Department of Information Science and Technology

Shaper-GA: Automatic Shape Generation for Modular housing

Bruno Miguel Teixeira Taborda

A Dissertation presented in partial fulfilment of the Requirements for the Degree of Master in Computer Engineering

Supervisor:

PhD, Ana de Almeida, Assistant Professor, ISCTE-IUL

Co-supervisor:

PhD, Filipe Santos, Assistant Professor, ISCTE-IUL

September, 2018

Resumo

Este trabalho apresenta um sistema automático que, a partir da especificação de uma linguagem arquitetural de design, gera plantas alternativas para residências de construção modular.

O sistema usa Algoritmos Genéticos e é capaz de produzir várias soluções de plantas de modo eficiente. As regras de arquitetura são implementadas na função de fitness a partir de uma Gramática de Forma criada pelo arquiteto.

São geradas diferentes soluções de plantas exequíveis, isto é, soluções que obedecem à Gramática de Forma e não têm sobreposições entre as suas divisões. Pode ser futuramente integrado com uma interface amigável para o utilizador de forma a que este personalize e crie a sua futura casa. Tal ferramenta pode também ser entregue às companhias de construção de forma a que estas gerem uma planta para uma casa modular personalizada.

Palavras-Chave: Algoritmos genéticos, Design de layout automático**,** Cutting and Packing, Linguagem de Design

Abstract

This work presents an automatic system that, from the specification of an architectural language of design, generates several alternative floor plants for the construction of modular homes.

The system uses Genetic Algorithms and is capable of efficiently producing various plant solutions. The rules of architecture are implemented in the fitness function translating the rules of a Shape Grammar created by the architect.

Different solutions of feasible plants are generated, that is, solutions that obey the rules of Shape Grammar and do not have overlays between the rooms. The system can be integrated with a user-friendly interface in the future, to allow for the house owners customization of their own house. Such a tool can also be delivered to construction companies for them to manage the design of modular houses that meet specific clients requirements.

Keywords: Genetic Algorithms, Automatic Layout Design, Cutting and Packing, Language of Design

Acknowledgments

I would like to thank Professor Ana Maria de Almeida and Professor Filipe Santos for all the support and availability during this dissertation. In addition, I thank them and Professor Sara Eloy for all the confidence and teachings during this dissertation.

My current areas of interest were triggered by some classes with Professor Ana Maria de Almeida, Professor Luis Nunes, Professor Luis Botelho, Professor Filipe Santos and Professor Pedro Sebastião. They all have a great personality and are also the best teachers of ISCTE-IUL, in teaching and motivating students in their goals.

I would also like to thank my colleagues, friends, family and girlfriend who supported me in this dissertation.

Index

Index of images

Index of tables

Abbreviations

2SLOPP	**T**wo-**D**imensional **S**ingle **L**arge **O**bject **P**lacement **P**roblem
2D-SPP	**T**wo-**D**imensional **S**trip **P**acking **P**roblem
C&P	**C**utting **&** **P**acking
DSR	**D**esign **S**cience **R**esearch
EA	**E**volutionary **A**lgorithm
EC	**E**volutionary **C**omputation
GA	**G**enetic **A**lgorithm
GECCO	**G**enetic and **E**volutionary **C**omputation **C**onference
KSG	**K**wiecinski **S**hape **G**rammar
RWS	**R**oulette **W**heel **S**election
RRC	**R**andom **R**espectful **C**rossover
VLSI	**V**ery-**L**arge-**S**cale **I**ntegration

Chapter 1. Introduction

This chapter provides an overview of the context that brought up the challenge addressed by this dissertation. Over the next sections, the problem will be formally formulated, and we will present, in sequence the following topics: motivation, research methodology, objectives and proposed approach. The structure of this dissertation will also be outlined at the end.

1.1. Introduction

This dissertation addresses the problem of the automation of floor plan design in accordance with a given architectural language of design for modular housing mass production.

