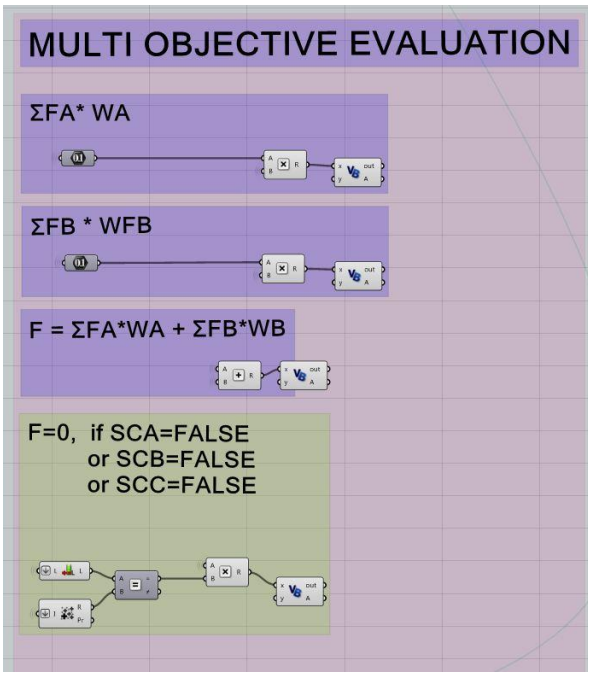


Figure 3.21: Multi Optimization Function

M.O. Optimization

Since the optimization solver can only evaluate one number and cannot set any constraints, a function has to be formulated, that combines all Evaluation Criteria and keeps the Spatial constraints, resulting in one final evaluation value.
First of all a weight for each evaluation criterion is set, so that the user can decide whether or not this criterion is important for the design and if yes, then to what percentage, related to the other criteria. The weights have a value from 0 to 1(equation (3.12))
The weighted evaluation values (∑F'), are then added resulting in the total evaluation value F_{Final} (equation (3.14)).
The value of F_{Final} is set to zero when the spatial constraints are not met; thus all zero values represent invalid solutions. This final value F_{Final}' is being evaluated by the solver through the optimization iterations (equation (3.13)).

$$\sum F' = \sum F * w \text{ , where } 0 \le w \le 1 \text{ the weight of the criterion} \quad (3.12)$$

$$F_{Final} = \sum F_A' + \sum F_B', \text{ for criteria A and B} \quad (3.14)$$

$$F_{Final}' = F_{Final} * SC_A * SC_B * SC_C \quad (3.13)$$

3.2.3 Output

The output is given in two ways: by documenting the results and by visualizing them on the screen

Documentation

The results are documented in an excel file, whose location and name is given by the user. The data are sorted in such a way that each row is a solution and each column displays the solution ID number (the numbering is chronological), the final evaluation value, the evaluation of each criterion and the solution variables (point and location for each building)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	SOLUTION_ID	FITNESS	GENE_P1	GENE_P2	GENE_P3	GENE_A1	GENE_A2	GENE_A3	FA (ACC)1	FA (ACC)2	FA (ACC)3	FB (PROX)1	FB (PROX)2	FB (PROX)3	FC (EXP)1	FC (EXP)2	FC (EXP)3
2	1	0	0	822	1462	58	0	0	0	0,42783	1	0,588838	0,785006	0,767976	0,05803	0,09513	0,06164
3	2	0	0	822	1462	58	0	0	0	0,42783	1	0,457711	0,6873	0,746559	0,05803	0,09513	0,06164
4	3	0	0	822	1462	58	0	0	0	0,42783	1	0,590862	0,78668	0,774125	0,05791	0,09556	0,06197
5	4	0	0	822	1462	58	0	0	0	0,42783	1	0,590862	0,78668	0,774125	0,05803	0,09589	0,06292
6	5	0	0	822	2113	58	0	0	0	0,42783	1	0,568367	0,753088	0,6581	0,05803	0,09589	0,06292
7	6	0	0	822	2113	58	0	0	0	0,42783	0	0,566343	0,751415	0,651467	0,05803	0,09511	0,0668
8	7	0	0	822	2113	58	0	0	0	0,42783	0	0,438145	0,542444	0,444197	0,05803	0,09435	0,06663
9	8	0	0	3660	2113	58	0	0	0	0,42783	0	0,49772	0,595297	0,695692	0,06147	0,0962	0,07051
10	9	0	0	3660	2113	58	0	0	0	0	0	0,374457	0,396148	0,466157	0,04935	0,08435	0,05813
11	10	0	0	3660	2113	58	0	0	0	0	0	0,500872	0,595096	0,695958	0,05087	0,08456	0,07006
12	11	0	0	3660	2059	58	0	0	0	0	0	0,511117	0,579602	0,676238	0,04935	0,08435	0,05813
13	12	0	3311	3660	2059	58	0	0	0	0	0	0,637838	0,678266	0,751298	0,04921	0,08469	0,07605
14	13	0	3311	3660	2059	58	0	0	0	0	0	0,627325	0,668396	0,740636	0,05617	0,08262	0,07914
15	14	0	3311	4706	2059	58	0	0	0	0	0	0,470445	-0,25	0,560782	0,05639	0,08262	0,07892
16	15	0	3311	3660	2059	58	0	0	0		0	0,605413	0,616614	0,702385	0,05556	0	0,07797
17	16	0	3311	4706	2059	58	0	0	0	0	0	0,470445	-0,25	0,560782	0,05648	0,08254	0,07914
18	17	0	3311	3660	2059	0	0	0	0		0	0,723387	0,702516	0,776187	0,05556	0	0,07797
19	18	0	3311	3660	2059	58	0	0	0	0	0	0,599859	0,627475	0,709608	0,0801	0,08492	0,07186
20	19	0	440	3660	2059	58	104	0	0	0	0	0,598278	0,644423	0,727883	0,05624	0,08281	0,07818
21	20	0	440	712	2059	58	104	0	0	0	0	0,676029	0,804592	0,697967	0,06879	0,03982	0,07329

Figure 3.22: Documentation file

Visualization

After documenting the results the user can choose to display the ranked according to efficiency solutions on the screen in order to visualize the output. The ranking can be according to the total evaluation value or according to the evaluation value of one of the criteria. The user can select to dislpay a desired number of solutions, not necessarily continuously but also every x solution and can also select which of the solutions characteristics should be displayed (parameters, evaluation criteria values, etc).

There is also a set of options regarding graphical aspects like for example the scale of the displayed solutions the distance that should be kept between them etc., since these depend on the size of each site given as output and should be easily manipulated by the user.

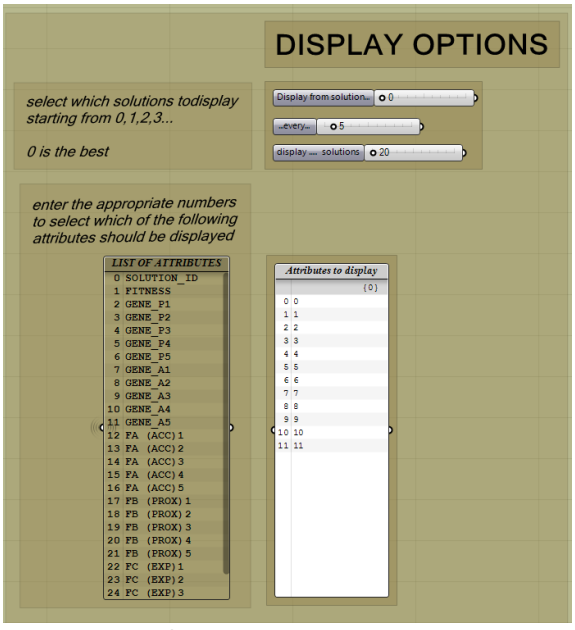


Figure 3.23: Display options

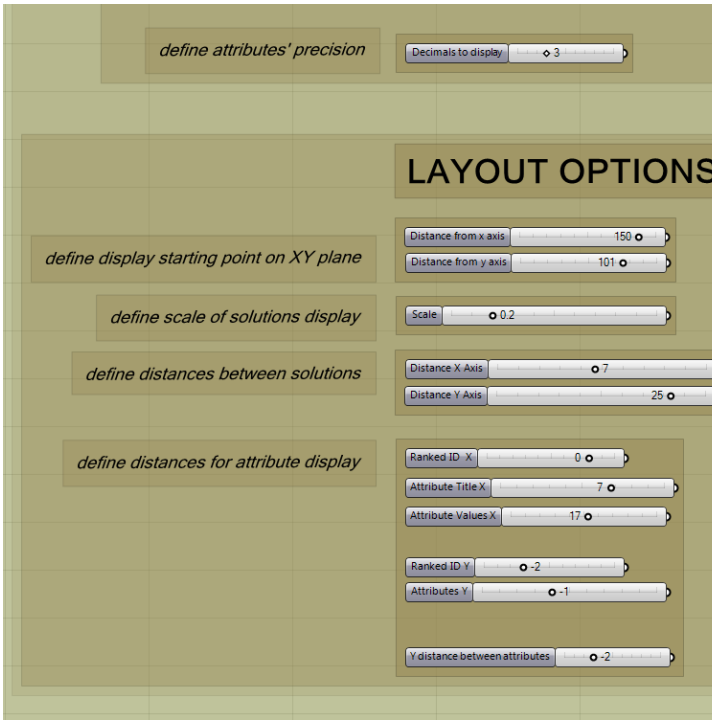


Figure 3.24: Visualization options

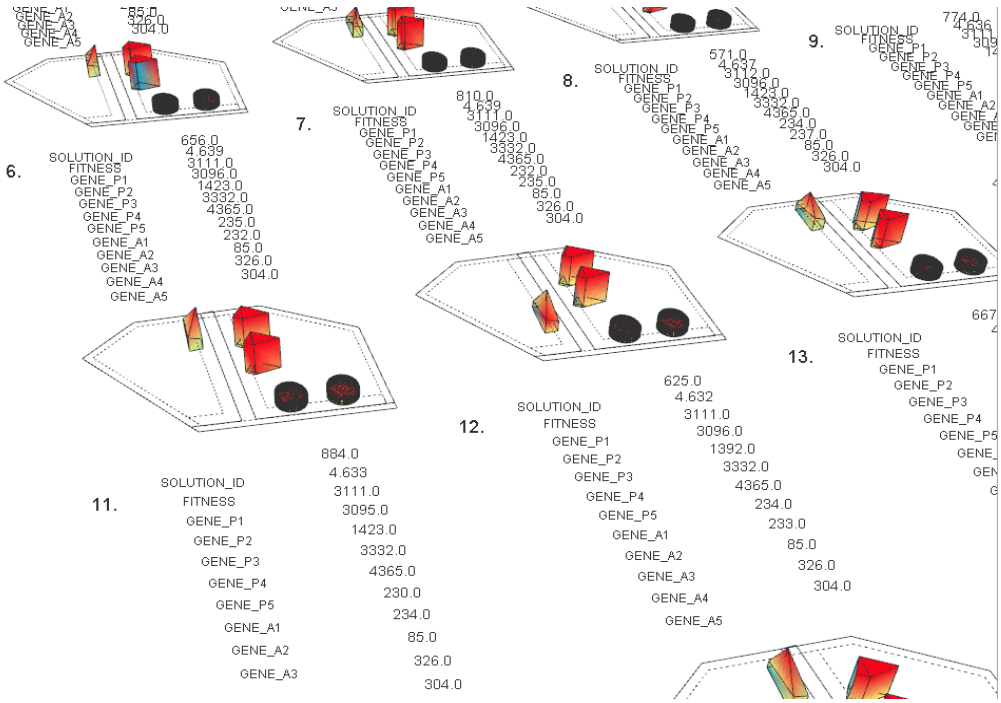


Figure 3.25: Perspective display of output

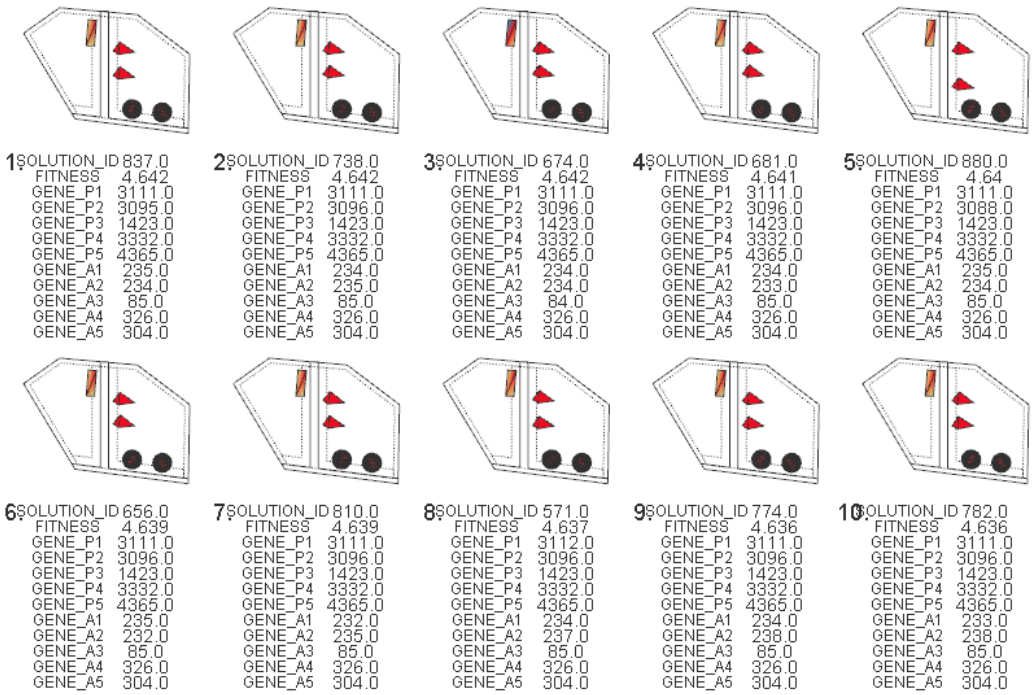


Figure 3.26: Floor plan display of output

3.3 Fine-tuning

3.3.1 Experimenting with Spatial Constraints

Efficiency of Spatial Constraints

The accessibility constraint that turns the final evaluation into zero when no overlap between the building access area and the overlap zone exists turned out after several experiments to be a major drawback for the efficiency of the algorithm. After several runs of both the Genetic and the Simulated Annealing solver, no positive evaluation output came up.

The problem is that the case that all buildings display at least 5% overlap, is extremely rare and until one such solution is found, the algorithm wanders around in the solution space, receiving no feedback about whether or not it is getting closer to a better solution.

'Bonus' in Evaluation Value when Spatial Constraints are met

While trying to improve the system performance the following alternative was tested:

Instead of turning the total evaluation of the solution to zero, the overlap of the buildings' access area with the overlap zone is measured and added as a positive value to the sum of the evaluation. In this way, the solutions that do not display the desired overlap, but meet the other two constraints, can still be positively evaluated and forward the solution search of the solver, while an extra 'bonus' value would lead the solver to find the solutions that satisfy the third constraint as well. In order to maintain the importance of the evaluation criteria as guidelines for the solver, this extra value is translated within a numerical range that has a lower maximum value, than the evaluation criteria. Since the value range for each criterion is set to [0,1], the value range assigned to the constraint is [0,0.1]. In order to verify whether or not such a variation would work correctly, some experiments were done. The outcome of two of them (one for each solver) is seen on the images and diagrams below. The images show the best 25 solutions of each experiment and the diagrams show the evaluation values along a timeline. The weight of the insolation criterion is turned into zero, in order to minimize calculation times. The type of criteria driving the optimization does not affect the efficiency of the spatial constraints.

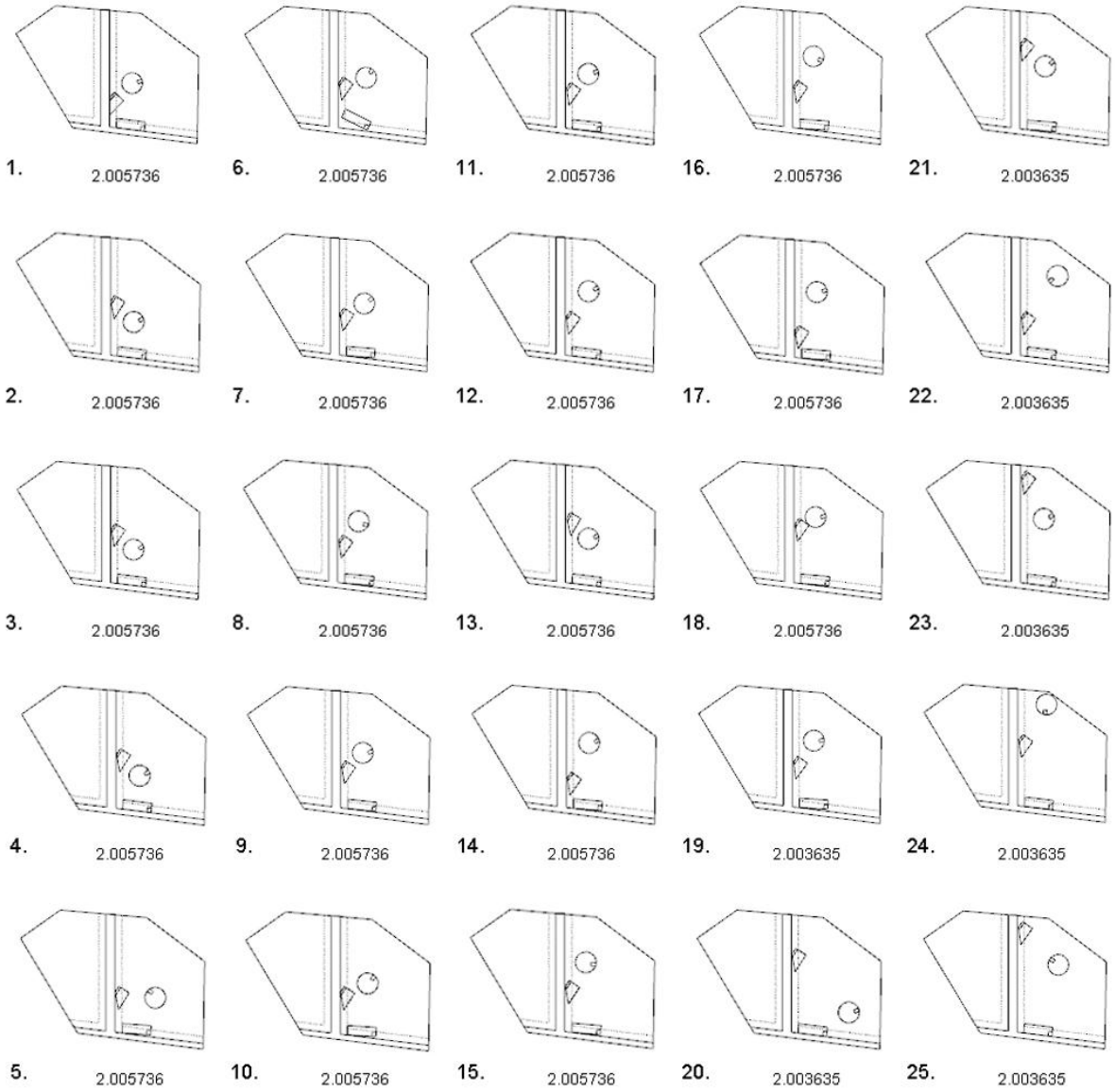


Figure 3.27 : 25 best solutions produced by the GA solver for constraint C acting as a 'bonus' value

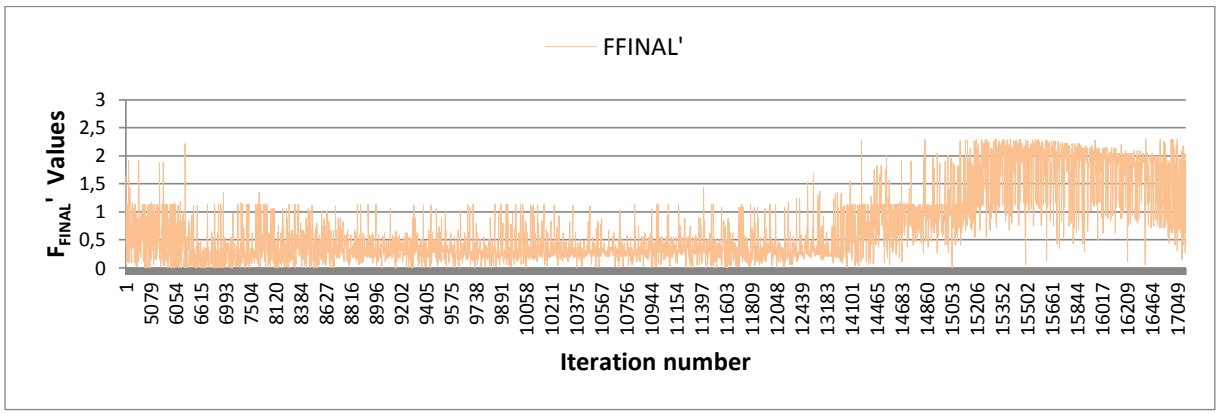


Chart 3.1: Evaluation values (F_{FINAL}') of EA solver for constraint C acting as a 'bonus' value

The algorithm terminated after running for 13h and finding 4235 solutions above zero after 17226 iterations. 27 of them have a value above 2 (maximum F_{FINAL}' =3,3). Even the ones with the higher values are not valid ones, since they do not have all buildings placed in the overlap zone. The criterion C is not met.