The first prefabricated houses appear during the gold rush in United States of America. This type of housing became popular during the Second World War, with the need to accommodate huge amounts of military personnel. The construction speed and the mobility of prefabricated houses were also a plus comparing with standard houses. Nevertheless, the idea of massive fabrication of houses seems to have started in mid 19th century and, since then, prefabricated houses have been successfully used (House, 2011). Modular houses should not be taken as single elements (i.e. doors, windows, walls) but as a composition of single elements that compose the house (rooms or divisions). The implementation of modular houses emerged with Fuller experimentation in 20s and 30s of last century and the Dymaxion House, that incorporated prefabricated bathroom modules. These houses embody a huge variety of construction elements, based on the climate and location of the house. While in standard housing construction increasing the house is not easy (e.g. add more rooms), with modular houses this becomes an is easy task to solve.

Modular construction is also used for skyscraper and commercial buildings. McDonald's uses prefabricated structures, having set a world record of 13 hours since the beginning of construction until the opening to the public. Mass and modular construction is raising nowadays and has been in the highlights, like with the 461 Dean Street modular skyscraper in Brooklyn, NY, USA. On the other hand, the customization of a modular house by its future owner is very limited so far, even with the architectural progress that resulted mainly in the development of design processes. In Poland, the search for a detached or semidetached familiar house is an ingrained cultural notion, and the most popular construction is that of modular wooden houses. In 2014, Kwiecinski and Slyk

presented a formal language of design (Kwiecinski & Slyk, 2014) to solve the customization problem, with the intention of calling Polish future house owners to participate in the design of their houses. This language is defined by using specified building elements (rooms or divisions), with predetermined range of individual dimensions (shapes) and architectural rules that restrain the possible combinations of housing elements' positions (Chapter 3). This language was translated into a shape grammar that allows for automatization of the floor planning design (Chapter 3). However, a pure procedural generation of a floor plan was drawbacks. Using a backtracking approach can, not only be a morose process, but also be inefficient, since the dimension of the floor plans' search space is combinatorial in nature. Therefore, we proposed to explore the development of an automated floor plan design complying with the language of design using an evolutionary approach.

1.2. Problem Formulation and Motivation

Most of the single-family houses build in Poland are not directly designed by architects but are built based on the documentation presented in catalogues of typical houses. Such architecture gained popularity, mostly due to low prices of purchase, even though they often do not suit local conditions of the site or offer poor flexibility in accommodation the needs of their users. In order to allow for costumer participation in their houses design, Kwiecinski and Slyk presented an architectural interface to generate mass-customized modular wooden houses in Poland (Kwiecinski & Slyk, 2014). In 2016, based on (Kwiecinski & Slyk, 2014), a shape grammar was created to define the previous model (Kwiecinski, Santos, De Almeida, Taborda, & Eloy, 2016). This language specifies that each house, after total width and depth are given, has a customized number of rooms, each one with predetermined possible dimensions. The design restraints the house layout by defining relative positioning relations between different types of rooms (like for kitchen and dining room) establishing a central axis that equally divides the total width of the house in two, allowing for a central corridor to access all the rooms. Other concepts are also incorporated in the language, like the ones of public and private areas. As such, a house layout can be viewed as a set of positions that represent the spatial relations between rooms – a floor plan.

In order to formally define the problem to be solved, one must begin by specifying the constraints and modeling the objectives to be met. As already mentioned, the total layout area is previously fixed. Both the area and the rooms are defined by rectangular formats,

thus having a width and a depth as dimensions. The rooms (smaller rectangles) must be placed in a way that all rules of the shape grammar that implements the design (Kwiecinski et al., 2016) are fulfilled. The architectural rules present in the shape grammar define relative positioning constraints for the rooms and rotations are not allowed. An optimal solution, or proper layout, is one that obeys all the shape grammar rules and layout positioning constraints. The latter consist in obvious restraints: the rooms may not cross the central axis that divides the total width of the house and may not overlap. Thus, the problem to be solved is a two-dimensional Cutting & Packing problem. More specifically, this problem is considered within the Bin-Packing class of NP-hard problems (Maxence Delorme, Manuel Iori, 2016). In fact, and even for our particular scenario, the number of possible arrangements for the placement of the rooms in the given layout area is combinatorial. For the small example described in (Almeida, Taborda, Santos, Kwiecinski, & Eloy, 2016), having 8 different rooms with fixed dimensions, there are about 2×10^{12} possible combinations for the positioning of the rooms, that is, different floorplans.