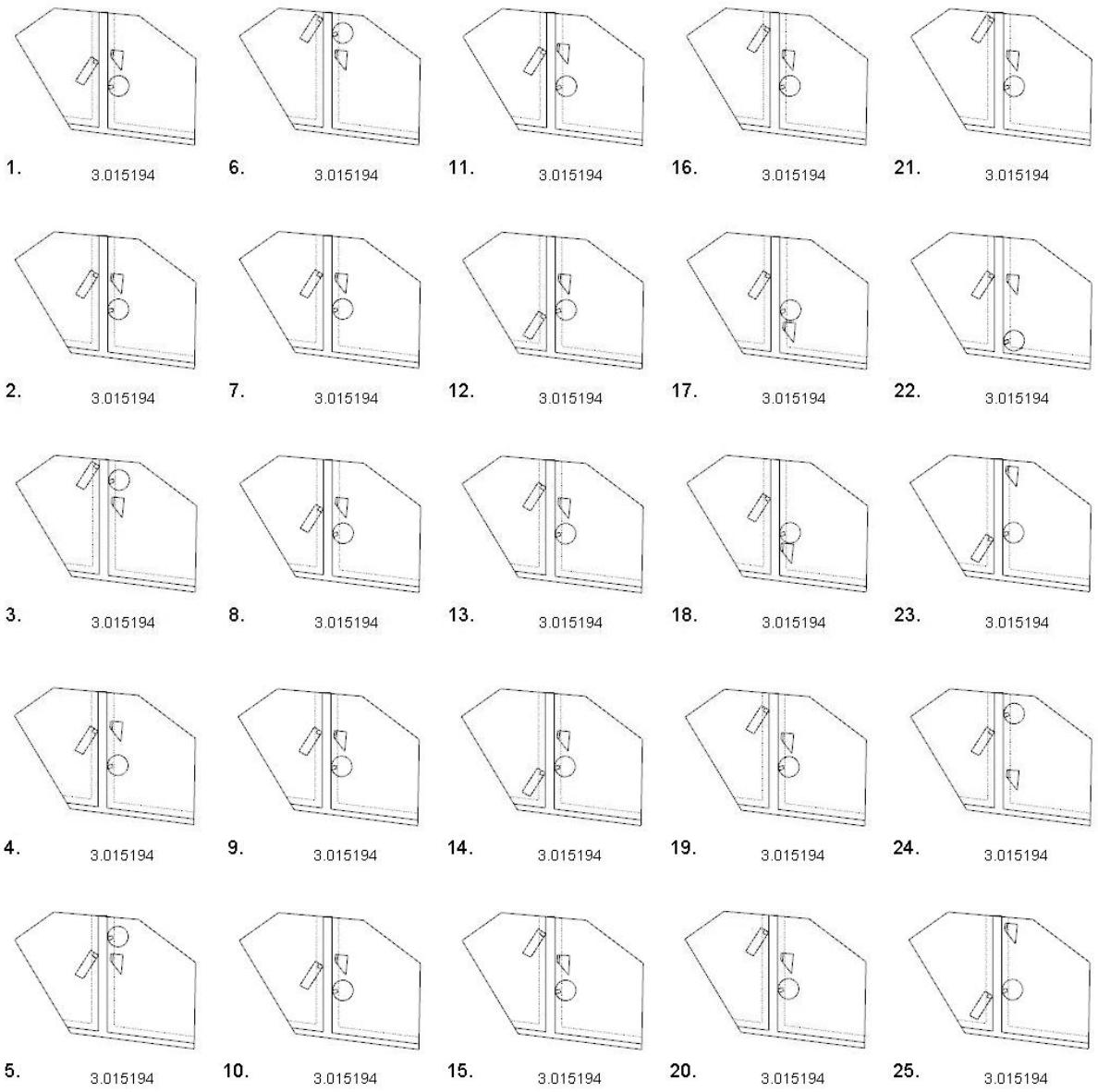


Figure 3.28: 25 best solutions produced by the SA solver for constraint C acting as a 'bonus' value

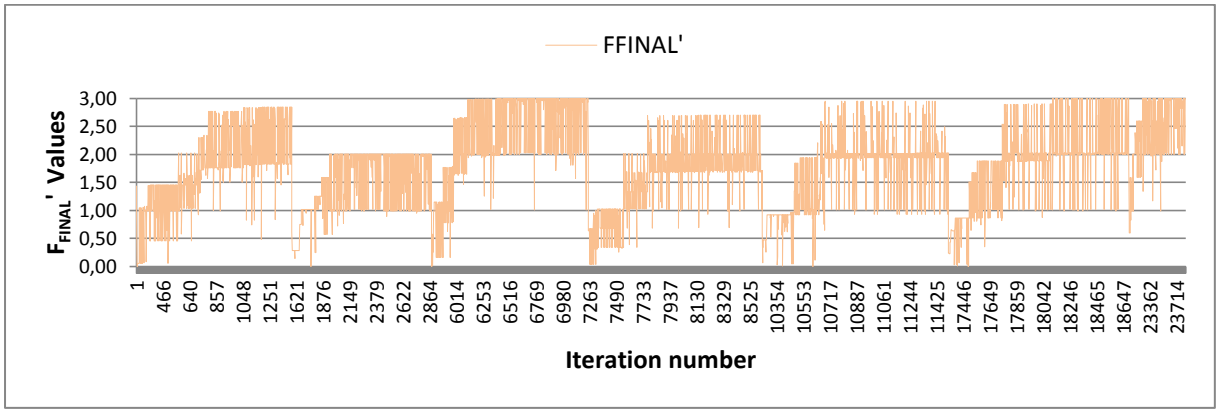


Chart 3.2 : Evaluation values (F_{FINAL}) of SA solver for constraint C acting as a 'bonus' value

The algorithm ran for 13h finding 3935 solutions above zero in 23816 iterations. 1815 of them have a value above2 (maximum F_{FINAL}' =3,3). That doesn't mean though that all 3935 solutions are valid solutions since the fulfillment of constraint C isn't a restriction. We can see though that, at least in the best solutions, all buildings

are placed inside the overlap zone; thus there are valid solutions produced but their number cannot be directly calculated.

Distance from path used as a guideline

Even though the SA algorithm did manage to find proper solutions without having Constraint C setting the evaluation to zero, it was considered more appropriate to find a way to implement the constraints into the system in a way that we can have a clear expression of the results according to whether a solution is valid or not.
Therefore criterion C was once again activated to turn the evaluation to zero and a workaround was implemented to ensure that the algorithm receives some sort of feedback while spatial constraints are not met.

This feedback is the distance that each building has from the site access boundary. This distance is given a negative sign and added to the final evaluation value. That way instead of a constant zero value, when the performance criteria are not met, now the algorithm receives a negative value and tries to maximize it by bringing the buildings closer to the site access path. This distance acts as a negative evaluation criterion, in the sense that it doesn't actually evaluate a valid solution but an invalid one in order to improve it. As soon as the spatial constraints are met the negative distance value is no longer added to the total evaluation value, so that it won't alter the evaluation results.

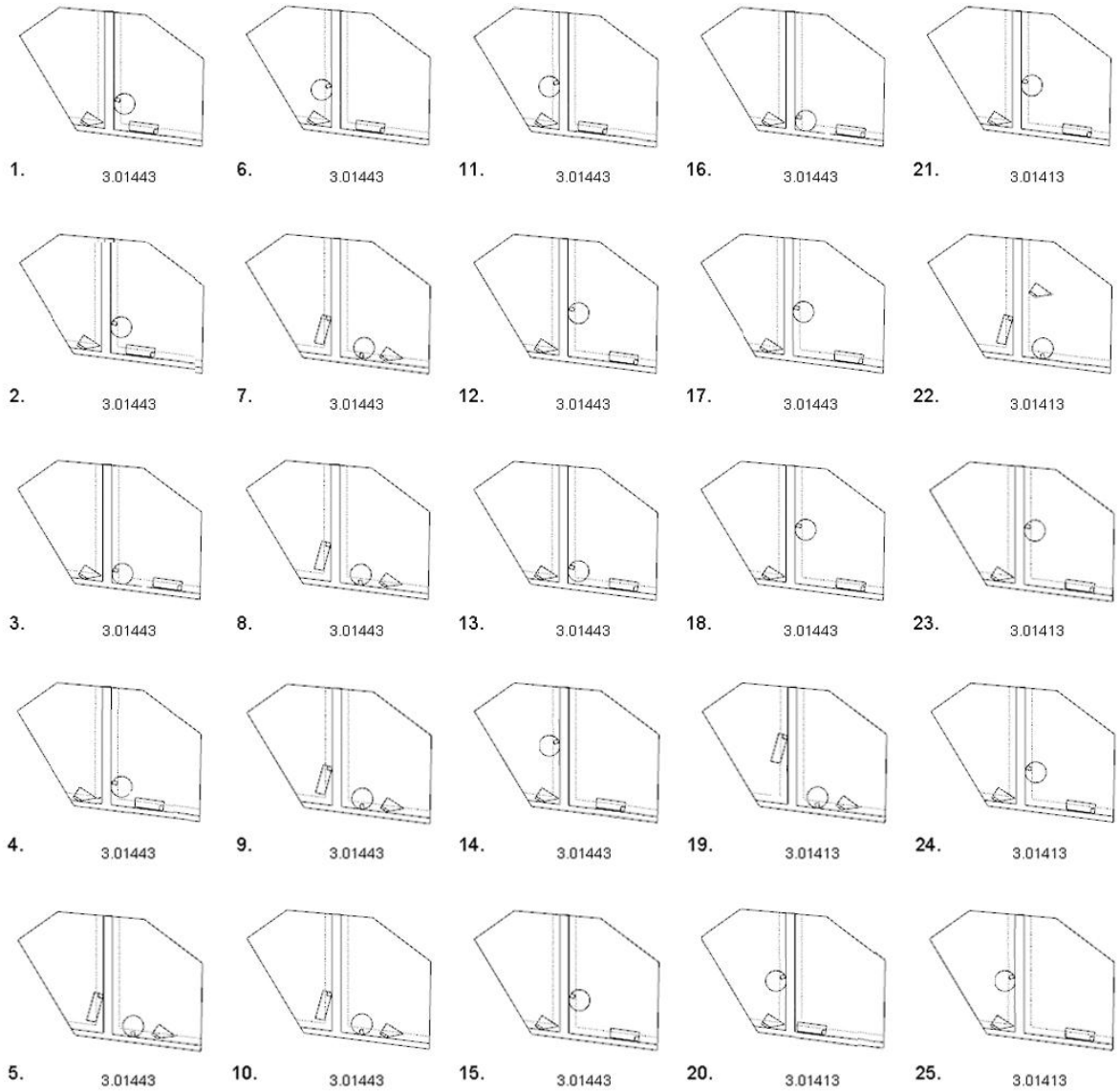


Figure 3.29: 25 best solutions produced by the SA solver while feedback for fulfillment of SC_C is given

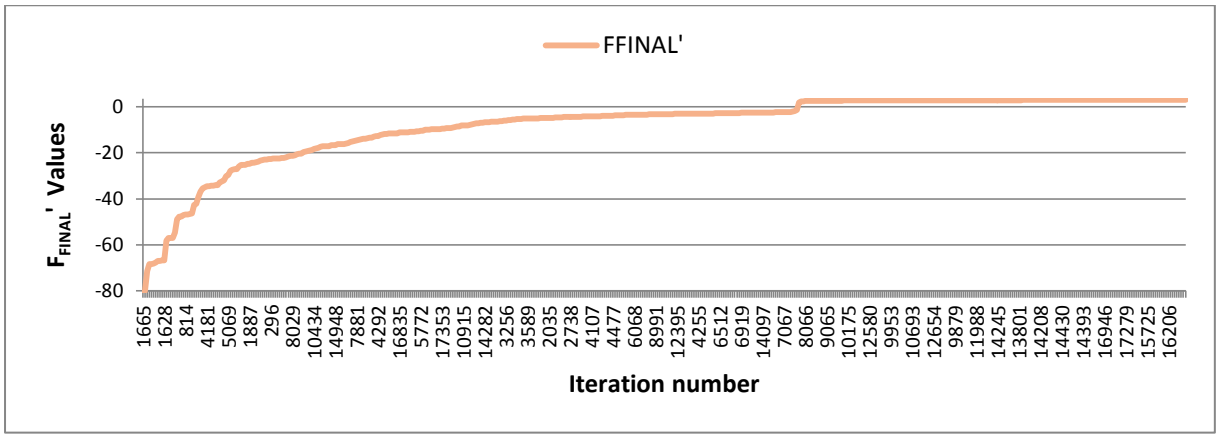


Chart 3.3: Evaluation values (F_{FINAL}') of SA solver for constraint C acting as a negative evaluation criterion. All solutions displayed

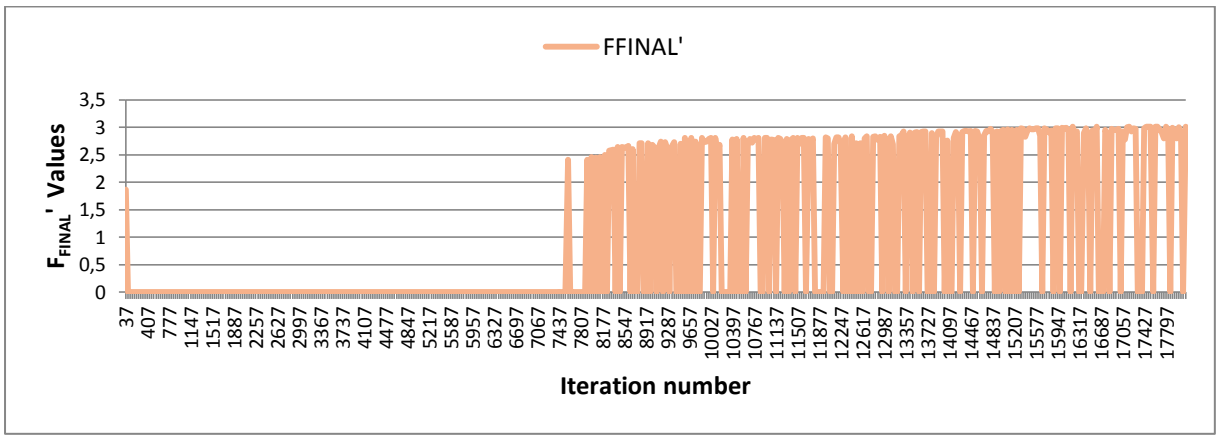


Chart 3.4: Evaluation values (F_{FINAL}') of SA solver for constraint C acting as a negative evaluation criterion. Only valid solutions displayed

The algorithm ran for 10h finding 5409 valid solutions in 18106 iterations. 393 of them have a value above2 (maximum F_{FINAL}' =3,3). All solutions above zero are valid solutions that meet all three constraints

Alteration of Spatial Constraint C

Since the buildings' distance from the site access path also controls the accessibility requirements, it is considered unnecessary to maintain the operator that controls the overlap of the building's access area with the overlap zone. Therefore the accessibility requirements constraint is no longer given by measuring the containment of the buildings' access area in the overlap zone" but, by measuring the distance between the building's access area and the site access area (d). The constraint leads to a valid solution if this distance is smaller than the longer edge of the building access area's bounding polygon (L). In order to forward further flexibility in the accessibility requirements, this constraint is controlled by

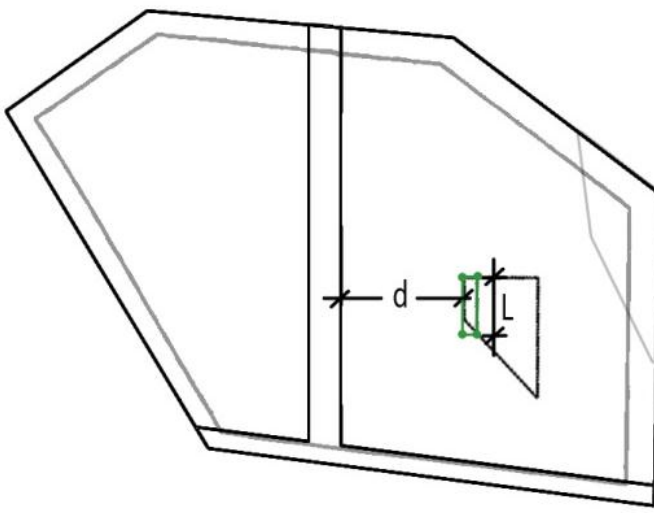


Figure 3.30: Accessibility Constraint Operators

a weight parameter by the user. This accessibility weight "loosens" the constraint with a value range from 0-100. 0% constraint relaxation means that the constraint acts as described previously and results in a solution where the building is adjacent to the site access area. 100% relaxation means that the building is allowed to be as far away from the path as possible. All the intermediate values result in respective way.

SPATIAL CONSTRAINT C: ACCESSIBILITY REQUIREMENTS:

SC_C = TRUE, when $d < f(w) * L$
$f(w)=((D-L)*w)/100+L$ **(3.15)**

where d the distance between the building's access area and the site access area
L the longest edge of the building access area bounding polygon
D the longest one of the distances between the points of the constructible site and the site access area
$0 \leq w \leq 100$ the weight controlling the constraint's relaxation

Negative Evaluation Criteria

After some experiments done for a relative big number of buildings the algorithm's performance was still low because of the restrictions imposed by the other two spatial constraints: For a problem of locating many buildings into a relative small site (high density) the solver easily got lost without finding positive solutions, since the buildings almost always would overlap with each other and return 0 as an evaluation value without helping the algorithm orient in the solution space. Therefore the same logic was once again applied here: The overlap of the buildings with each other and also with the non-constructible area of the site is given a negative sign and added to the final evaluation value. The site's non-constructible area A_{NCS} is the union of a)the site access area, b)the non-constructible zone and c) the difference of the area of the site's bounding box minus the site offset to y (y is the minimum distance that should be kept from the site boundaries). As soon as the spatial constraints are met, the overlaps turn, by definition, into zero and can be added to the final result without altering it.

Thus the system is now producing three more values: NC_A, NC_B, NC_B, the negative evaluation criteria. Each one of them corresponds to one of the spatial constraints and mainly expresses how far the solution is from the constraint's fulfillment. They all have negative or zero values, and they are added to the final evaluation value.

$NC_A = - \Sigma(A_{Bi} \cap A_{NCS})$ **(3.16)**

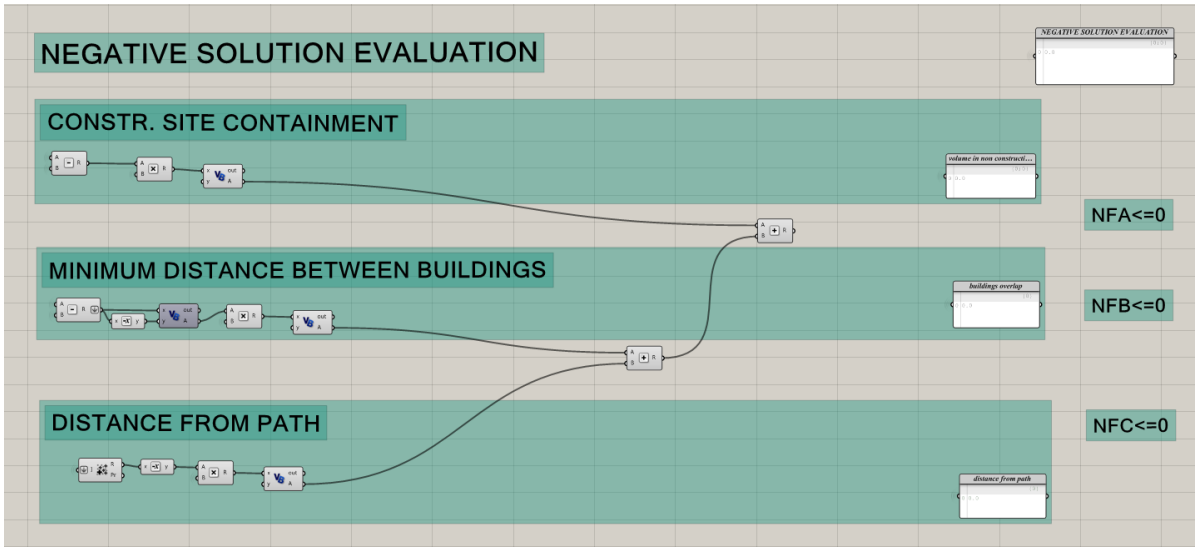
where A_{Bi} the area of building(i) floor plan, $i \in \{1,2,\ldots,N\}$
N the number of buildings on site
A_{NCS} the site non constructible area

$NC_B = - [(A_{B1'} + A_{B2'} + \ldots + A_{BN'}) \cap (A_{B1'} \cup A_{B2'} \cup \ldots \cup A_{BN'})]$ **(3.17)**

where $A_{Bi'}$ is the area included in the boundary offset from the buildings boundary to x/2, $i \in \{1,2,\ldots,N\}$,N the number of buildings on site
x is the minimum distance between two buildings,

$$NC_C = \begin{cases} -d & \text{,when } SC_C = \text{FALSE} \\ 0 & \text{,when } SC_C = \text{TRUE} \end{cases}$$ **(3.18)**

where d the distance between the building's access area and the site access area



The system performance was tested for 5 buildings, while both criteria weights were set to 1 and so as to maximize both buildings proximity and solar insolation.

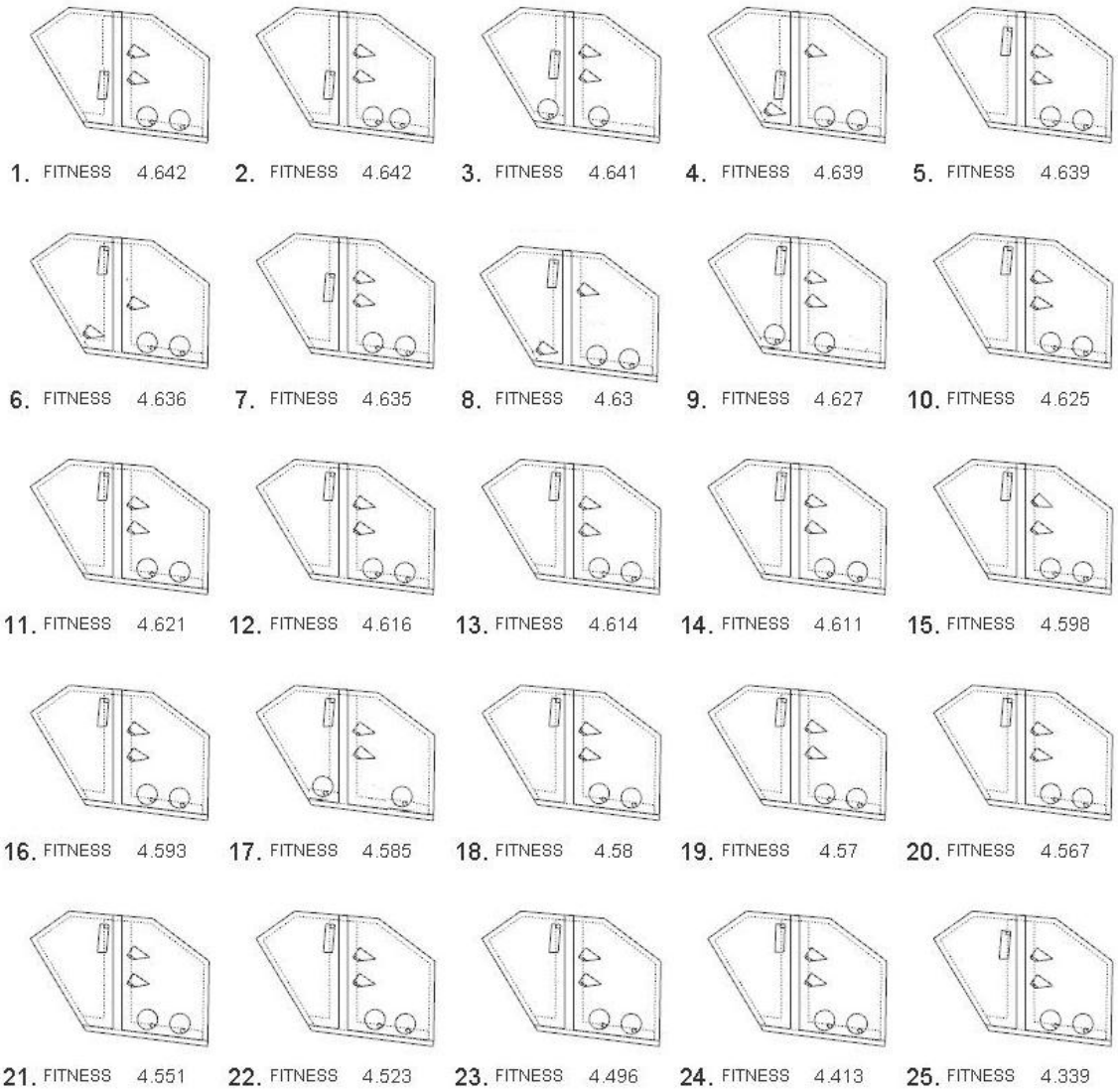


Figure 3.31: 25 samples of the 200 best solutions produced by the SA solver after implementing negative evaluation criteria

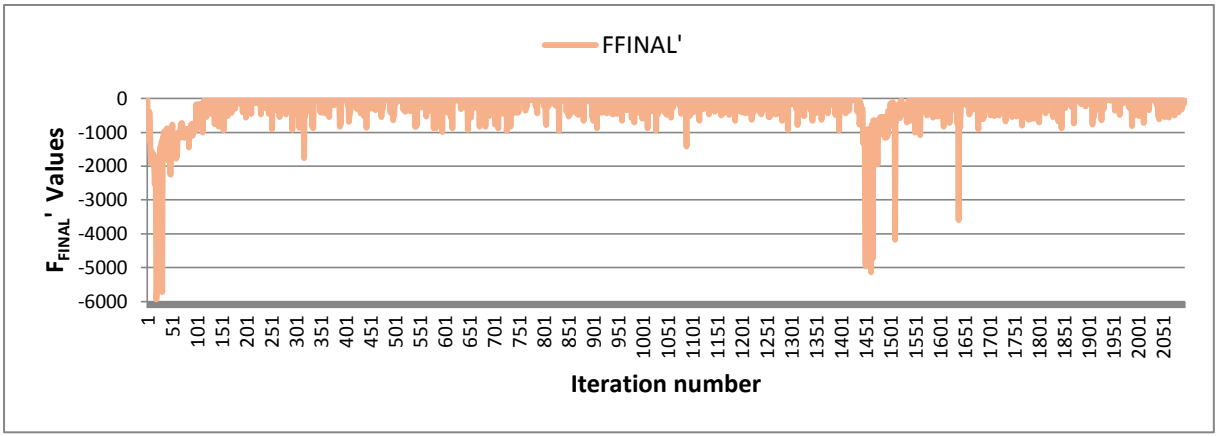


Chart 3.5: Evaluation values (F_{FINAL}') of SA solver after implementing negative evaluation criteria. All solutions displayed.

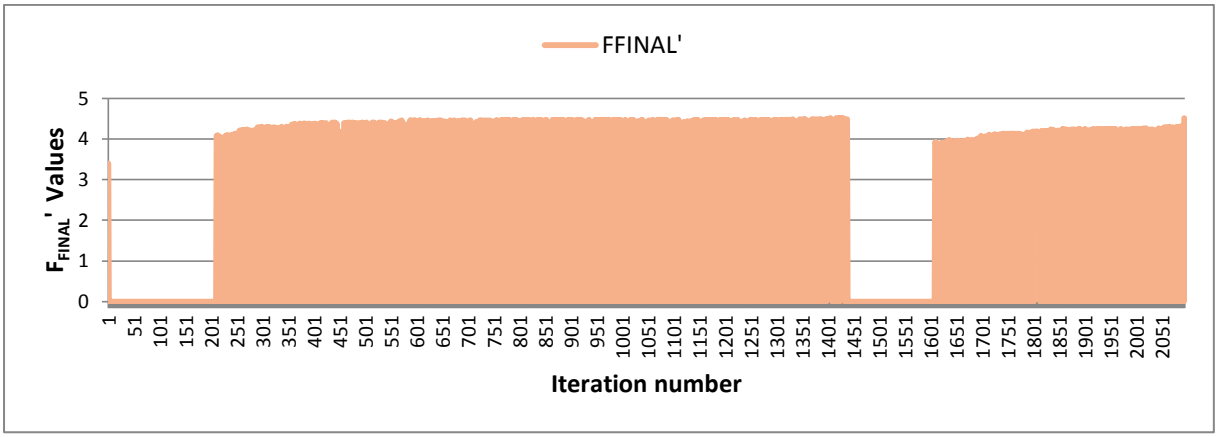


Chart 3.6: Evaluation values (F_{FINAL}') of SA solver after implementing negative evaluation criteria. Only valid solutions displayed

The algorithm ran for 13h and finding 852 valid solutions in 2094 iterations.

3-Dimensional spatial constraints

The spatial constraints as already applied, express the relationships between the buildings and the site on two-dimensional space, by controlling overlap and containment constraints through their floor plans. For irregular building geometries, where the building volume doesn't exactly correspond to the floor plan, it is possible that the buildings overlap with each other or that they are not contained in the site even if the spatial constraints are met. Therefore the two first criteria should be adjusted in order to provide valid solutions independently of the geometry input. The negative evaluation criteria are also respectively adjusted.

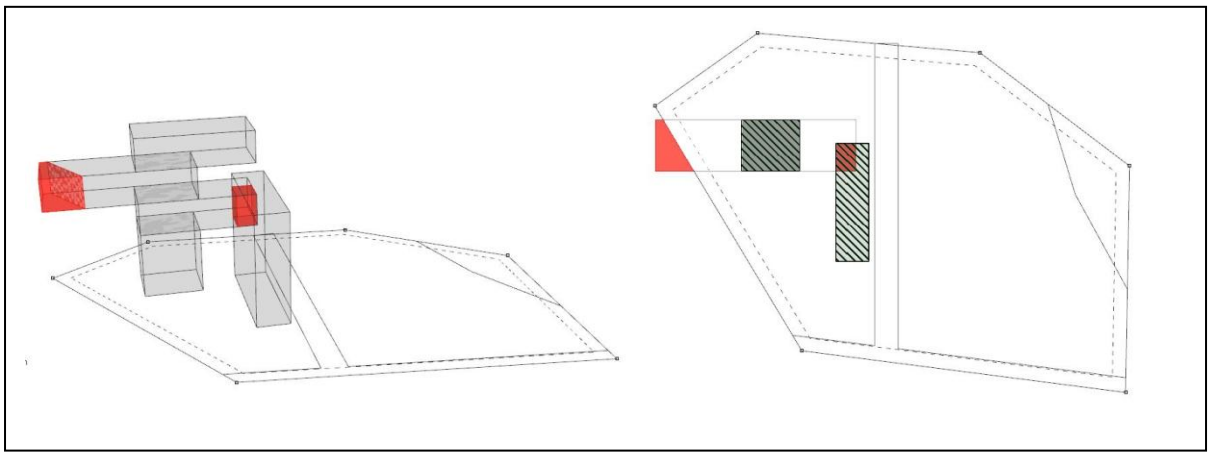


Figure 3.32: Volume overlap and failure in site containment for two buildings that meet the two-dimensional spatial constraints

SC_A = TRUE, when $V_{B1} + V_{B2} + ... + V_{BN} = (V_{B1} \cup V_{B2} \cup ... \cup V_{BN}) \cap V_{CS}$

where N the number of buildings on site
V_{Bi} the volume of building(i), $i \in \{1,2,...,N\}$
V_{CS} is the z extrusion of the site constructible area
$V_{CS} = V_{S'} \setminus (V_{SA} \cup V_{NCZ})$
$V_{S'}$ is the z extrusion of the boundary offset from the site to y,
y is the minimum distance that the buildings have to keep from the site,
V_{SA} is the z extrusion of the site access area,
V_{NCZ} is the z extrusion of the non-constructible zone

(3.19)

SC_B = TRUE, when $V_{B1'} + V_{B2'} + ... + V_{BN'} = V_{B1'} \cup V_{B2'} \cup ... \cup V_{BN'}$

where $V_{Bi'}$ is the volume of building (i)

(3.20)

$NC_A = -\Sigma(V_{Bi} \cap V_{NCS})$

where V_{Bi} the volume of building(i) , $i \in \{1,2,...,N\}$
N the number of buildings on site
V_{NCS} the z extrusion of the site non-constructible area

(3.21)

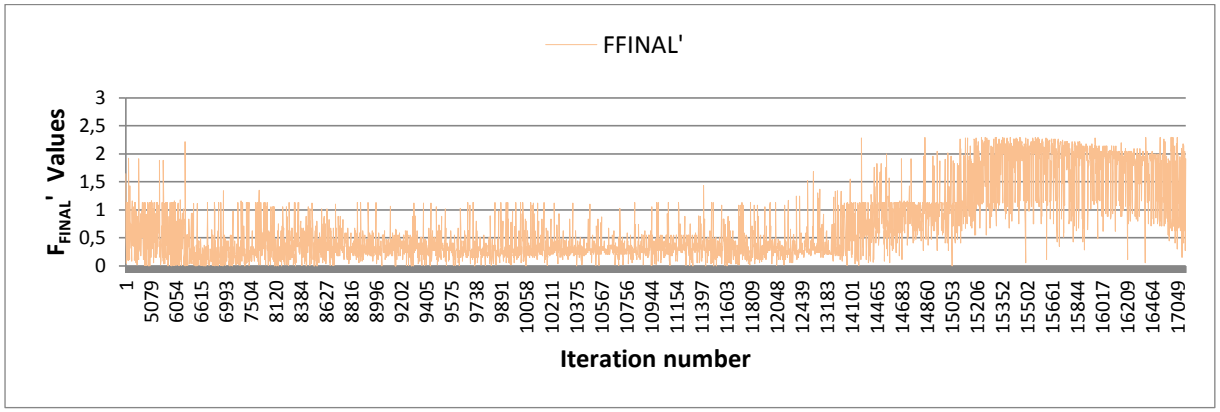
$NC_B = -[(V_{B1'} + V_{B2'} + ... + V_{BN'}) \cap (V_{B1'} \cup V_{B2'} \cup ... \cup V_{BN'})]$

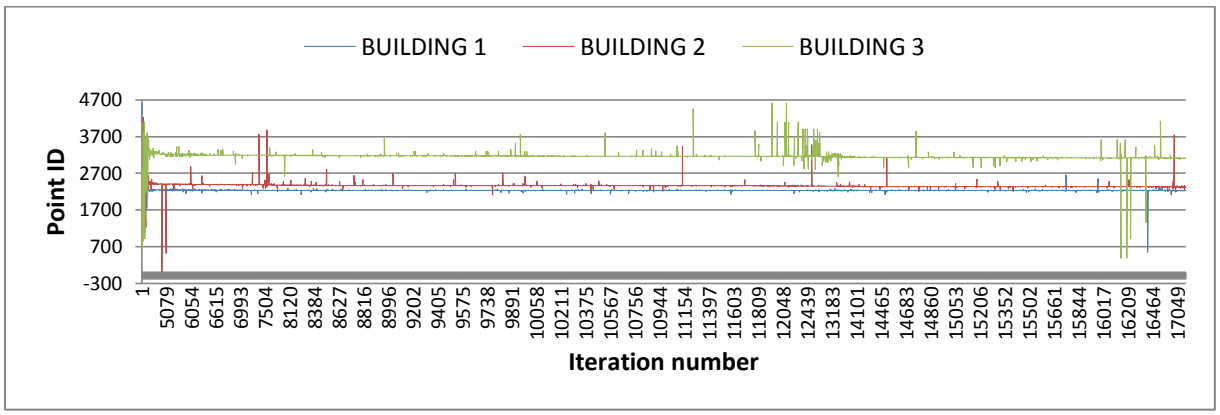
where $V_{Bi'}$ is the volume produced by an offset of the buildings volume to x/2,
$i \in \{1,2,...,N\}$,N the number of buildings on site
x is the minimum distance between two buildings,

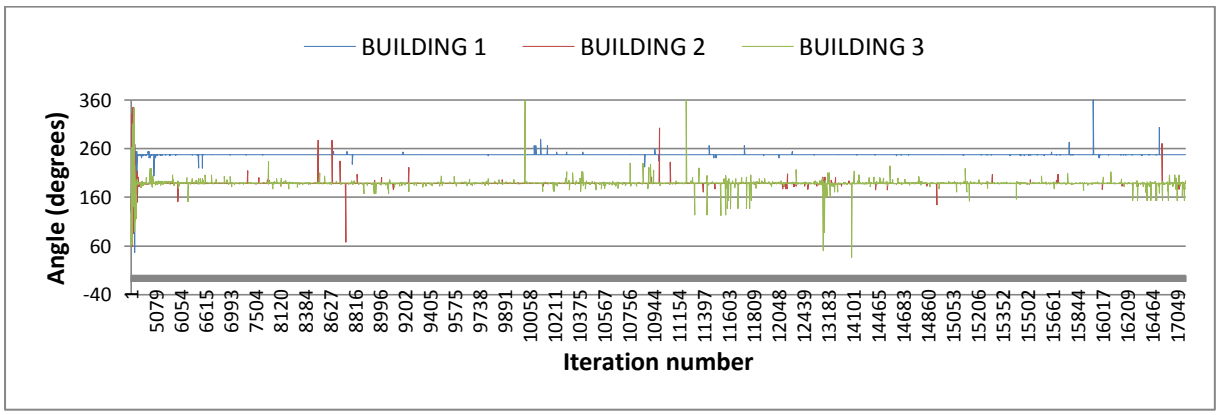
3.3.2 GA VS SA

As seen in the above experiments, the simulated annealing solver works better than the genetic solver in all of the implementation variations. The diagrams below show the evaluation values along with the parameters values that produce each solution. We can see that the genetic solver has a small range of values in the parameters, meaning that doesn't actually experiment a lot in the solution space. It is rather focusing on one solution and tries to optimize it, in contrast with the SA, which finds a new solution in every iteration. We could conclusively say that the GA would be more suitable for a problem where only one solution exists and we aim at getting as closer to it as possible. In our case it is preferred that we find many good solutions, whose slight differences in the evaluation value are not particularly important. It is more important that a wide variety of solutions is produced each one with its pros and cons, so that the user can decide which is more suitable for the specific project.

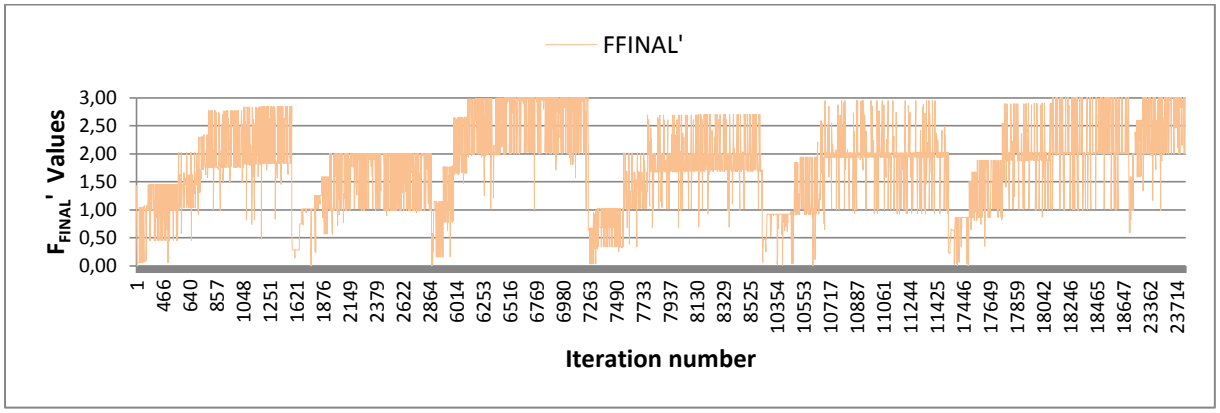
GENETIC SOLVER

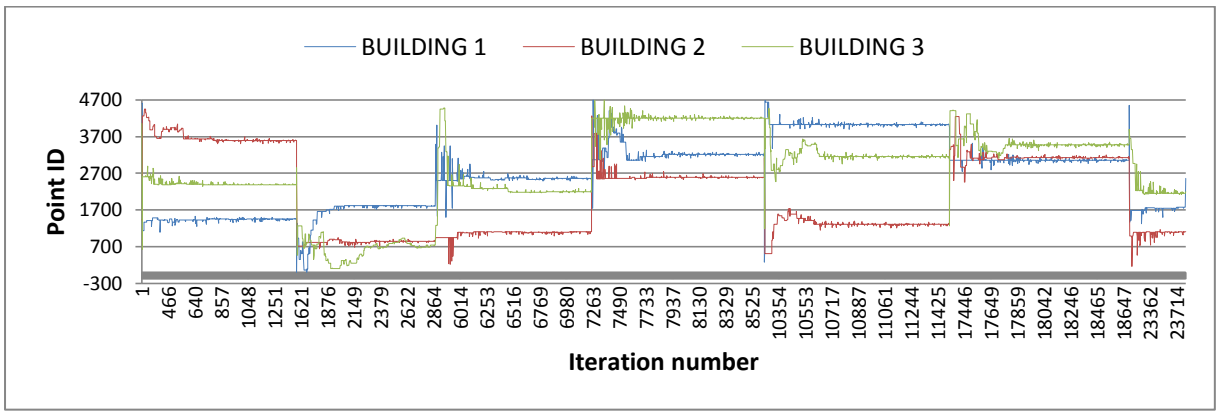


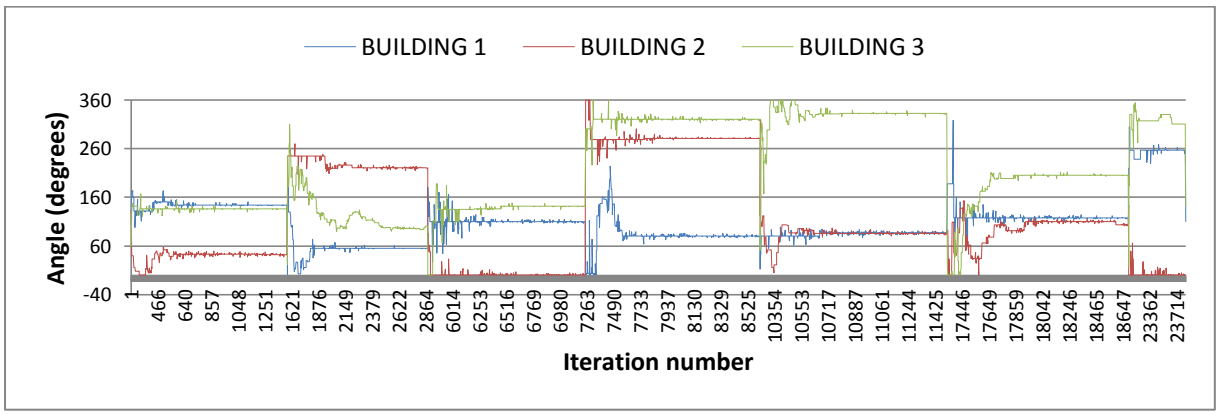




SIMULATED ANNEALING SOLVER







3.3.3 Automated parameters input

According to the way the system is finally implemented, the parameter input to the optimization solver has to be manually done by the user (3.2.2.Optimization.Solution Definition). Manually means that the user will have to copy and paste the sliders for a given number of times and then connect them to the solver component. Since the system designed should be a general application and is not designed to solve one specific problem, it would be more appealing to have in the end only one slider per parameter, one for the points and one for the angle. This could be done if the parameter input for the solver was for each parameter a slider with a value from 0 to 1, which then leads to a selection of n random values from the original lists, where n is the number of buildings. In this way two numbers, both from 0 to 1 produce the solutions and each one of them results in n point ID and n angle values. The user doesn't have to interfere with the procedure since it is automated and works independently of the number of buildings that exist in the specific problem. There is though one major drawback in this choice: the parameter input of the solver, these two 0-1 sliders, produce the buildings' location points and values in an unexpected and random way and this is minimizing the efficiency of the solver, since it produces a problem whose solution landscape is discontinuous (Chart 3.8).

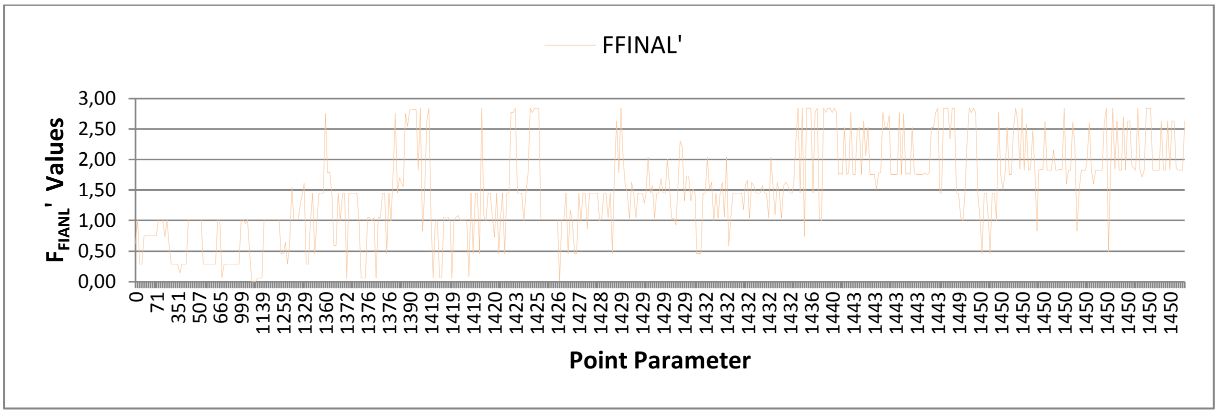


Chart 3.7: Evaluation values for point parameter : discontinuous landscape

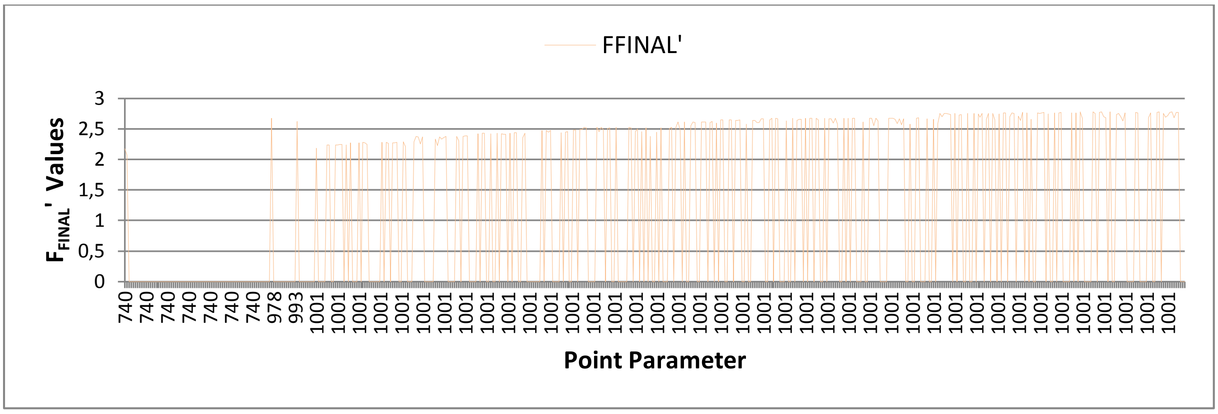


Chart 3.8: Evaluation values for point parameter: continuous landscape

Therefore the decision made was to sacrifice the user's convenience in some degree in order to gain a much higher efficiency of the system.

3.4 Use Scenario

The actual user interface is mainly the 'Grasshopper' interface. The user shouldn't have any previous programming experience, knowledge of the optimization algorithms or of the system structure. Previous experience in 'Grasshopper's" environment would be helpful since the user would already be familiar with the actions required in order to set the input and handle the setting options. It isn't though a requirement, since it is easily manageable from anyone with experience in CAD systems.

The system is graphically divided into 'user' and 'technical' area. The red outline defines the user area which is numbered according to the steps the user should complete in order to perform an optimization run:

1. Design parameters input
2. Optimization settings
3. Optimization
4. Documentation
5. Visualization

Explanation comments describe the procedure that should be followed in every step. The technical part is available for users who are already in a high extend familiar with the software, to facilitate possible experimentation or system improvement. Thus all the options and settings could potentially be manipulated by the user, allowing for a high extend of flexibility, without though resulting in a complex interface, since the absolute necessary steps are isolated and ordered.