The problem to be solved may be looked as a two-dimensional single large object placement problem (2SLOPP) (Gerhard Wäscher, Heike Haußner, 2007), where the overall dimensions are both fixed and there are further positioning restrictions to be obeyed. If variable depths are allowed in rooms, the problem formulation change and may be seen as a two-dimensional strip packing problem (2D-SPP) (Bortfeldt, 2006) (Thomas, 2013) where the objects have fixed width and variable depth.

1.3. Research Context

1.3.1. Research Method

The system here proposed was developed under the vision Design Science Research Methodology (DSRM) principles proposed by (Hevner & Chatterjee, 2010). Design Science Research (DSR) paradigm is intended to guide the development of inventions or artifacts, "defining ideas, practices, capabilities and products to accomplish effectively and efficiently the use of information systems (Hevner, March, Park, & Ram, 2004). DSR targets the development of artifacts with the purpose of refining its functional performance, thus, to attain human goals (Simon, 1996)

. The artifact is considered as a method that define a process and will lead on how to solve the specific problem.

Peffers et al., (2007) propose seven guidelines for a design science research methodology (Peffers, Tuunanen, Rothenberger, & Chatterjee, 2007), which we adapt for our artifact (automatic floor plan design system) construction: Problem identification (Section 1.2), define objectives for a solution (what should be answered with the present approach), design and development (iterative artifact development), demonstration (use of a concrete shape grammar for empirical tests achieving a valid house layout), evaluation and communication (final evaluation with experts and publications in peer-reviewed events).

1.3.2. Research Objectives

This dissertation main goal is to propose an automated floor planning application for modular house layout user customized. This means that feasible solutions that fulfill the architectural rules and typological specifications provided by the end-users must be generated, making the process of acquiring a modular house a participated one.

Thus, the following artifacts are to be created: (a) A Genetic Algorithm method able to produce layout solutions obeying the architectural rules; and (b) An interface (model) that retrieves the end-user's specification input, runs the method and creates a visualization of the results.

The main research questions that guide this dissertation are:

RQ 1. Is the Genetic Algorithm approach viable for generating effective house layouts, that is, obeying the architectural rules of design?

RQ 2. Is such an approach time feasible in close to real-time?

RQ 3. How to implement dynamic room's dimensions?

1.4. Thesis contributions

The main contribution of this dissertation is the development of a system (method), Shaper-GA, that is, an artifact able to generate multiple solutions based on the design rules defined by the architects. This system, whose iterative design and development is described by Chapters 4 and Chapter 5, enables the user to participate in the design process of the house.

Other contribution is the interface artifact that enables the non-expert user to specify the personal requirements (number and type of rooms and room's dimensions) into the design system.

The communication of results has already begun with the publication of two papers in scientific related conferences, one of those being one of the most prestigious in the area of Evolutionary Algorithms, GECCO, and the other the IEEE International Conference on Systems, Man, and Cybernetics:

Almeida, A. De, Taborda, B., Santos, F., Kwiecinski, K., & Eloy, S. (2016). A genetic algorithm application for automatic layout design of modular residential homes. Proceedings of the 2016 IEEE International Conference on Systems, Man and Cybernetics (SMC), 2774–2778.

Taborda, B., de Almeida, A., Santos, F., Eloy, S., & Kwiecinski, K. (2018). Shaper-GA: Automatic Shape Generation for Modular House Design. In Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '18 (pp. 937–942). New York, New York, USA: ACM Press.

1.5. Outline

After this introductory chapter, the remainder of this dissertation is organized as follows: In Chapter 2, a review on the most relevant related literature is presented. Major preliminary concepts along with some more specific scientific references for this dissertation are addressed in Chapter. Chapters 4 and 5 explain in detail the system created, the implementation, the results and its discussion. This dissertation ends presenting the main conclusions and directions for future work in Chapter 6.

Chapter 2. Literature Review

The following chapter shows the literature related with Shaper-GA, more specifically, Cutting & Packing, Automatic Layout Design and Genetic Algorithms.