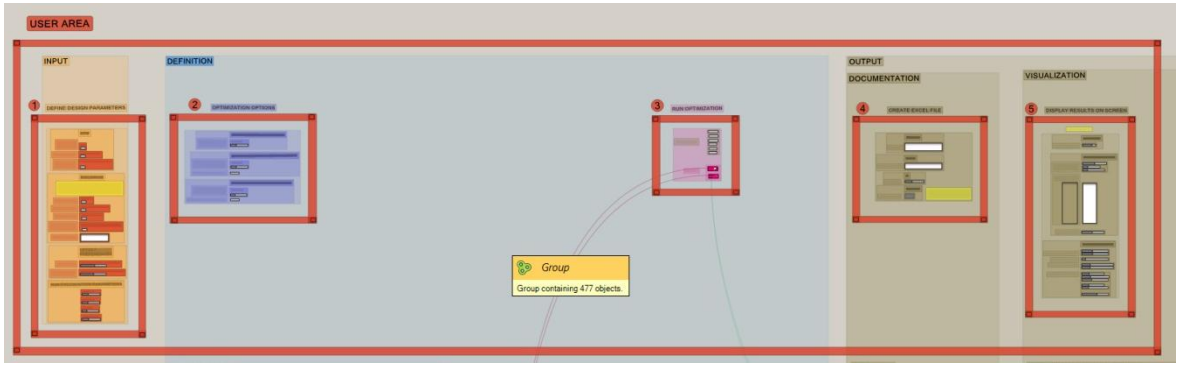


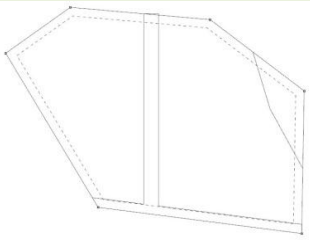
Figure 3.33: 'Site-Planner' Interface

4 Examples

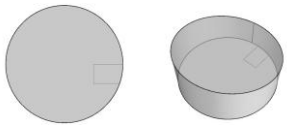
EXAMPLE 1

INPUT:

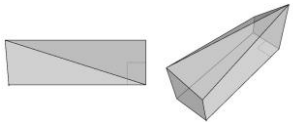
SITE INPUT:



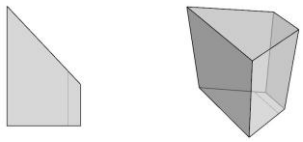
BUILDINGS INPUT:



(Building 1) * 2



(Building 2) * 1



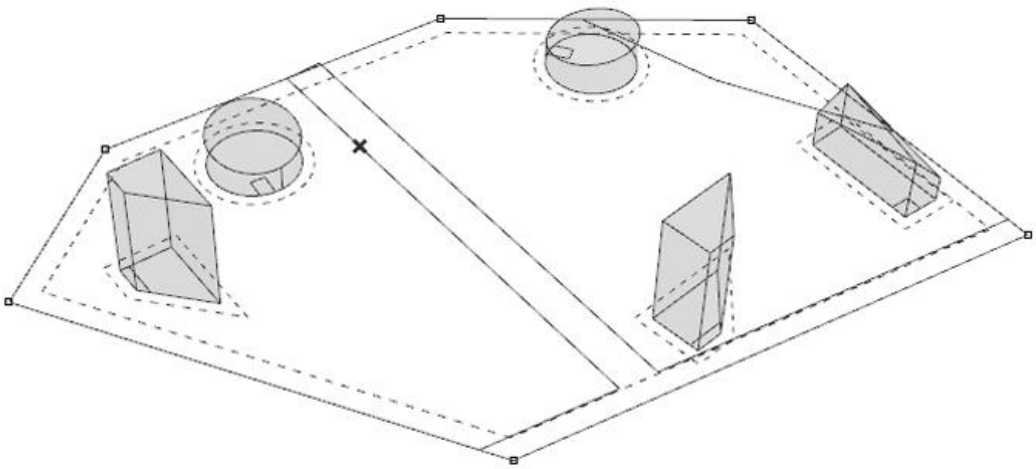
(Building 3) * 2

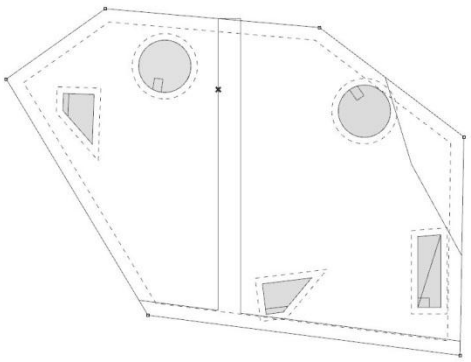
EVALUATION CRITERIA:

W BUILDINGS PROXIMITY = 1 **minimize buildings proximity**
W SOLAR INSOLATION = 0

declination from site access: 70%

OUTPUT:



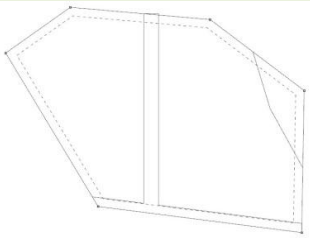


BUILDINGS PROXIMITY = 1,79 (maximum=5)
Calculation time = 17 min

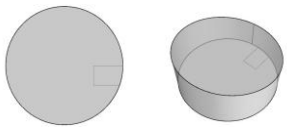
EXAMPLE 2

INPUT:

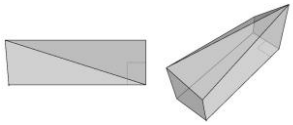
SITE INPUT:



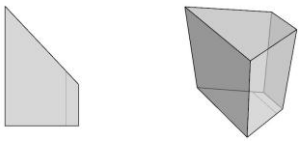
BUILDINGS INPUT:



(Building 1) * 2



(Building 2) * 1



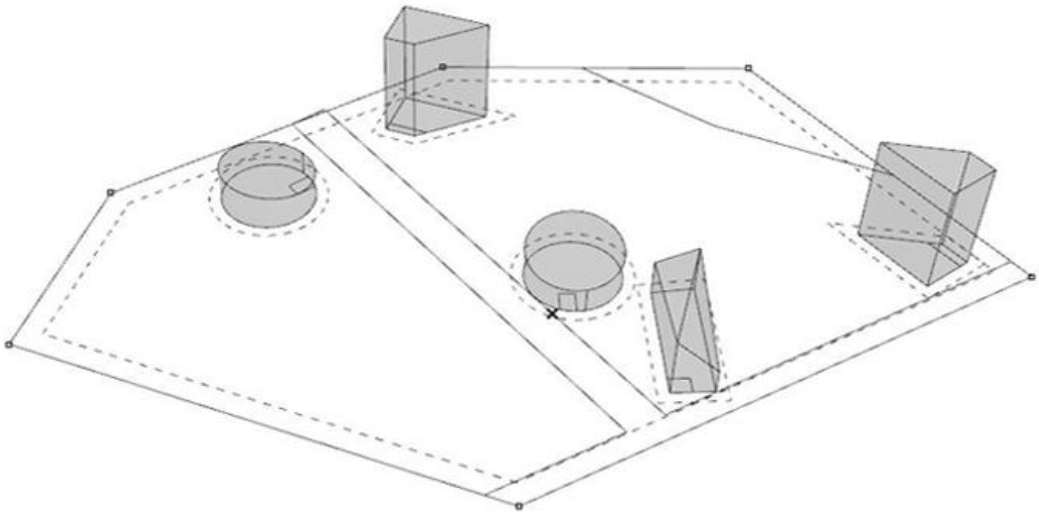
(Building 3) * 2

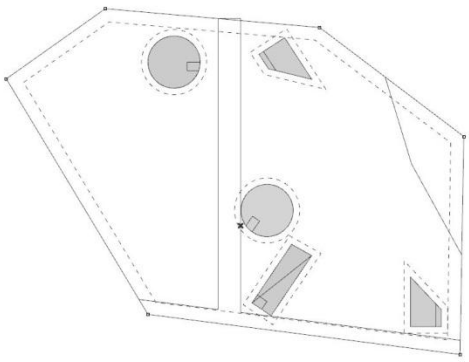
EVALUATION CRITERIA:

W BUILDINGS PROXIMITY = 1 **minimize buildings proximity**
W SOLAR INSOLATION = 0

declination from site access: 0%

OUTPUT:



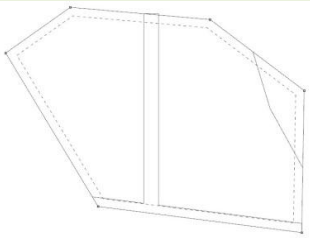


BUILDINGS PROXIMITY = 1,43 (maximum=5)
Calculation time = 42 min

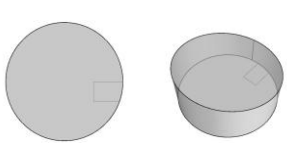
EXAMPLE 3

INPUT:

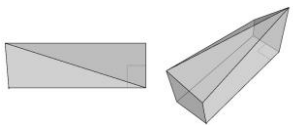
SITE INPUT:



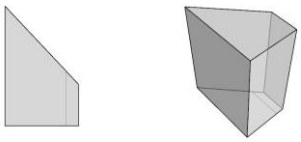
BUILDINGS INPUT:



(Building 1) * 2



(Building 2) * 1



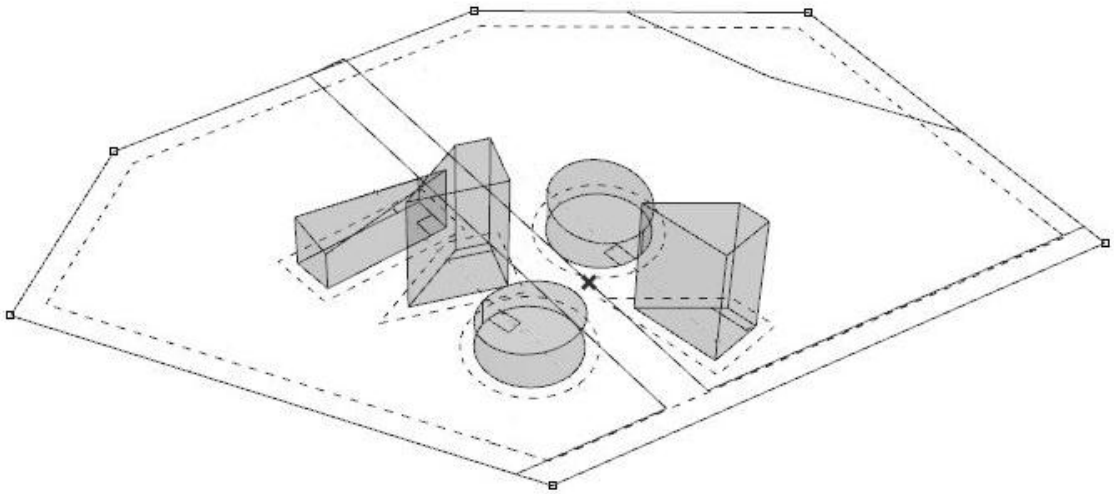
(Building 3) * 2

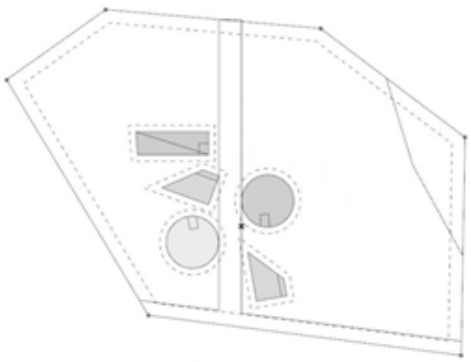
EVALUATION CRITERIA:

W BUILDINGS PROXIMITY = 1 **maximize buildings proximity**
W SOLAR INSOLATION = 0

declination from site access: 0%

OUTPUT:



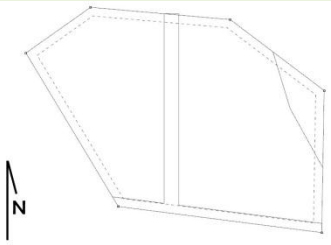


BUILDINGS PROXIMITY = 4,48 (maximum=5)
Calculation time = 36 min

EXAMPLE 4

INPUT:

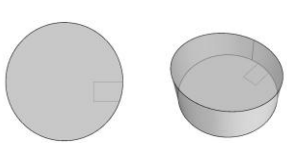
SITE INPUT:



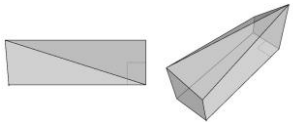
Latitude: 48.21°

Longitude: 16.37°

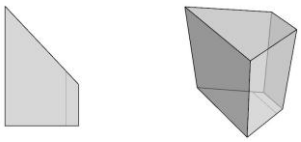
BUILDINGS INPUT:



(Building 1) * 2



(Building 2) * 1



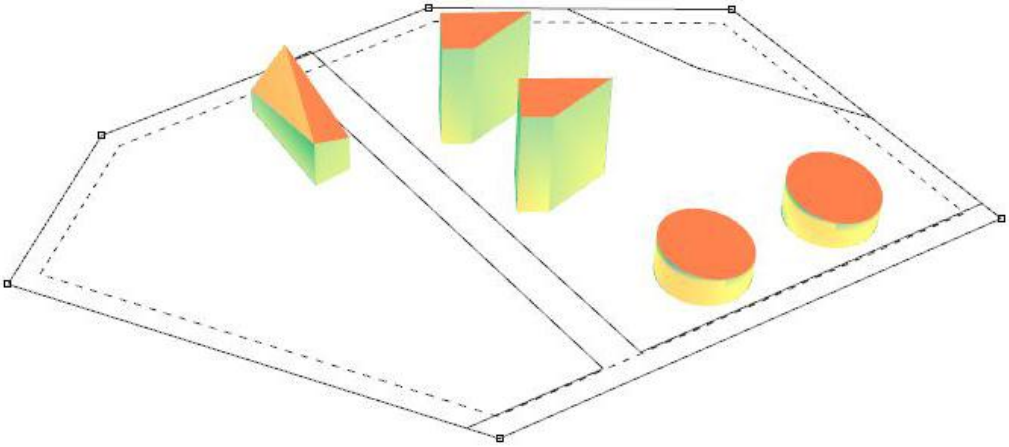
(Building 3) * 2

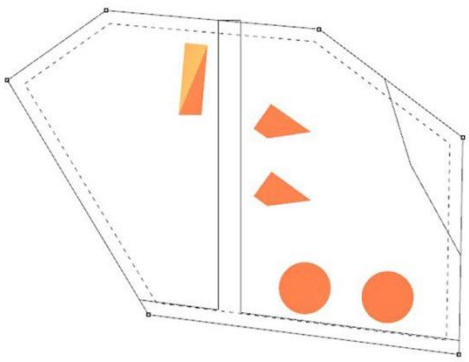
EVALUATION CRITERIA:

W BUILDINGS PROXIMITY = 0
W SOLAR INSOLATION = 1 **maximize solar insolation**

declination from site access: 10%

OUTPUT:



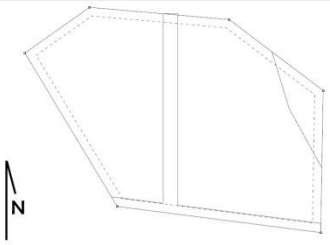


SOLAR INSOLATION = 0,66 (maximum=5)
Calculation time = 6h 21 min

EXAMPLE 5

INPUT:

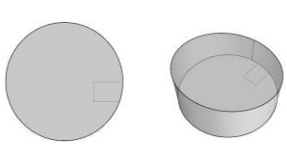
SITE INPUT:



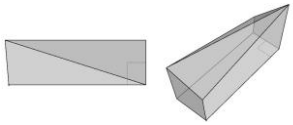
Latitude: 25.03°

Longitude: 121.53°

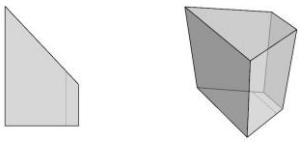
BUILDINGS INPUT:



(Building 1) * 2



(Building 2) * 1



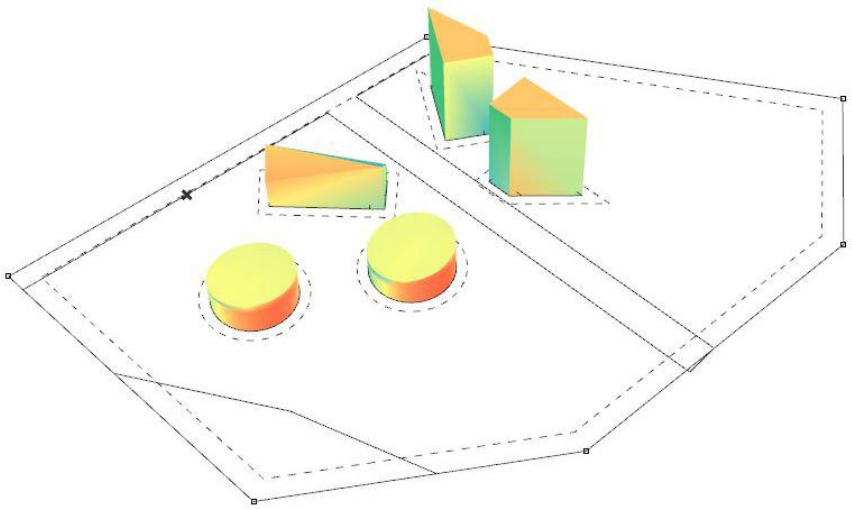
(Building 3) * 2

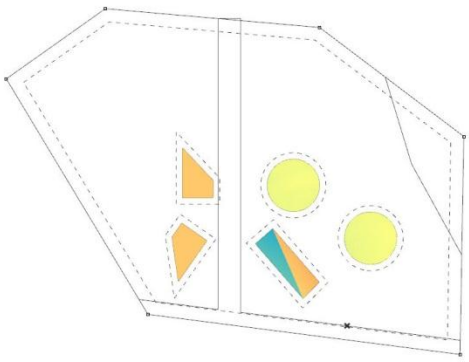
EVALUATION CRITERIA:

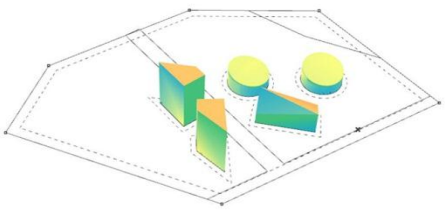
W BUILDINGS PROXIMITY = 1 — **maximize buildings proximity**
W SOLAR INSOLATION = 1 — **maximize solar insolation**

declination from site access: 30%

OUTPUT:







BUILDINGS PROXIMITY = 4,35 (maximum=5)
SOLAR INSOLATION = 1,16 (maximum=5)
Calculation time = 7h 08 min

5 Case studies

In order to verify how useful such a system would be and whether or not it could produce meaningful results we will examine two case studies, that represent existent design, simulated in the above described implementation. The case studies are chosen with main criterion that the designer was driven by the specific constraints we already set as evaluation criterion i.e. solar insolation and distance between buildings. Under such circumstances, we can prove whether or not the above criteria are confronted by the system in a better way than by human design and what the consequences are for other design aspects.

5.1 Taipei City Wall by BIG Architects

Figure 5.1: Taipei City Wall (http://www.big.dk/#projects)

The Taipei City Wall, a 82,000 m2 residential project in the center of Taipei, was assigned to the Danish architectural practice BIG by Taiwan Land Development Corporation.

The architects wanted to achieve the high density of a city center without losing the benefits of the less dense populated suburban areas. Maximization solar insolation and wind penetration, maximization of green roofs and gardens and the creation of a more 'neighborhood-like' atmosphere were the main goals of the design.

Starting point for the design process was a set of 5-floor cubical building modules of 15m long edges, with equivalent in floor plan exterior space. As a second step the modules were placed on top of each other in order

to increase the density. The overlap of the buildings is minimized to a small surface large enough to host the vertical connections, in order to achieve the desired garden areas.

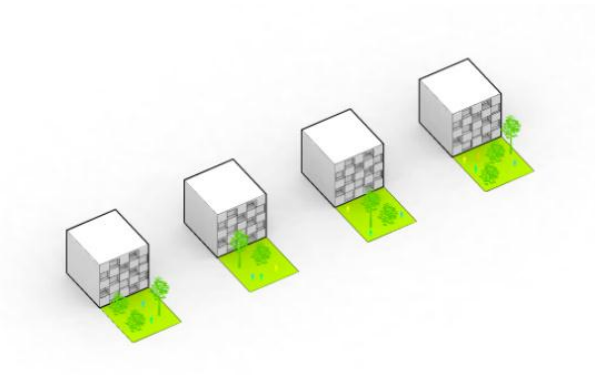


Figure 5.2: Building Module
(http://www.big.dk/#projects)

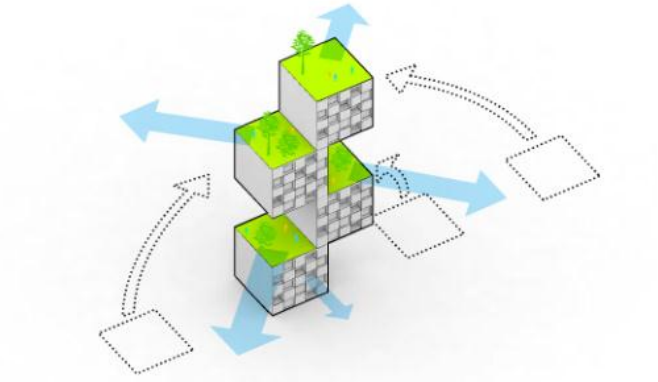


Figure 5.3: Vertical placement
(http://www.big.dk/#projects)

This unit was then multiplied across the central axis of the site resulting in a 72-cube "wall", leaving most of the site's area free to create public green space.

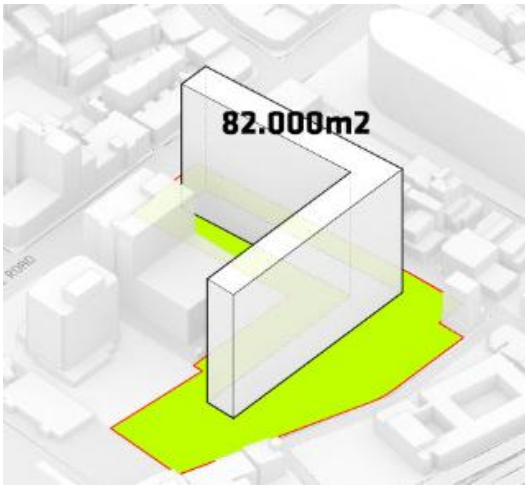


Figure 5.4: Maximal height and central placement in order to gain public green space
(http://www.big.dk/#projects)

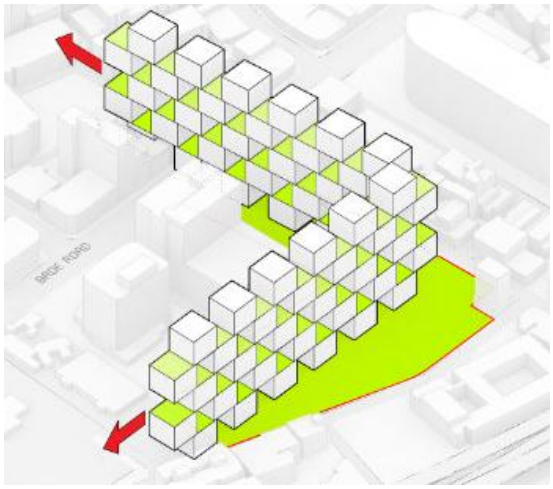


Figure 5.5: Stretching the volume to promote daylight and add shared recreational gardens
(http://www.big.dk/#projects)

By ‘pressing’ the cubes in order to be contained in the site, the wall becomes curved and the gardens between them become trapezoid in floor plan instead of rectangular, therefore having an opening to the sun which is on the one side bigger than on the other.

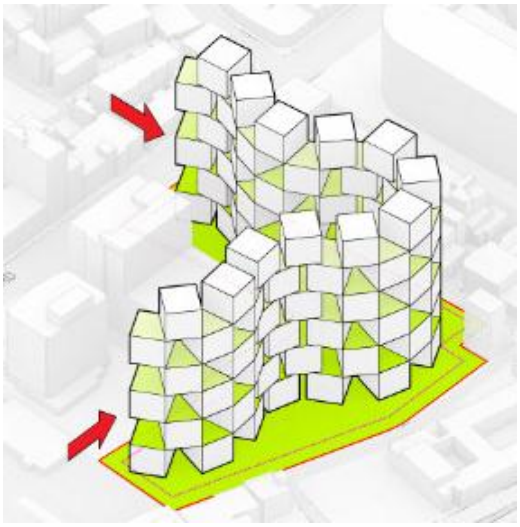


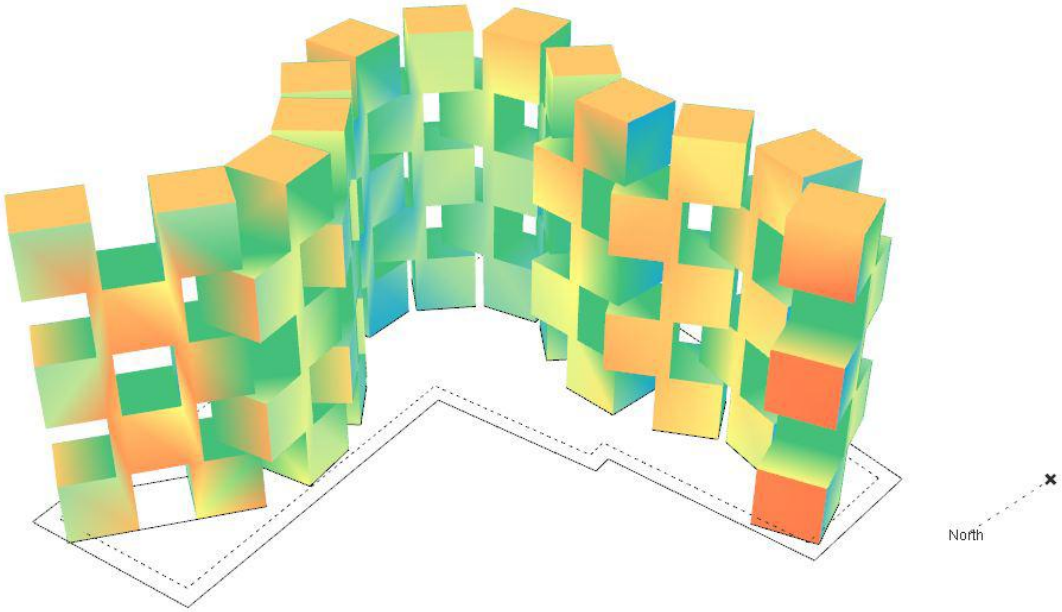
Figure 5.7: Pushing the building back into the site creates overlapping which hold the cores
(http://www.big.dk/#projects)

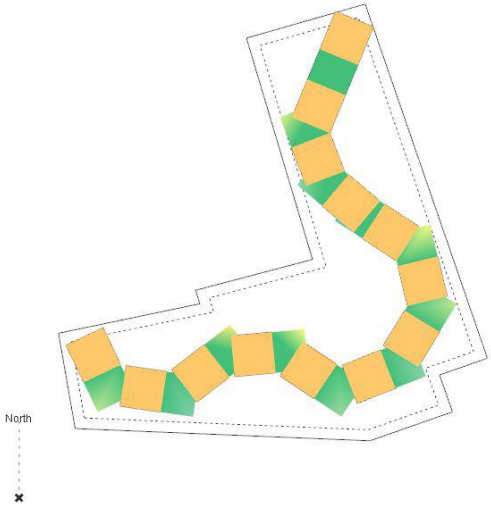
Figure 5.6: The volume is lowered at one side in order to break down the scale of the building
(http://www.big.dk/#projects)

At the end 6 cubes are removed in an attempt to balance the relationship of the building with the neighboring plots, where lower building and open heights are more dominant.

The result is as dense as desired by the developers while simultaneously doubles the free space available.

The desire to maximize solar insolation in this project makes it an ideal case study for testing the implemented system. The building simulation will be done based on the initial concept of the vertical unit repetition, without the final adjustments of extracting some units and lowering one side. This is a decision that can be made after the basic design is produced. The solar insolation criterion can be evaluated once in the given design and once in the system outcome and the results will be compared also taking in mind the other criteria of the design. The maximization of the buildings proximity is also object of optimization since, the intention was to gather the buildings to the center of the site. Both weights are set to 1.





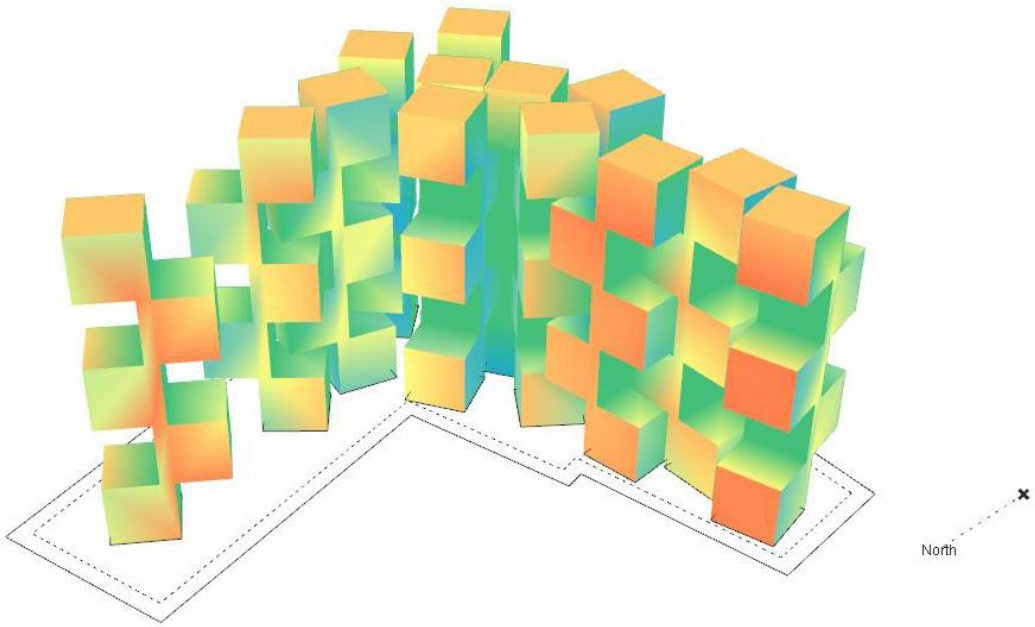
CURRENT BUILDING

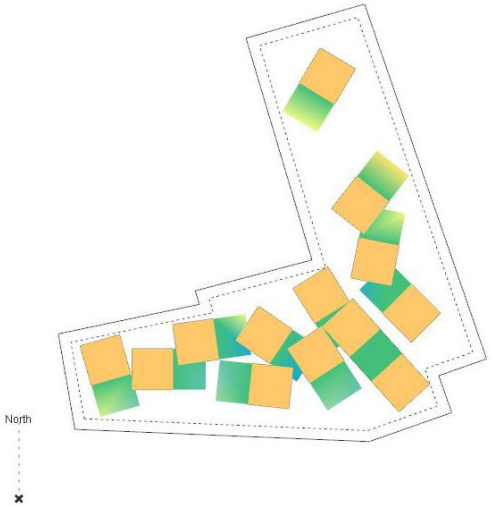
EVALUATION VALUES:

SOLAR INSOLATION: 1.44 (12 max.)
BUILDING PROXIMITY: 8.45 (12max.)

ABSOLUTE VALUES:

SOLAR INSOLATION: 552.38
BUILDING PROXIMITY: 1488.10 m





SOLUTION 1

EVALUATION VALUES:

SOLAR INSOLATION: 1.19 < 1.44 (12 max.)
BUILDING PROXIMITY: 8.63 > 8.45 (12 max.)

ABSOLUTE VALUES:

SOLAR INSOLATION: 456 < 552.38
BUILDING PROXIMITY: 1321 < 1488.10 m

Calculation time: 72,5 h

The system didn't achieve a better insolation value than the one that the original layout of the buildings displayed. Moreover the result displays major 'disorder' in the way the buildings are placed next to each other and therefore it is more likely that it wouldn't be approved by the designers.
This mainly happened because of the extremely large calculation times needed. The high density in the site means that the system needs to search for a long time in the solution space until a valid solution, where the spatial constraints are met, is found. Therefore it produces very few valid solutions in a long time period. During the 72,5h run, the system was able to perform three SA runs which means that the results depict three solutions categories. Each category includes many valid solutions which mainly express one basic design with only small alterations that sometimes are not even noticeable.

5.2 Greater Noida Housing by FXFOWEL

The Greater Noida Housing Project is a 400,000 m² residential development, in the planned city of Greater Noida situated 30 miles southwest of New Delhi. It is designed by FXFOWEL architects, who considered the environmental and social sustainability as main criterion of this project.

Figure 5.8: Greater Noida Housing Complex (http://www.fxfowle.com/)

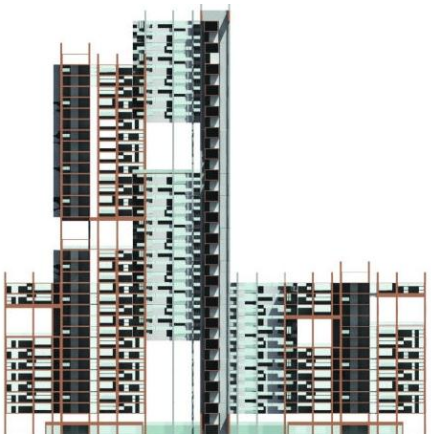


Figure 5.9: Buildings Elevation (http://www.fxfowle.com/)

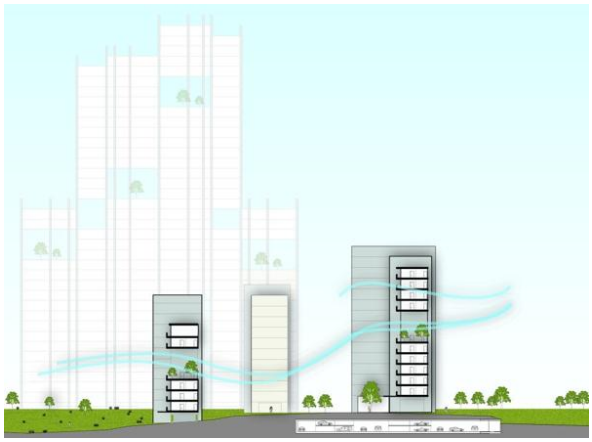


Figure 5.10: Building Section (http://www.fxfowle.com/)

In order to reach environmental sustainability, the architects placed the units above each other while shifting floors and creating voids and open outdoor spaces, thus not only adding a textural vertical living environment and lightening the scale of the project, but also letting air and sunlight penetrate the complex.

The complex consists of 22 buildings which are formulated by modules stacked between terracotta colored sheet walls that also act as sun blocks. The majority of the planned 1700 residential units have a north-south orientation to maximize solar exposure in the winter months. A number of floors are left open to allow cooling summer monsoon breezes maintain comfortable living conditions within the elevated living spaces. The voids between floors also contain balconies and public spaces encouraging interaction between the residents.
Larger buildings stand up to forty-five stories and block winds on the north side of the 47 acre site. A cluster of smaller buildings to the south let winter light penetrate the green belt between the buildings, creating an overall effect of a small scale city.

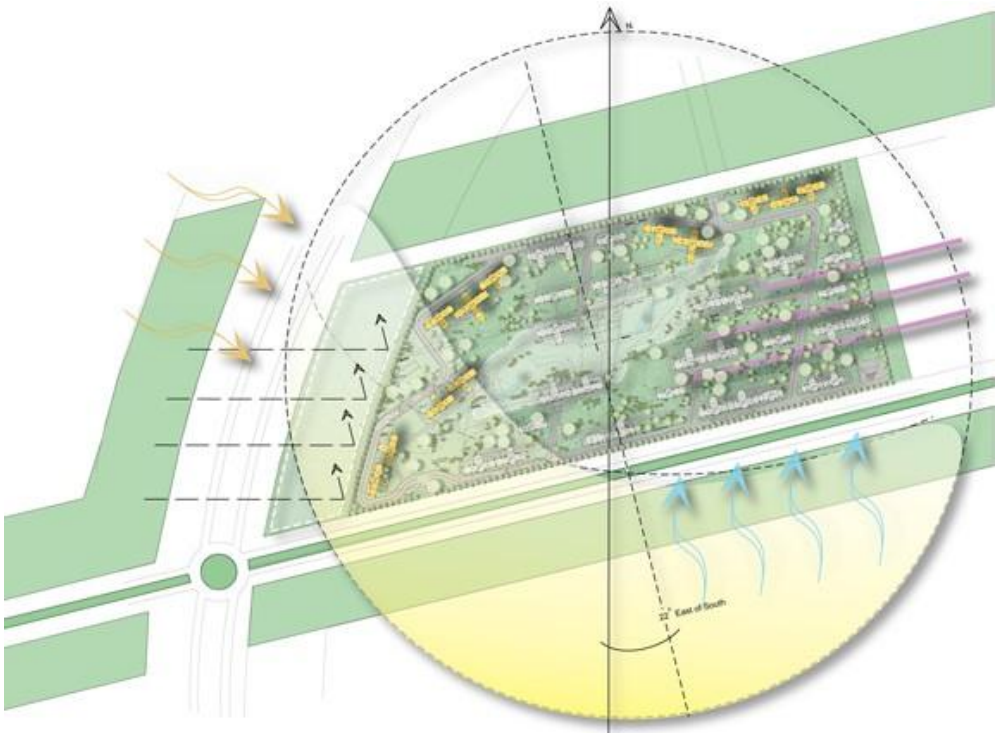
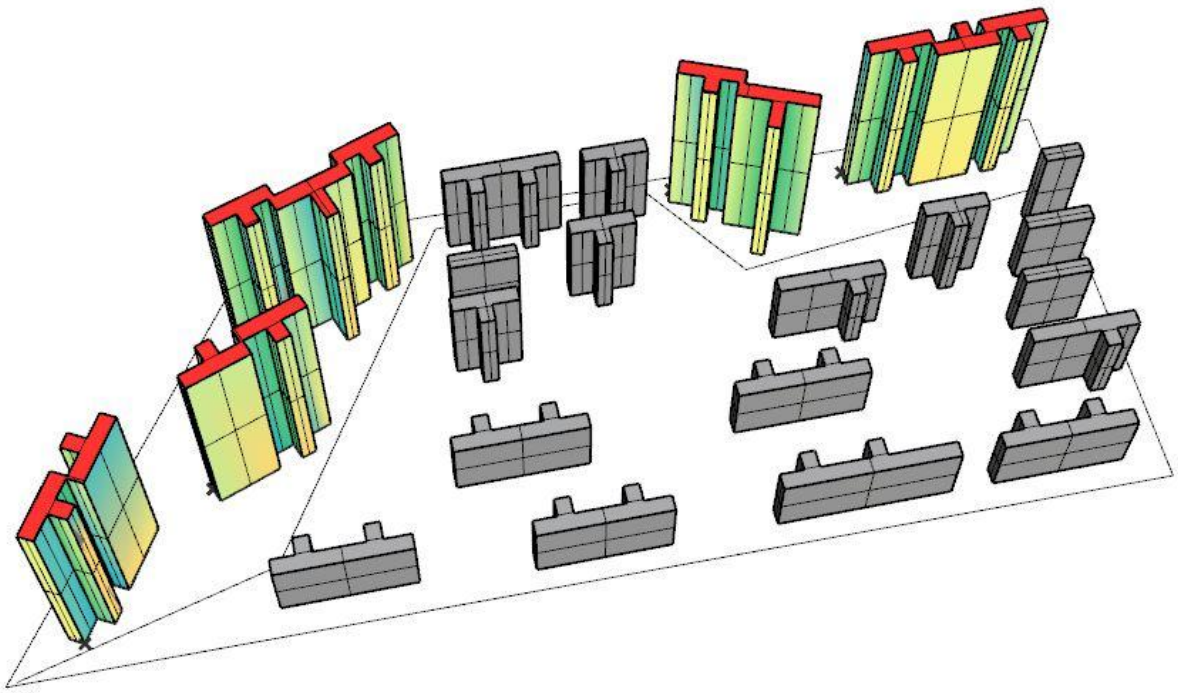
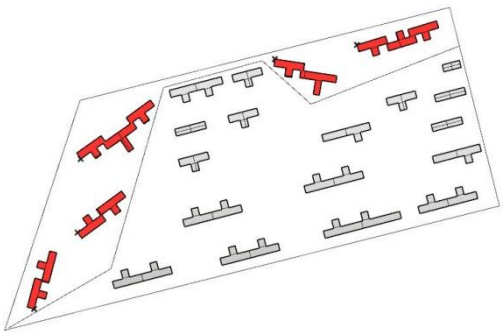


Figure 5.11: Site Plan (http://www.fxfowle.com/)

For the needs of the simulation it was considered an appropriate solution to divide the site in two parts, since the strategy was to place all the high buildings to the north and the lower ones to the south. Therefore the site is divided according to the given design in two parts: The northern one containing 5 higher buildings and the southern one containing the 17 rest. The intention is to maximize solar insolation and minimize buildings proximity, since it is important that wind and light penetrates the site. Both criteria weights are set to 1.





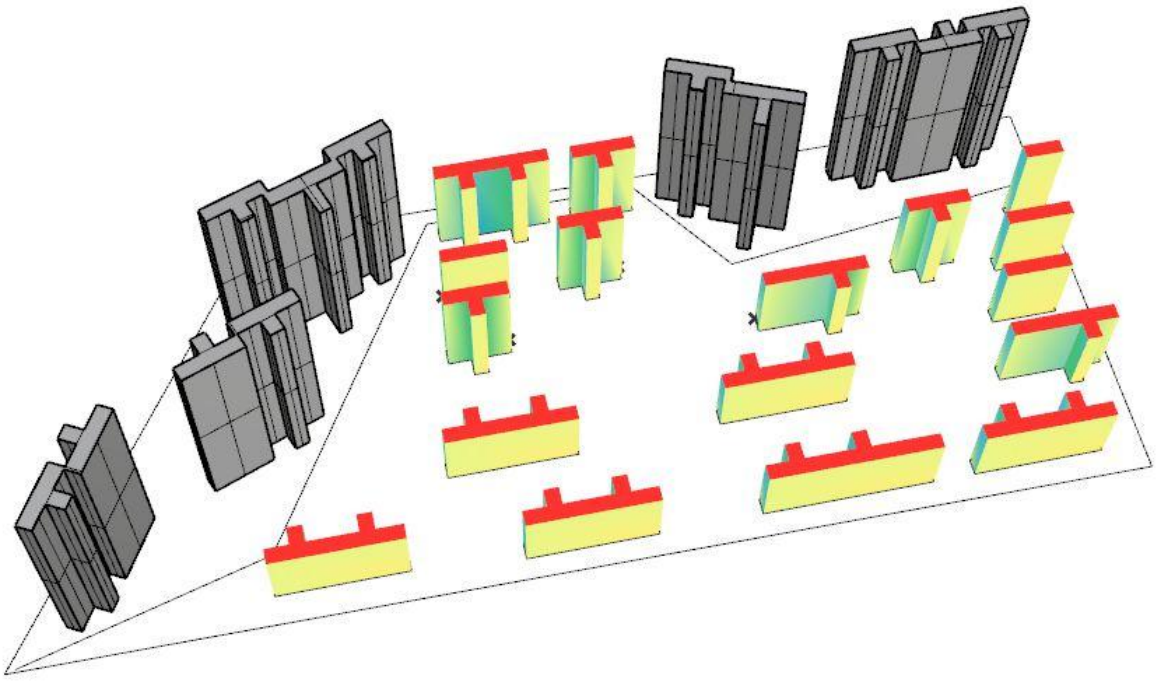
CURRENT BUILDING – Part A

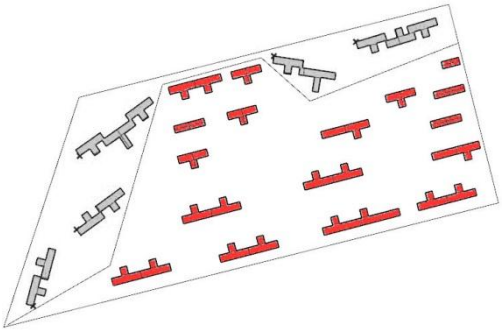
EVALUATION VALUES:

SOLAR INSOLATION: 0.96 (max. 5)
BUILDING PROXIMITY: 1.58 (max. 5)

ABSOLUTE VALUES:

SOLAR INSOLATION: 347
BUILDING PROXIMITY: 1140.00 m





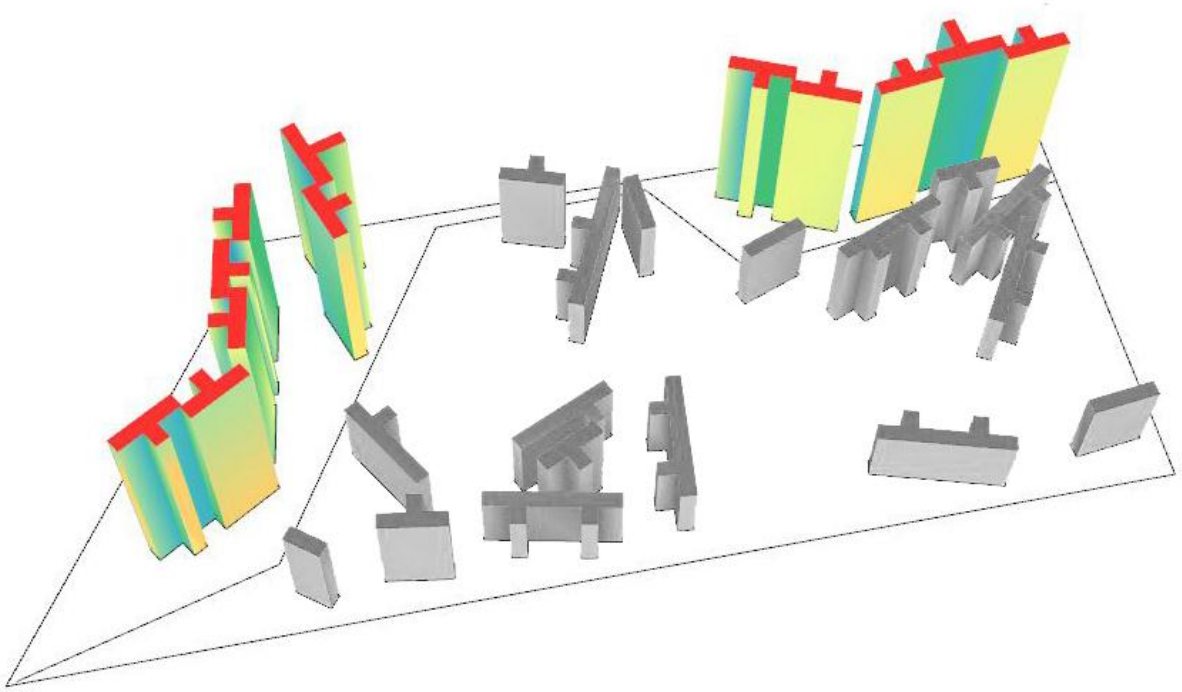
CURRENT BUILDING – Part B

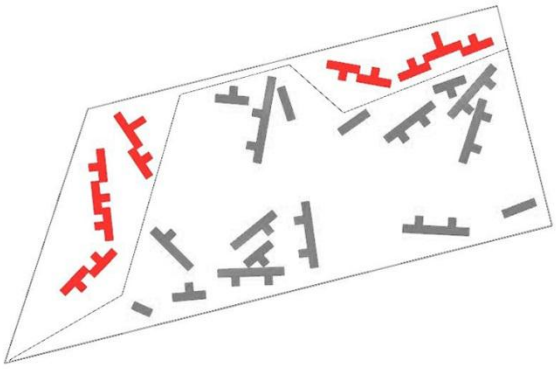
EVALUATION VALUES:

SOLAR INSOLATION: 2.61 (max. 17)
BUILDING PROXIMITY: 5.17 (max. 17)

ABSOLUTE VALUES:

SOLAR INSOLATION: 948
BUILDING PROXIMITY: 3418.00 m





SOLUTION 1 – Part A

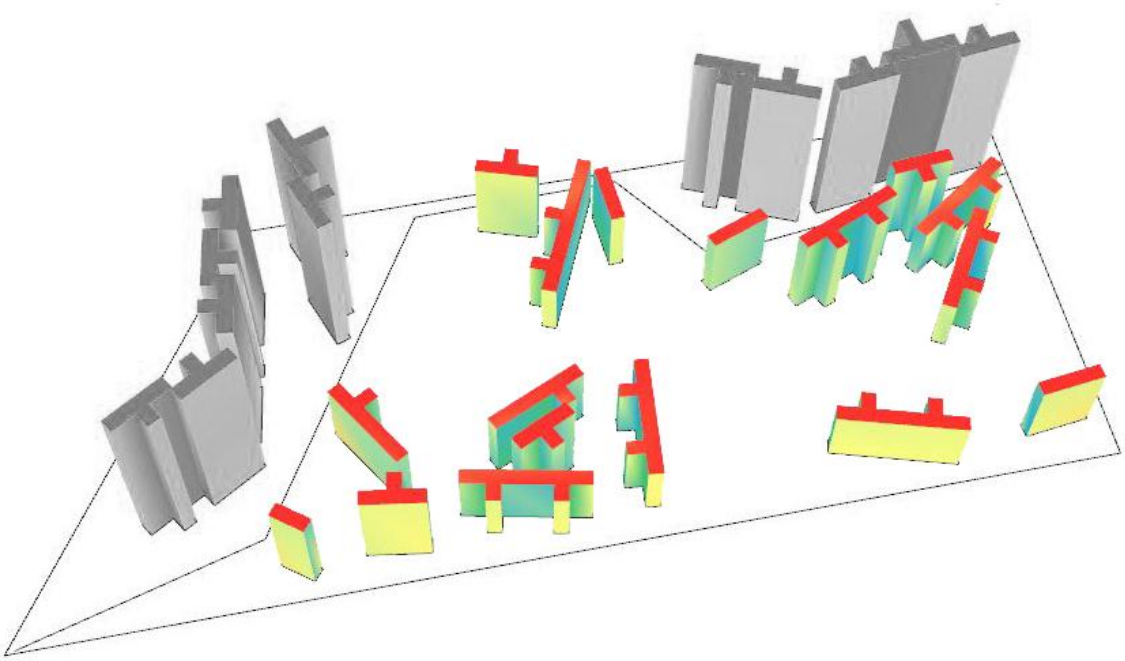
EVALUATION VALUES:

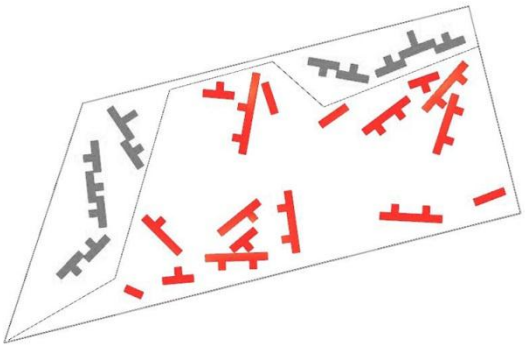
SOLAR INSOLATION: 1.10 > 0.96 (max. 5)
BUILDING PROXIMITY: 1.46 < 1.58 (max. 5)

ABSOLUTE VALUES:

SOLAR INSOLATION: 400 > 347
BUILDING PROXIMITY: 1052.31<1140.00 m

Calculation time: 2,8 h





CURRENT BUILDING – Part B

EVALUATION VALUES:

SOLAR INSOLATION: 2.77 > 2.61 (17 max.)
BUILDING PROXIMITY: 5.12 < 5.17 (17 max.)

ABSOLUTE VALUES:

SOLAR INSOLATION: 1003.82>948
BUILDING PROXIMITY: 3389.85<3418.00 m

Calculation time: 8,5 h

The interpretation of the final arrangement of the buildings is controversial, basically because very small distances between the buildings where sometimes kept and it can be considered that this contradicts the initial wish of the designers to keep large distances and let sun and wind penetrate the building. Another issue would be here as well the amount of order in the final layout, which is rather a matter of aesthetics and personal preferences.

The solar insolation of the building complex was indeed improved by 7,45%.

6 Conclusion

6.1 Capacities/limitations

User Input

The user input regarding the buildings is not quite straight forward. The system is designed in such a way, that the user won't have to manually insert the appropriate number of components according to the number of buildings on the site. In order to accomplish that and facing the fact that the system is not actually implemented in a programming language but in a graphical algorithm editor, that treats data structures with certain limitations, the input process is designed in such a way that the user has to be extremely careful when selecting the input geometry: all the buildings have to be selected the same order for each parameter. That means that the user should select one polyline defining the buildings floor plan boundary for Building 1, Building 2, …. and then maintain this order of buildings while selecting the rest of the parameters (building access boundary, building volume). All the parameters have to be one single object. If one building has for example two entrances, they still have to be entered as one polyline. If one parameter is omitted for one building, then the output will be incorrect: if one building does not have a defined entrance area the user should enter the whole area in the building access input, in order to maintain a proper data structure in the system.

Solar Insolation

The solar insolation criterion is evaluated through an abstracted method, of calculating the sun rays that hit the buildings envelope. A more accurate calculation could be done through a software for solar analysis 'Ecotect', since there is already a connection between these platforms that allows interactive feedback by exporting the grasshopper geometry into the ecotect environment and importing the results for visual feedback. Such a solution would be though unfeasible, since the calculation time is at least 10 times bigger than is currently needed and for a big number of buildings or for complex geometry, that would strongly reduce time efficiency.

Offsets

The spatial constraints are built on operators that produce curve and polysurface offsets. Therefore they are subject to malfunctions that often occur in design software depending on the geometry input. Such malfunctions may result in falsification of the results and the solution evaluation.

Calculation Time

The calculation time grows along with the complexity of the design simulated. For a set of five buildings of relatively simple geometry and both criteria weights bigger than zero, the system needs approximately 3-4 hours to produce one solution. Reaching the number of 10 buildings the calculation time can last 24h. The long calculation times are mainly a result of the time needed to calculate the solar insolation. Even though the system doesn't perform a regular solar analysis but just a calculation based on a simplification method, its efficiency regarding the time parameter is still low. Considering the fact that the system is designed in order to provide solutions for a complex design input, this is a major drawback.
Respectively, when displaying on screen results of solutions where solar insolation is calculated, the response time while manipulating the display settings is also large.

6.2 Proposals for future research

Partial Solar Insolation Evaluation

It is seen in many cases that the placement of a building is done in a way that tries to optimize sun exposure only in parts of the buildings, either because of the intended usage (different uses in south/north façade) or because of the intention to use specific facades or the roof for solar panels. Thus it would be a very useful feature if the user could only select parts of the building for the solar insolation evaluation.

Buildings' Distribution on the Site

We noticed through several experiments and examples that the system tends to 'group' the buildings while trying to minimize their proximity. Since this is not always the desired result, it would be a useful feature if the type of distribution of the buildings on the site could be controlled by the user.

Layout Rules

As seen by the results, the system output almost always diverges from conventional site planning design because of the lack of order in the layout. Independently of whether order should be a desired feature or not, the system could be adjusted to fit such design criteria. A simple method to achieve that, would be to implement a settings option restricting the difference of the value angles of the buildings to n*90 degrees with $n \in (0,1,2,3)$. This would of course have a negative impact on the solar insolation optimization, since the solution space would be considerably reduced.

Neighboring Buildings Input

The neighboring buildings affect the shading efficiency of the buildings on site, and should be in the future incorporated into the system implementation.

Site Input

Another useful feature that affects both accessibility and solar insolation is the sites morphology; the current implementation considers the site as a two dimensional object. By introducing a z-value, a reconsideration of the spatial constraints regarding accessibility would be necessary.

Design Criteria

In general, more evaluation criteria could be implemented so that the system could be more efficient in terms of meeting the needs of a designer. One of these criteria could be the ability to obtain certain views from certain locations on the site or the privacy/publicity of space.

References

Bibliography

Aarts, Emile, H.,L., Jan,H.M. Korst, and Peter,J.M. van Laarhoven. "Simulated Annealing." In *Local Search in Combinatorial Optimization*, by Emile, H.,L. Aarts, & J.K. Lenstra, 91-120. New Jersey: Princeton University Press, 2003.

Aho, A. V., J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms.* Boston: Addison-Wesley, 1974.

Akin, O., B. Dave, and S. Pithavadian. "Heuristic generation of layouts (HeGeL): based on a paradigm for problem structuring." *Environ. Plan. B:Plan. Des.* 19 (1992): 33-59.

Baeck, Thomas. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies , Evolutionary Programming, Genetic Algorithms.* New York: Oxford University Press, 1996.

Baykan, C. *Formulating spatial layout as a disjunctive constraint satisfaction problem.* Ph.D. diss. Department of Architecture, Carnegie Mellon University, 1991.

Ciftcioglu, O., and M.S. Bitterman. "Solution Diversity in Multi-Objective Optimization: A study in Virtual Reality." *IEEE Word Congress on Computational Intelligence.* Hong Kong, 2008.

Dorigo, Marco, and Thomas Stuetzle. *Ant Colony Optimization: Overview and Recent Advances.* Brussels: IRIDIA-Technical Report Series, Université Libre de Bruxelles, 2009.

Eastman, Charles. "Automated space planning." *Artificial Intelligence* (Elsevier) 4, no. 1 (1973): 41-64.

Flemming, U., C. Baykan, and R. Coyne. "Hierarchical generate-and-test versus constraint-directed search." Edited by J. Gero. *Proceedings of the Artificial Intelligence un Design Conference '92.* Dordrecht: Kluwer, 1992. 817-838.

Flemming, U., R. Coyne, S. Fenves, J. Garrett, and R. Woodbury. "SEED-software environment to supprot the early phases in building design." *Proc. IKM '94.* Weimar, 1994. 5-10.

Glover, F, and M. Laguna. *Tabu Search.* Boston: Kluwer, 1986.

Glover, F. "Future paths for integer programming and links to artficial intelligence." *Computers and Operations Research* 13 (1986): 533-549.

Glover, F., and G. A. Kochenberger. *Handbook of metaheuristics.* Springer, 2003.

Grason, J. "An approach to computerized space planning using graph theory." *Proceedings of the Design Automation Workshop.* New York: IEEE, 1971. 170-179.

Holland, J. *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press, 1975.

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing." *Science* 220, no. 4598 (1983): 671-680.

Koziel, Slawomir, and Xin,She Yang. *Computational Optimization, Methods and Algorithms .* Berlin: Springer, 2011.

Ligget, Robin S. "Automated facilities layout:past, present and future." *Automation in Construction* (Elsevier) 9 (2000): 197-215.

Lobos, D., and D. Donath. "The problem of space layout in architecture: A survey and reflections." *arquiteturarevista* 6 (2010): 136-161.

Mitchell, Melanie. *An Introduction to Genetic Algorithms.* Massachusetts: MIT Press, 1996.

Muther, R. *Systematic Layout Planning.* Boston: Cahners Books, 1973.

Pfefferkorn, C. *The design problem solver: a system for designing equipment or furniture layouts.* Pittsburgh: Department of Computer Science, Carnegie-Mellon University, 1972.

Poli, Richardo, William,B. Langdon, and Nicholas,F. McPhee. *A Field Guide to Genetic Programming.* Riccardo Poli, William B. Langdon, Nicholas Freitag McPhee, 2008.

Sharpe, R., and B. Marksjo. "Solutions of the facilities layout problem by simulated annealing." *Comp. Environ. Urban Syst.* 11, no. 4 (1986): 147-154.

Yang, Xin,She. *Nature-Inspired Metaheuristic Algorithms.* Frome: Luniver Press, 2010.

Web Sources

http://alexandria.tue.nl/repository/freearticles/496713.pdf
http://artint.info/html/ArtInt.html
http://housing03.blogspot.co.at/
http://ieatbugsforbreakfast.wordpress.com/2011/03/04/epatps01/
http://slingshot-dev.wikidot.com/ghdb-genome
http://theprovingground.wikidot.com/usc-spring2011-resources
http://www.grasshopper3d.com/forum
http://www.srrb.noaa.gov/highlights/sunrise/azel.html
http://www.tedngai.net/

Appendix A. System Operators

Spatial Constraints

A: CONTAINMENT IN CONSTRUCTIBLE SITE
SC_A = TRUE, when $V_{B1} + V_{B2} + ... + V_{BN} = (V_{B1} \cup V_{B2} \cup ... \cup V_{BN}) \cap V_{CS}$ where N the number of buildings on site V_{Bi} the volume of building(i), i$\in${1,2,…,N} V_{CS} is the z extrusion of the site constructible area $V_{CS} = V_{S'} \setminus (V_{SA} \cup V_{NCZ})$ $V_{S'}$ is the z extrusion of the boundary offset from the site to y, y is the minimum distance that the buildings have to keep from the site, V_{SA} is the z extrusion of the site access area, V_{NCZ} is the z extrusion of the non-constructible zone

B: BUILDINGS OVERLAP
SC_B = TRUE, when $V_{B1'} + V_{B2'} + ... + V_{BN'} = V_{B1'} \cup V_{B2'} \cup ... \cup V_{BN'}$ where $V_{Bi'}$ is the volume of building (i)

C: ACCESSIBILITY REQUIREMENTS
SC_C = TRUE, when $d < f(w) * L$ $f(w)=((D-L)*w)/100+L$ where d the distance between the building's access area and the site access area L the longest edge of the building access area bounding polygon D the longest one of the distances between the points of the constructible site and the site access area $0 \le w \le 100$ the weight controlling the constraint's relaxation

Evaluation Criteria

A: BUILDINGS PROXIMITY
$FA_i =$ $1-[((DB_1B_i + DB_2B_i + ... + DB_NB_i)/(N-1)) / DP_oP_M]$, when maximizing proximity $((DB_1B_i + DB_2B_i + ... + DB_NB_i)/(N-1)) / DP_oP_M$, when minimizing proximity Where DB_iB_j the distance between building (i) and building (j), N the number of buildings and i, j $\in$ (1, 2, …, N) and Po, Pm the two more distant points on the site
$\sum FA = FA_1 + FA_2 + ... + FA_N$, for N number of buildings

B: SOLAR INSOLATION

$$FB_i = \begin{cases} \Sigma\sin(a)/N_R, \text{ when maximizing solar insolation} \\ 1 - \Sigma\sin(a)/N_R, \text{ when minimizing solar insolation} \end{cases}$$

where a the angle between the sun ray vector and the building and N_R the total vectors of the sun analemma produced

$\sum FB = FB_1 + FB_2 + \ldots\ldots + FB_N$, for N number of buildings

Negative Evaluation Criteria

A: CONSTRUCTIBLE SITE CONTAINMENT

$NC_A = - \Sigma(V_{Bi} \cap V_{NCS})$

where V_{Bi} the volume of building(i) , $i \in \{1,2,\ldots,N\}$
N the number of buildings on site
V_{NCS} the z extrusion of the site non-constructible area

B: BUILDINGS OVERLAP

$NC_B = - [(V_{B1'} + V_{B2'} + \ldots + V_{BN'}) \cap (V_{B1'} \cup V_{B2'} \cup \ldots \cup V_{BN'})]$

where $V_{Bi'}$ is the volume produced by an offset of the buildings volume to x/2,
$i \in \{1,2,\ldots,N\}$,N the number of buildings on site
x is the minimum distance between two buildings,

C: ACCESSIBILITY REQUIREMENTS

$$NC_C = \begin{cases} -d & \text{,when } SC_C = \text{FALSE} \\ 0 & \text{,when } SC_C = \text{TRUE} \end{cases}$$

where d the distance between the building's access area and the site access area

MO Optimization

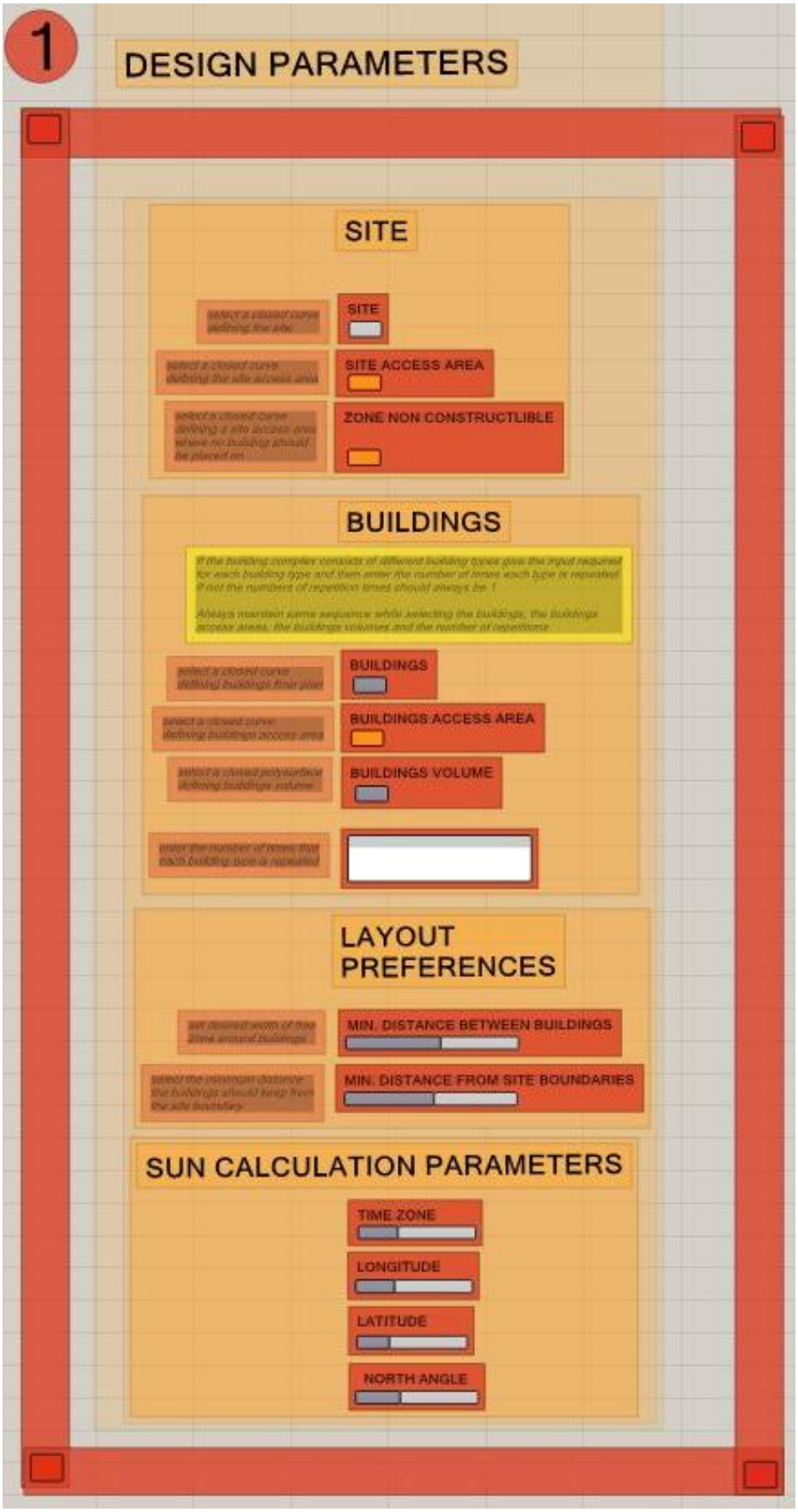
$\sum F' = \sum F * w$, where $0 \leq w \leq 1$ the weight of the criterion

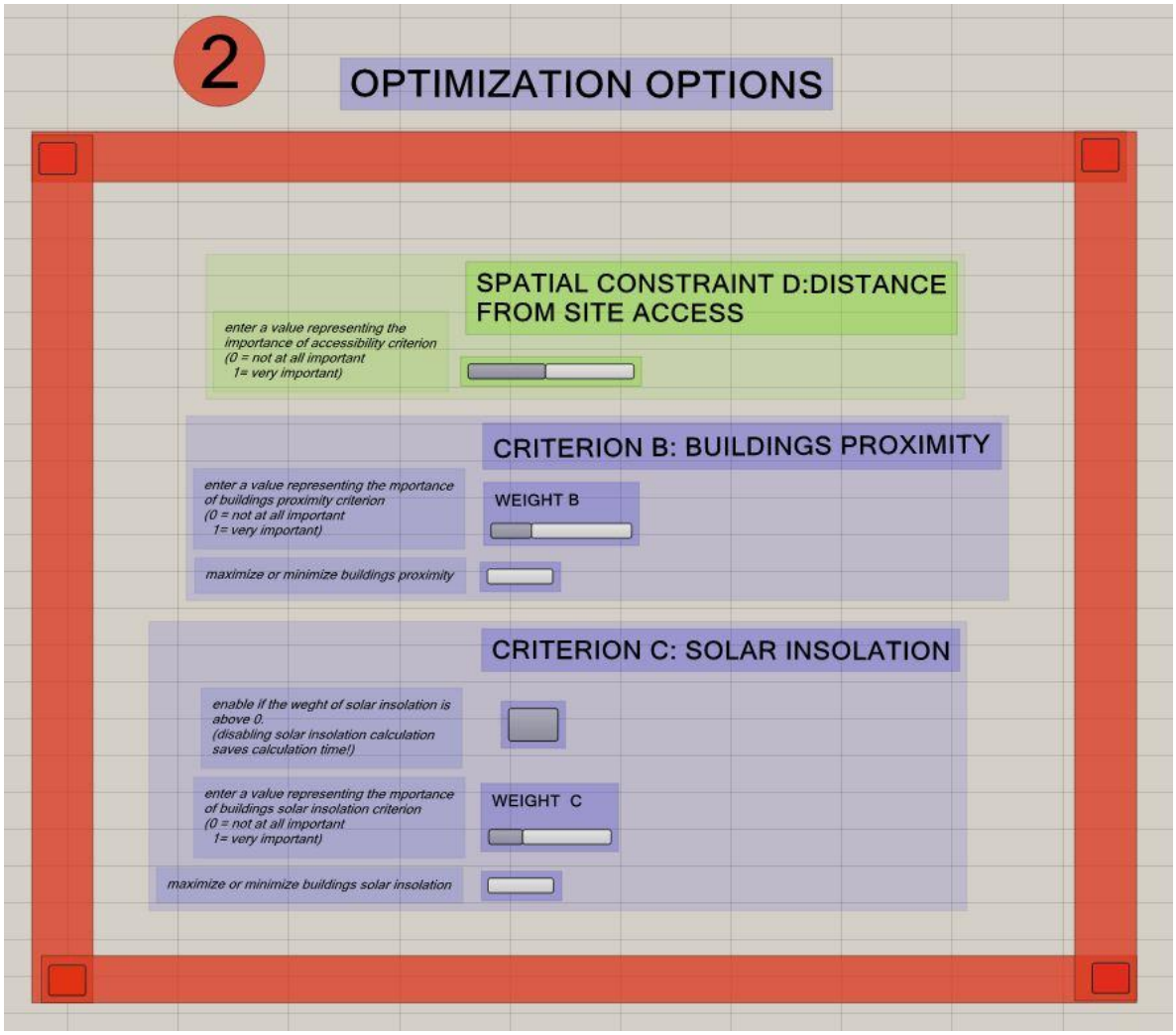
$\sum NC = NC_A + NC_B + NC_C$

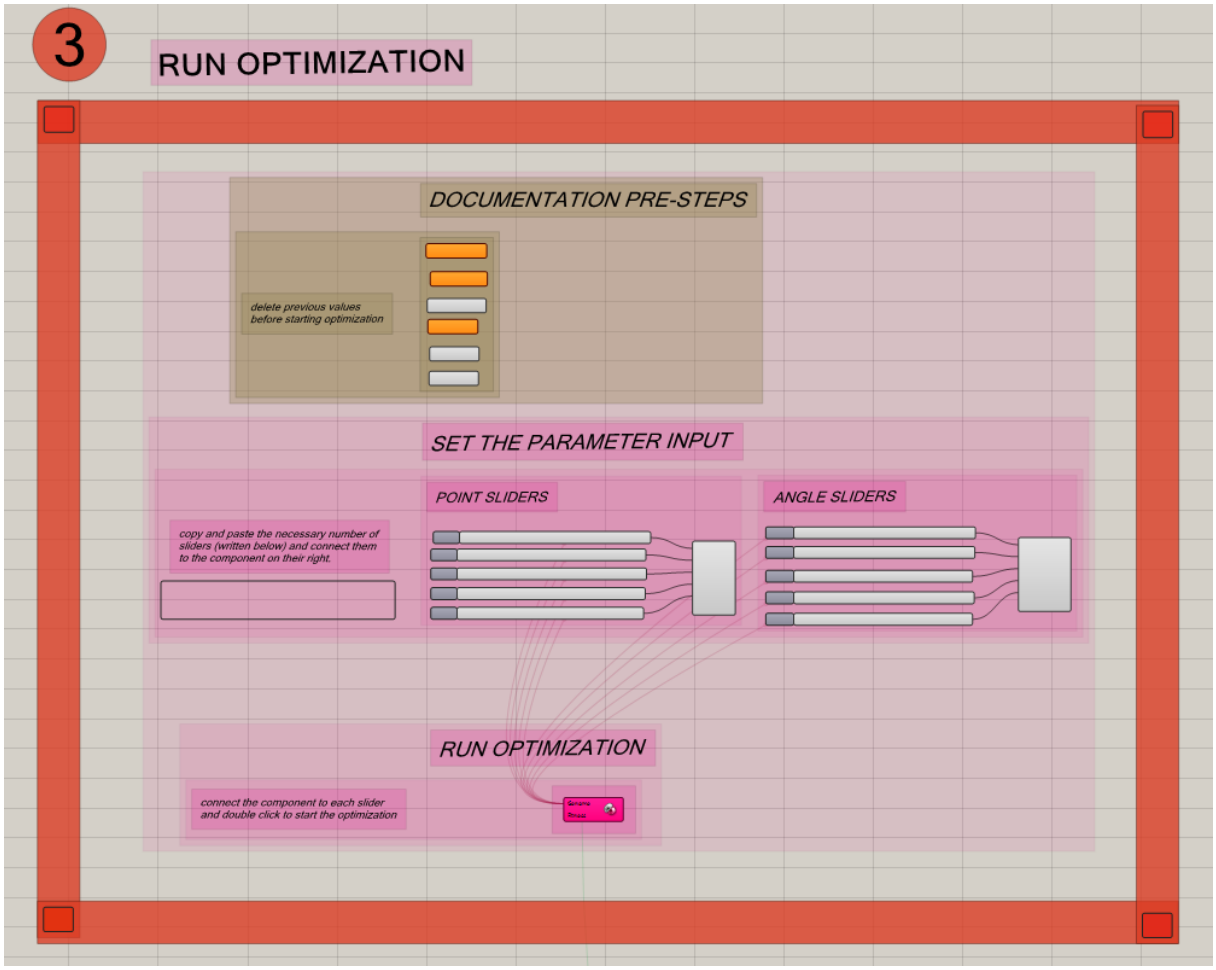
$F_{Final} = \sum F_A' + \sum F_B' + \sum NC$, for criteria A and B

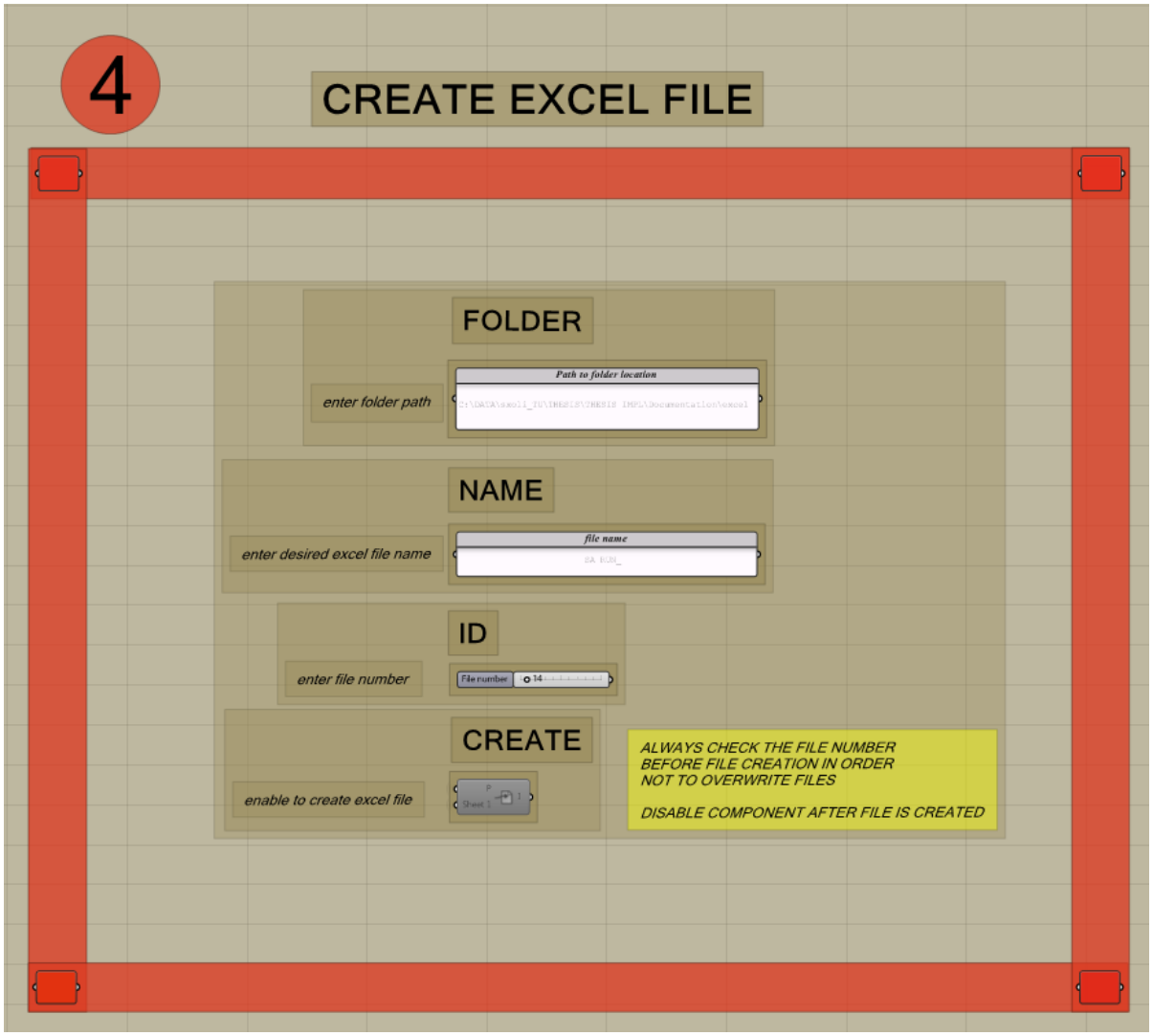
$F_{Final}' = F_{Final} * SC_A * SC_B * SC_C$

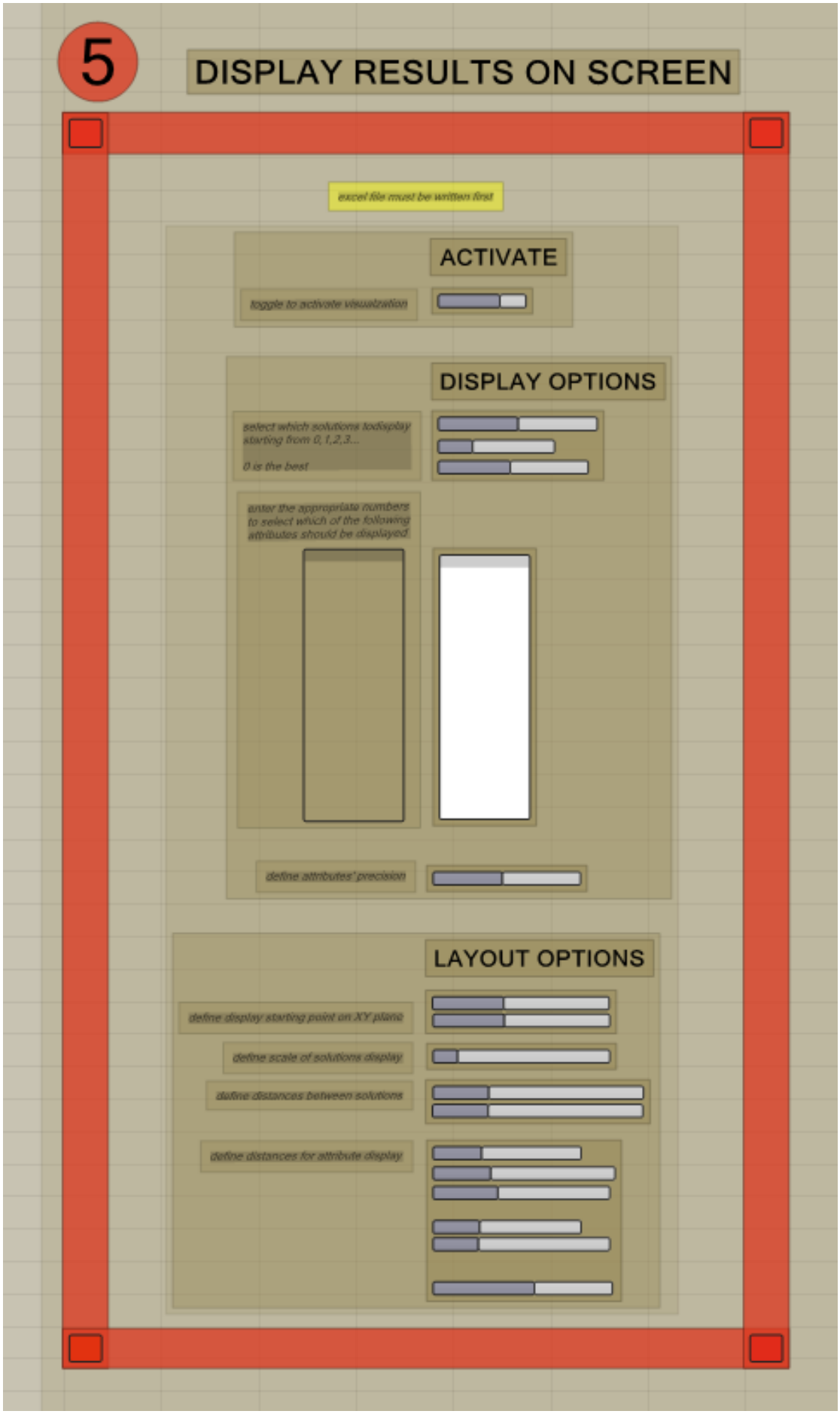
Appendix B. User Interface




2
OPTIMIZATION OPTIONS
SPATIAL CONSTRAINT D:DISTANCE FROM SITE ACCESS
enter a value representing the importance of accessibility criterion (0 = not at all important 1= very important)
CRITERION B: BUILDINGS PROXIMITY
enter a value representing the mportance of buildings proximity criterion (0 = not at all important 1= very important)
WEIGHT B
maximize or minimize buildings proximity
CRITERION C: SOLAR INSOLATION
enable if the weght of solar insolation is above 0. (disabling solar insolation calculation saves calculation time!)
enter a value representing the mportance of buildings solar insolation criterion (0 = not at all important 1= very important)
WEIGHT C
maximize or minimize buildings solar insolation


3
RUN OPTIMIZATION
DOCUMENTATION PRE-STEPS
delete previous values
before starting optimization
SET THE PARAMETER INPUT
POINT SLIDERS
ANGLE SLIDERS
copy and paste the necessary number of
sliders (written below) and connect them
to the component on their right.
RUN OPTIMIZATION
connect the component to each slider
and double click to start the optimization


4
CREATE EXCEL FILE
FOLDER
enter folder path
Path to folder location
NAME
enter desired excel file name
file name
ID
enter file number
File number
14
CREATE
enable to create excel file
ALWAYS CHECK THE FILE NUMBER
BEFORE FILE CREATION IN ORDER
NOT TO OVERWRITE FILES
DISABLE COMPONENT AFTER FILE IS CREATED


5
DISPLAY RESULTS ON SCREEN
excel file must be written first
ACTIVATE
toggle to activate visualization
DISPLAY OPTIONS
select which solutions todisplay starting from 0, 1,2,3...
0 is the best
enter the appropriate numbers to select which of the following attributes should be displayed
define attributes' precision
LAYOUT OPTIONS
define display starting point on XY plane
define scale of solutions display
define distances between solutions
define distances for attribute display

Appendix C. Genetic Solver

Provided in Galapago's documentation by David Rutten

(http://ieatbugsforbreakfast.wordpress.com/2011/03/04/epatps01/)

PROCCESS

For the problem subject to optimization we should be able to define a set of variables to whom in terms of evolutionary computing we refer to as genes. The fitness of every solution is the function, depending on these variables, that decides how suitable every solution is or how close it is to the optimal solution, if there is one. For a model that would contain two variables (genes) the fitness landscape would look like Figure C.1.

As we change Gene A, the fitness of the solution model changes and it either becomes better or worse (depending on what we're looking for). But for every value of A we can also vary gene B, resulting in better or worse combinations of A and B. Every combination of A and B results in a particular fitness and this fitness is expressed as the height of the Fitness Landscape.

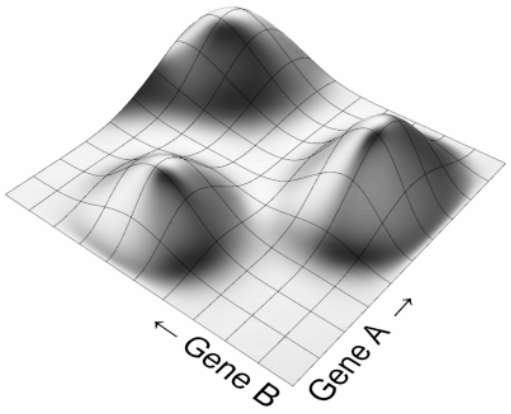


Figure C.1: : Fitness Landscape for a problem with two variables

Of course a lot of problems are defined by a much higher number of genes, in which case there is no longer a "landscape" in the traditional sense but a n-dimensional fitness volume deformed in n+1 dimensions,(where n is the number of genes), as in the previous example a two dimensional fitness plane is deformed in 3 dimensions.

For simplification reasons the process is explained using the two-dimensional example.

Beginning to find the best fitting solution we start with a randomly selected population of genomes – Generation 0-since we have no information about what should be the starting point. After the fitness of every genome of generation 0 is evaluated, they are sorted from lower to higher value. Based on the reasonable assumption that the higher genomes are closer to the highest solution, the worst performing genomes are excluded and the remainder are the ones on which the focus is on.

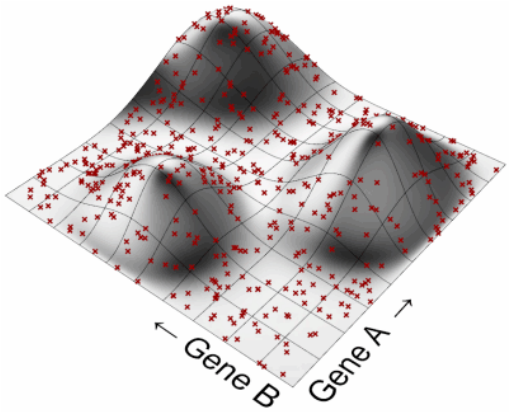


Figure C.2: Generation 0 Population

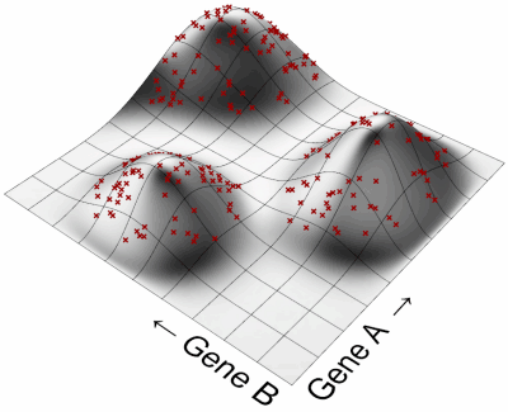


Figure C.3: Intermediate Generation Population

By breeding these more efficient genomes we can form the next generation, generation 1. When we breed two genomes their offspring will end up somewhere in the intermediate model space; thus exploring fresh ground. The new population, which is no longer completely random, is already starting to cluster around the fitness "peaks". Repeating the above steps will lead eventually to the highest one.

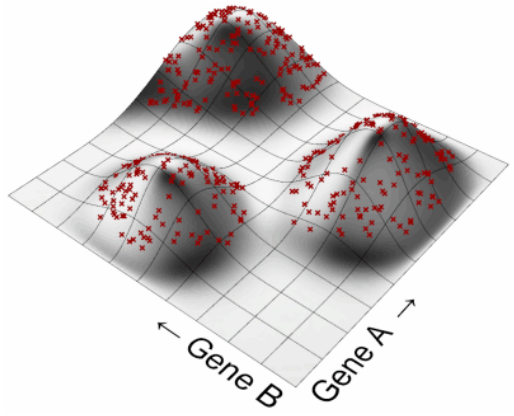


Figure C.4: Later Generation Population

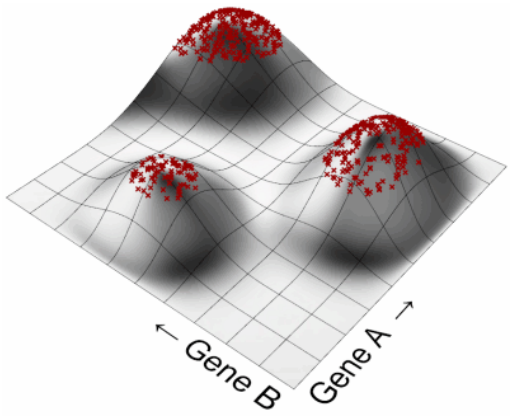


Figure C.5: Last Generation Population

In order to perform this process, an evolutionary solver requires five interlocking parts, which will be further described:

- Fitness Function
- Selection Mechanism
- Coupling Algorithm
- Coalesence Algorithm
- Mutation Factory

Fitness Functions, Definition of Fitness

Opposed to biological evolution where fitness is the result of a huge number of conflicting forces, in evolutionary computation fitness is a very easy concept. It can be whatever we want it to be. When we are trying to solve a specific problem we know what it means to be fit and we can define a strict fitness function to express it.

Taking once more as an example the fitness landscape that refers to a problem with two variables, we can depict in the following shape how every genome is trying to improve its fitness through the process of the solver. Focusing on a genome that was initially positioned on the bottom of the fitness landscape (Figure C.6) the route that represents the pathway of its most successful offspring is a route climbing uphill along the steepest slope to the nearest local optima. If this representation was done for a large amount of sample points the fitness landscape would look like the shape of Figure C.7

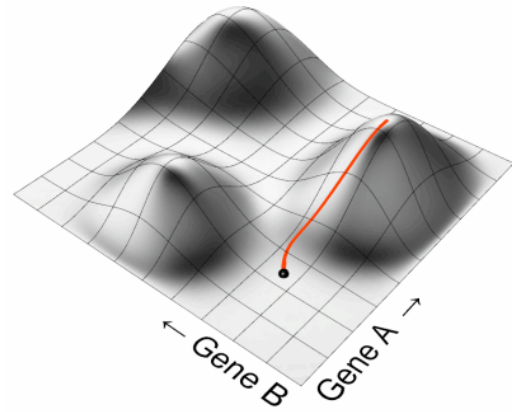


Figure C.6: Genome route

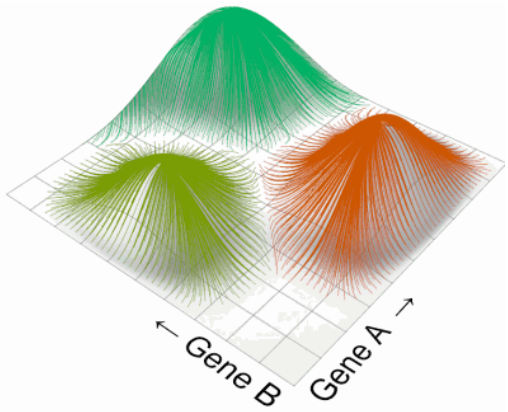


Figure C.7: Population routes

As shown in the figures above, every peak in the fitness landscape has a basin of attraction around it that represents all genomes that converge upon that specific peak. The area of each basin is definitely not representative of the quality of the peak. A poor solution can actually have a large basin of attraction, while a good solution may have a very small one, depending on the nature of the problem. This relation between the area of the basin of attraction and the quality of the solution and in general the form of the fitness landscape has major impact on how successful the solver is going to be.

In the following example (Figure C.8) we see a 2-dimensional graph representing the fitness landscape while the solver tries to find the minimum bounding box of a cylinder, using as variables two axis of rotation (two genes). We can see that such a problem could be successfully confronted by the solver: the fitness landscape is periodic, the solutions are of equal quality and there are no local optima. No matter where the starting point of a genome is, it will definitely end up in a peak that is of maximum quality.

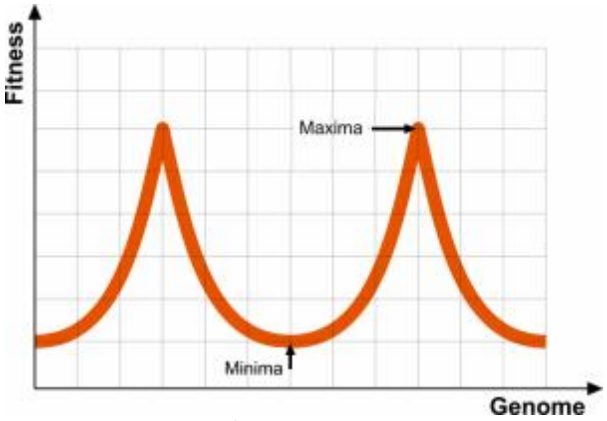


Figure C.8: Fitness Landscape

Unfortunately, not all problems can be so easily solved. In the following graphs Figure C.9-Figure C.12) we can see a number of different cases representing problems, whose solutions are much more difficult to find.

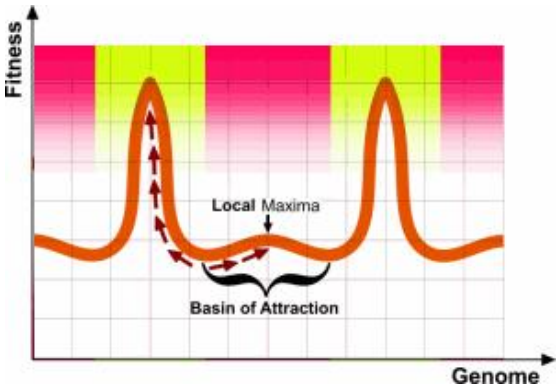


Figure C.9 Local maxima

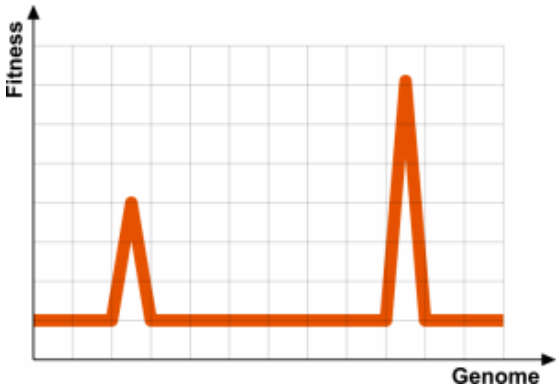


Figure C.10 Maxima with small basin area

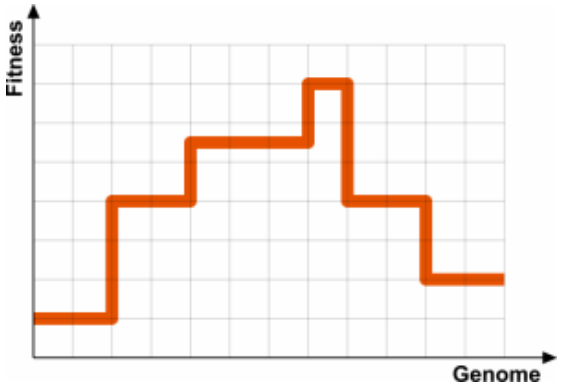


Figure C.11: Flat basins

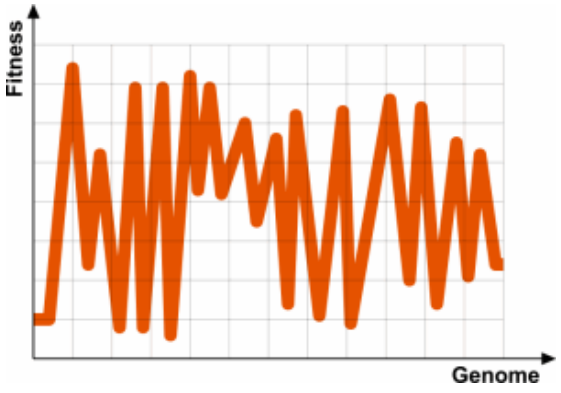


Figure C.12: Noise in fitness landscape

In Figure C.9 we can see that there are two types of solution: one of lower quality but larger area of basin of attraction (represented in pink color) and one of better quality but much smaller basin of attraction (yellow

color). It is easily understood that many genomes (almost half) can be trapped in the area of the poorest quality local maxima and thus result in an inefficient solution.
An even worst case is shown in Figure C.10, where the solutions are found only when a genome happens to be generated in a very small basin. Here it is really easy to totally miss the solution or miss the best solution when finding one local optima.

In Figure C.11 there are no forces leading the genomes while they are located in the intermediate steps toward the peak. Therefore it is not easy to follow a route from the poorest towards the best solution since the upgrade from step to step can only be done by luck without any indication of where the genome should lead to.

The last example Figure C.12 shows a fitness landscape with a large amount of noise, which makes it very hard- almost impossible- for the genomes to orient themselves towards the best solutions. Both ancestors may be of high quality but their offspring may result in a poor solution.

As seen through these examples the definition of the problem plays a significant role on how efficiently a genetic algorithm could act.

Selection

The process of selection refers to the selection of the genomes that get to "mate" and therefore pass their genetic characteristics to the following generation. There are many selection algorithms available in computation: isotropic, exclusive and biased selection.

Isotropic selection is the simplest selection algorithm; in fact it expresses the lack of a selection algorithm, since all genomes have equal chances to mate regardless of their fitness evaluation. (Figure C.13).

Even though this strategy may seem that is not furthering the evolution of the gene-pool, it still serves the efficiency of the solver since it acts as a hold up-mechanism towards the fast colonization of local optima which could lead to an inferior solution.

Even in nature such a selection mechanism exists indeed, like for example in wind pollination, coral spawning or in a walrus colony where all female of the colony get to mate independent of how fit they are.

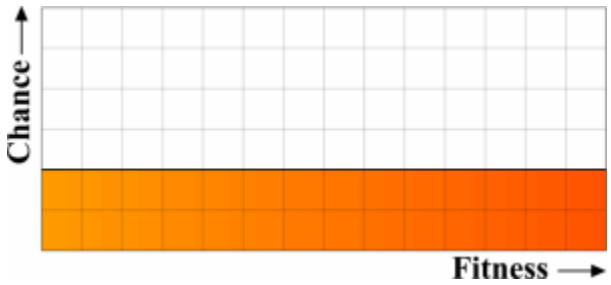


Figure C.13: Isotropic Selection

In Exclusive selection only the best N% of the population is selected to produce offspring for the next generation. In Biased Selection, the chances that a genome produces offspring increase as the fitness increases resulting in a curve like the one shown in Figure C.15 , which can be further flattened or exaggerated through the use of amplifying algorithms.

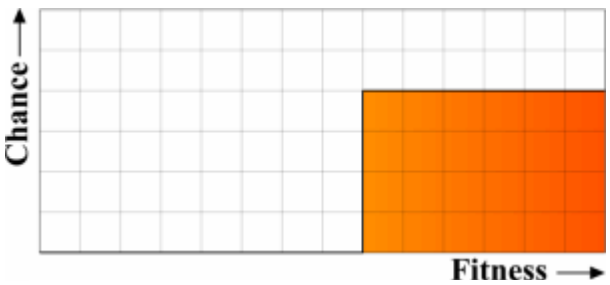


Figure C.14: Exclusive selection

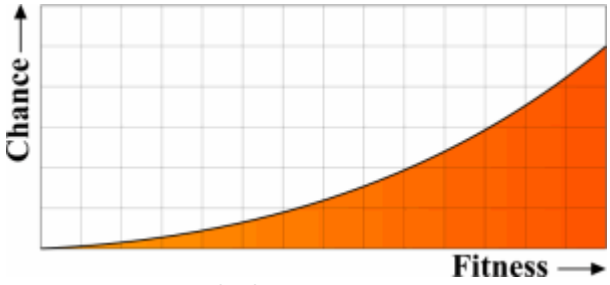


Figure C.15: Biased Selection

Coupling

As soon as the selection algorithm finds the appropriate genomes that will form the next generation, the coupling process begins. During this process each selected genome has to find its mate in order to produce the offspring. There are many algorithms for the coupling process.

This genomic distance is an abstract reference to the similarity of the genomes that are chosen for mating. This is represented in Galapagos' interface through a graph called Genome Map (Figure C.16), a two-dimensional mapping of the distance between the genomes in a multi-dimensional model space.

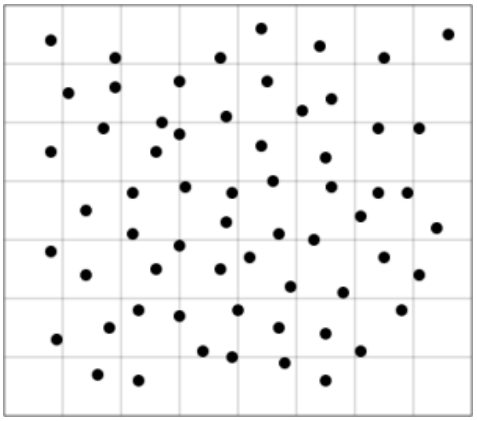


Figure C.16 : Genome Map

All individuals (genomes) are displayed as dots on a grid in a way that the distance between them relates to the distance in gene-space. Since it isn't possible to map an N-dimensional point cloud (where N is the number of genes) on a 2-Dimensional grid with pure accuracy, the genome map is by definition an approximation graph displaying rough analogies. Therefore there is also no value for the axis of the graph, the only information that we can get out of it is the similarity of two genes (depending on how close to each other or far away they are).

Based on this graph we can represent the ways that coupling takes place.

When a genome selects its mate from the area close it itself in the genome map, it selects a very similar genome and therefore the offspring that they produce will also be quite similar. This behavior is to some extend desirable but when taken to extremes it can harm the diversity of the population. It is usually characterized as incestuous mating behavior and it decreases the chances of finding alternative solution basins – thus increases the chances of getting stuck in local optima (Figure C.17).

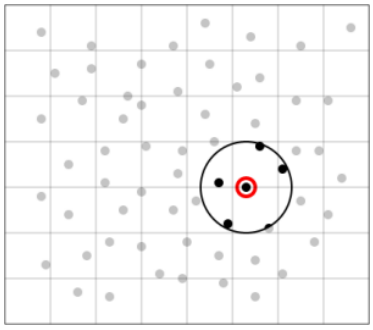


Figure C.17 : Incestuous mating

The other extreme is to mate with totally different genomes, located far away in the genome map. Excluding every genome that is near the selected one is called zoophilic mating (Figure C.18) and can also cause extinction, especially when the population is not a single group but a sum of several sub-species (Figure C.19). In that case it is possible that the offspring ends up somewhere in the middle between the two local optima, not resulting in a meaningful solution.

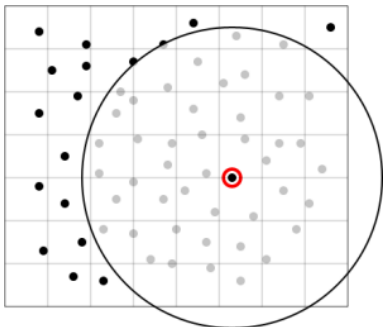


Figure C.18 : Zoophilic mating

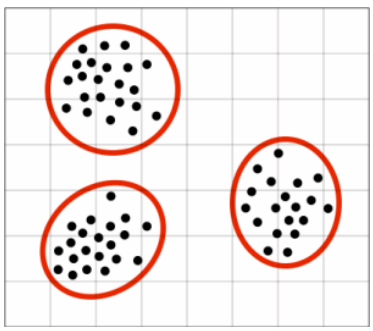


Figure C.19 : Population with sub-species

The best option would be to balance in- and out- breeding so that the individuals are neither too close nor too far (Figure C.20), taking of course in mind the nature of the problem that we are trying to solve. The morphology of the solution graphs of the problem and whether the problem has one peak or many is quite important for the coupling strategy that we will follow.

Working with Galapagos a certain degree of flexibility on the type of coupling strategy that we follow exists through the option of setting the in- and out- breeding.

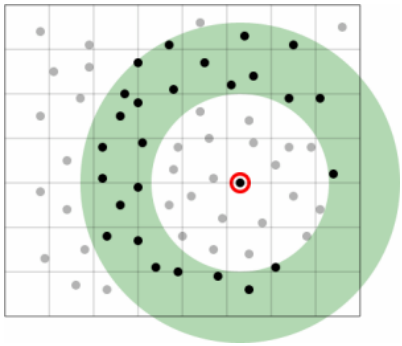


Figure C.20 : Balanced in- and out-breeding

The main disadvantage of the algorithm used, is that it completely ignores the fitness of the genomes that are selected for mating and only acts based on the similarity between them.

Coalescence

While producing the offspring of two genomes in order to populate the next generation, we have to decide what gene values are going to be selected. The gene recombination procedure in computation is much simpler than in nature, since the genes are continuously variable qualities that do not have discrete characteristics, but can assume instead all numerical values between two numerical extremes. There are many mechanisms for this procedure like crossover, blend and preferenced-blend coalescence.

Crossover coalescence is the recombination of genes between two genomes when the offspring inherits a random number of genes from one genome and the rest from the other one, so that the gene value is maintained. This procedure is best suited for cases when the two genomes initially selected to mate, are quite similar with each other.

During blend coalescence, the gene values are also changing, basically adapting an average value between the two original genes.

If we apply preferenced – blend coalescence, then the new value assigned to the offspring gene, tends to be more similar to the original gene value displaying the higher fitness.

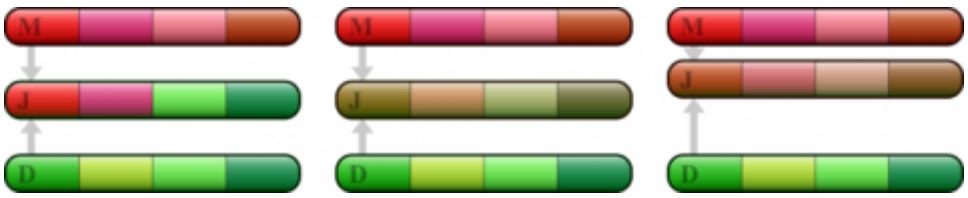


Figure C.21 : Crossover, Blend and Preferenced-Blend Coalesence

Mutations

The previous mechanisms incorporated in the procedure of producing the next generations improve the quality of the solution generation by generation, but unfortunately they tend to reduce the bio-diversity of the population causing in extreme cases the population to extinct or get caught in local optima not finding the best solution. The mechanism that can actually further bio-diversity is mutation.

The explanation of mutations will be done through Genome Graphs. A Genome Graph is a graph allowing the 2-dimensional representation of a multi-dimensional point. Every point in multi-dimensional space is displayed as a series of lines that connect different values. X axis is representing the genes and thus every vertical bar is another dimension. The Y axis is showing the gene values. These are absolute values in the sense that their value is showing their location regarding to the lower and upper limit. Using Genome Graphs not only we can describe multi-dimensional points but we can also represent points with different number of dimensions on the same graph.

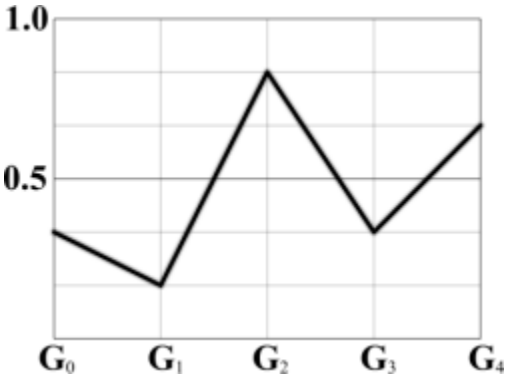


Figure C.22: Genome Graph of a 5-gene genome

During point mutation a single gene value is altered. Point Mutation is the only mutation available in Galapagos. Another type of mutation is swapping the values of two neighboring genes. This type of mutation (inversion mutation) is useful only when it serves an equivalent relation between the genes. Otherwise it has a negative impact on the process.

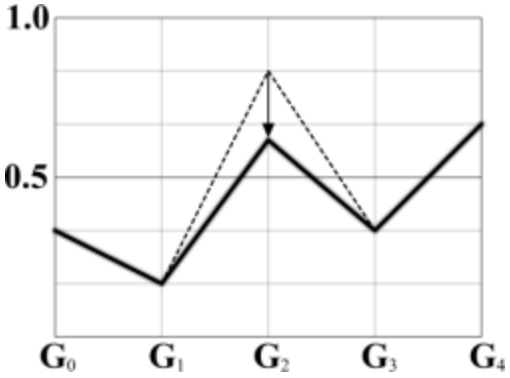


Figure C.24 : Point mutation

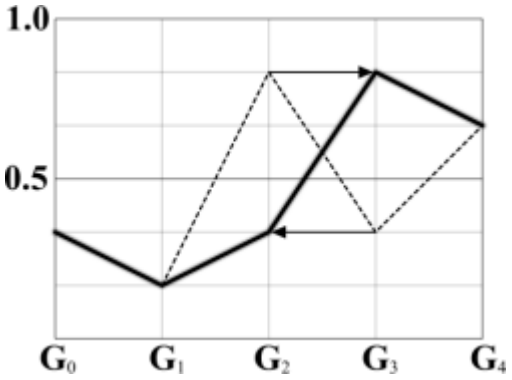


Figure C.23 : Inversion Mutation

Adding or deleting a gene is performed when the genomes do not require a fixed number of genes. This is not possible when working only with a fixed number of genes.

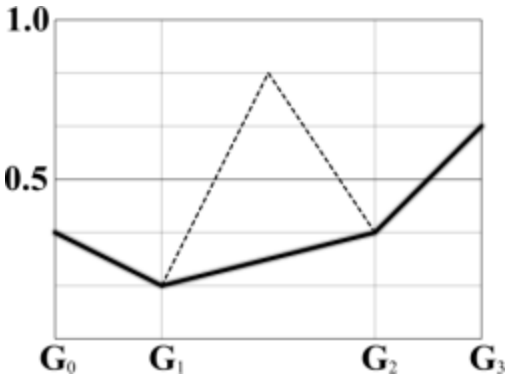


Figure C.26 : Deleting a gene

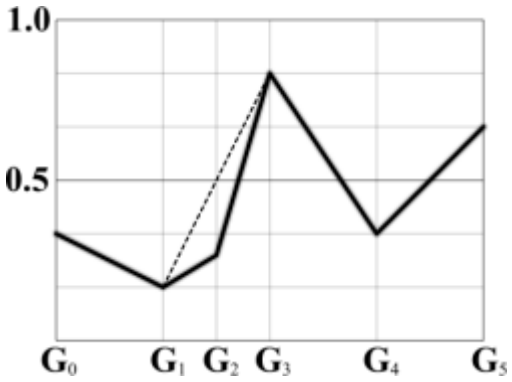


Figure C.25 : Adding a gene

Appendix D. Simulated Annealing Solver

Provided in Galapago's documentation by David Rutten

(http://ieatbugsforbreakfast.wordpress.com/tag/simulated-annealing/)

The second solver available within Galapagos implements the Simulated Annealing algorithm. Like the existing Evolutionary solver, Simulated Annealing is also a meta-heuristic technique, but works in a fundamentally different fashion. Having access to both solvers makes it easier to circumvent some of the shortcomings of each. Ironically, Simulated Annealing is a much simpler process than Simulated Evolution but may be harder to understand since the real-world analogy is more abstract and based on a less well known process.

In metallurgy, annealing is the process of controlled heating and cooling of metal to achieve certain material properties. At first, the metal is heated up to melting point so it can be cast or formed. At an atomic level, heat is nothing more than particle velocity. The particles (atoms & molecules alike) in a hot substrate move faster than the same particles in a cold substrate. At some point the velocity of two particles will be so high that they cannot succeed in forming a persistent bond between them. When this happens the substrate loses internal structure and turns liquid.

Similarly, when a substrate is liquid but starts to cool down, there will come a point where the atoms can form lasting bonds and the internal structure of the substrate is resurrected, turning the liquid into a solid. In 1866 James Clerk Maxwell formulated the equations that described the distribution of particle velocities in a gas of constant macro temperature. For example, a litre of helium gas at room-temperature contains roughly 3 × 1022 atoms and the most probable speed of these atoms is a little over a 1000 meters per second. However there will be a lot of very slow atoms as well as a few much faster ones. This same phenomenon holds for liquids too, although the velocity distributions are not as well defined nor do they cover quite so large a range.

The upshot of all this is that when a substrate is allowed to cool, some atoms will cross the liquid/solid threshold before others. In other words, a substrate doesn't freeze everywhere at once, small clumps of relatively slow atoms group together and form the freezing 'seeds'. Especially when we're talking about metals this is important because when metal atoms freeze, they like to form a regular lattice, or crystal.

These small islands of glued together atoms grow over time as more and more slow atoms attach themselves to the seeds, allowing the micro-crystals to expand. When cooling lasts long time, atoms are allowed to find the optimal (minimal energy) distribution. If on the other hand cooling is very quick —for example by dumping the hot metal into water— there is no time to form large crystals and the substrate becomes amorphous.

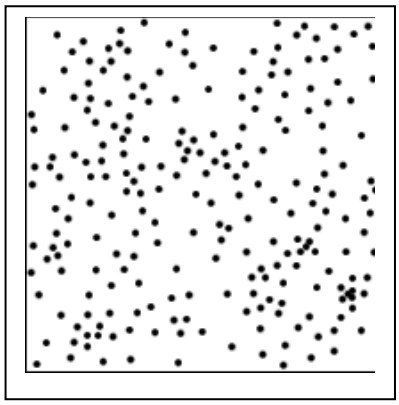


Figure D.2: Atoms in liquid metal

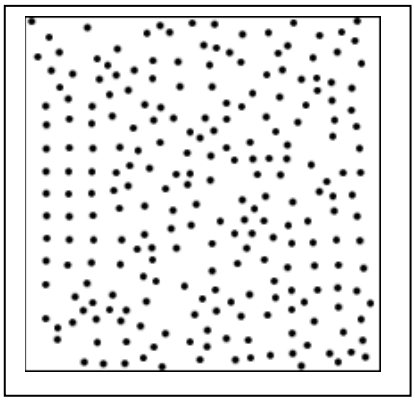


Figure D.3 Crystal seeds in semi-liquid metal

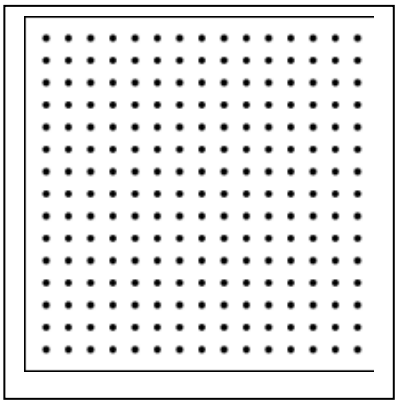


Figure D.4: Regular atomic lattice

Treating all the parameters in a problem as an atomic thermodynamic system allows finding relatively good answers relatively quickly. Basically, with Simulated Annealing, Galapagos seeks to crystallize the parameters into the lowest energy state.

A typical annealing run consists of a number of successive jumps in the problem phase-space, where the amplitude of each jump and its legitimacy are affected by the temperature of the system. When the temperature of the system is high, large jumps are allowed and there is a significant likelihood that a worse answer is adopted despite being a setback. The graph below shows a typical annealing track.

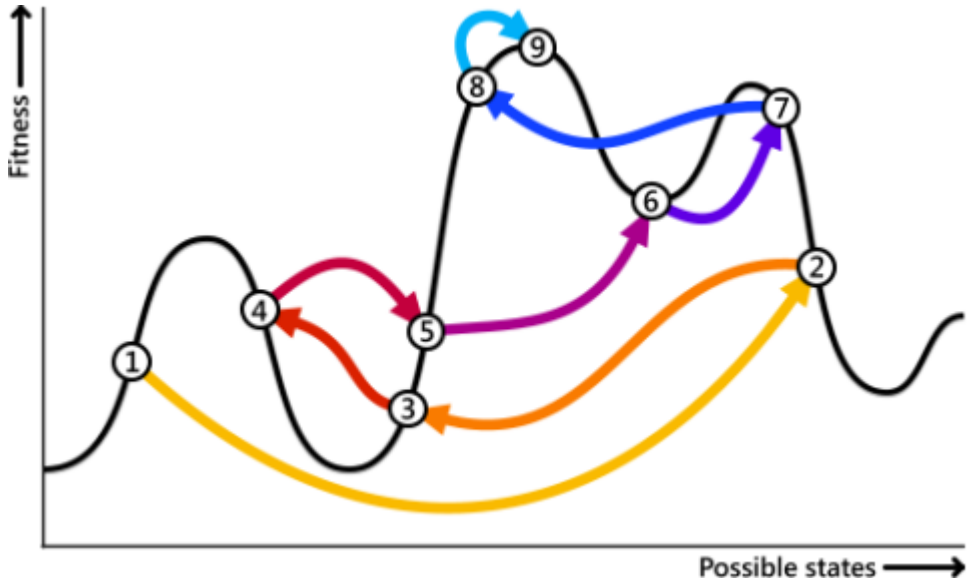


Figure D.5: A schematic annealing track

Along the horizontal axis all possible states of the problem are collected. If the problem phase-space is one-dimensional (i.e. only a single Grasshopper slider), then this is an exact representation, but since this is a schematic representation the graph x-axis can represent any number of parameter dimensions. The vertical axis represents the fitness of each distinct state, so the thick black curve represents the entire fitness landscape. This particular landscape has 4 local optima with varying degrees of quality.

The first annealing jump must start at a random location because nothing is known about the landscape so no informed decision can be made. In this particular example, the annealing track starts along the left edge of the landscape at location (1). At this point in time the temperature is very high, so large jumps across phase-space are allowed. A jump from (1) to (2) would indeed qualify as large as it spans almost the entire phase-space. Since (2) is higher than (1), the new solution is automatically accepted. Now there is less energy available as the entire system is cooling down. So the jump from (2) to (3) is most likely going to be shorter than the jump from (1) to (2). The fitness at (3) happens to be lower than (2), so the second jump actually represents a worse answer. However, since the temperature is still relatively high, sometimes (where “sometimes” is in accordance with thermodynamic stochastics) the worse case is accepted. This characteristic makes the solver less prone to getting ‘stuck’ to a local optimum.

As time goes on, the temperature of the system drops and smaller and smaller jumps are possible. Also, at low temperatures the chance that a worse case is adopted over a better one becomes insignificant. Eventually the temperature has dropped far enough for the entire system to be frozen, at which point the best answer from the run is cached and a second run is started, once again at high temperature.

Because of the cooling schedule, it means that every annealing run only has a limited number of steps and therefore terminates in a limited time-span.