2.1. Cutting & Packing

Cutting & Packing are problems of combinatorial optimization presenting multiple techniques and approaches. In the past, C&P problems were spread over the literature with different names (e.g. cutting stock, vehicle loading) and Dickhoff suggested a typology to classify such problems (Dyckhoff, 1990). This approach was very helpful to organize old and new literature but became insufficient for recent problems. Wäscher et al. suggested an improved typology to classify C&P problems (Gerhard Wäscher, Heike Haußner, 2007) based on Dickhoff's approach, introducing new criteria for the categorization. C&P can be divided in two main problems, Two-Dimensional Bin Packing Problem (2BP) and Two-Dimensional Strip Packing Problem (2SP) (Lodi, Martello, & Monaci, 2002), which are strongly NP-hard problems, making evolutionary algorithms stand as a valid method to solve such problems. Due the usual multidimensionality and vasteness of the search space, an efficient algorithm is required. The inherent parallelism of GAs makes the choice of this algorithms valid for optimization, especially for multi-criteria optimization. In 2005, a GA approach to solve 2BP of polygonal shapes on a rectangular canvas was created (Ayala-ramirez et al., 2005). Hybrid methods have also been implemented and tested. The authors of (Gharsellaoui & Hasni, 2012) implemented a hybrid algorithm that combines a genetic algorithm with a tabu search method. Tabu search was introduced by Glover (Glover, 1989) and the main objective of this method is to optimize the algorithm by guiding local heuristics in the search space.

Single Large Object Placement Problem (SLOPP) is a particularization of a Cutting & Packing problem where the main goal is to place small objects on a largest object (container), leaving as little free space as possible. SLOPP objects have fixed dimensions (width and depth) and the First Fit heuristic (Berkey & Wang, 1987) is one of the first approaches to solve it. Improved First Fit heuristic for SLOPP implemented using a genetic algorithm (Pál, 2006), only allows in the same row, rectangles with the same depth as the first one or smaller. Those rectangles have fixed orientation (length and width) and may have a 90º rotation.

Two-dimensional strip packing problem (2D-SPP) is a specific case of strip packing where a set of rectangles with one open dimension (the rectangles have a fixed width but a variable depth) should be inserted into one container without overlaps and in a way that the strip is minimized. Ayala-ramirez et al. (Ayala-ramirez et al., 2005) defined for each figure position coordinates and rotation based on XOY graph. Solving 2D-SPP with genetic algorithms is also feasible as shown in (Mancapa, van Niekerk, & Hua, 2009). The authors introduced a new encoding that uses a set of 3-tuples to represent important details (position, identification and orientation) of the item to be placed. A survey on the heuristics to solve 2D-SPP problems was presented in 2016 by Oliveira et al. (Oliveira et al., 2016). One of the most common positioning-based heuristics is called "Bottom-Left" heuristic and was proposed by Baker et al. (Baker, Coffman, Jr., & Rivest, 1980). The usage of hybrid meta-heuristics (genetic algorithm, simulated annealing and naïve evolution) is accepted to solve strip packing problems since they tend to present better results (Hopper & Turton, 1999). A review of the meta-heuristics algorithms to solve 2D-SPP was presented by Hopper et al. (Hopper & Turton, 2001).

For the aim of this work intends to address, there are special constraints that are imposed by the shape grammar rules that imply relative positioning of rooms that are not present in the more general approaches found in the related literature. This implies that different methods most be devised to answer the architectural impositions.

2.2. Automatic Layout Design

To generate layouts using evolutionary computation is a real challenge. Multiple constraints and hidden rules (rules that humans apply but don't think of usually) need to be placed in the algorithm. The authors of (Qian et al., 2016) demonstrate that a space station layout can be achieved using evolutionary algorithms. Due the huge complexity involved, the problem was decomposed in smaller sets using a tree structure. Each smaller set runs the EA separately, producing a component and merging all components at the end. A human interface was also built to control the layouts, avoid local optimums and accelerate the convergence of solutions.

Optimize the human interaction interface with the machines is also a problem that industry is facing. (Chen & Deng, 2011) shows that a genetic algorithm can optimize the layout of an oil driller rig console. In theory, and using some examples analysis, the algorithm can generate layout schemes quickly.

(Valenzuela & Wang, 2000) optimized a VLSI floorplan problem using a genetic algorithm, achieving important results when comparing with other approaches.

In 21st Century, mass production trend is changing towards a customized view of the constructions. (Oosterhuis, 2012) shows that this change also affects the architectural area or, to be more precise, the design expression and architectural complexity. When an architect starts a new project a set of design performances are intended: spatial, structural, lighting, acoustic and thermal performance. Such elements are continuously interacting, making the design process more complex. (Fasoulaki, 2007) demonstrate that the usage of EA, in particular GAs, is common in architecture and looked at as a necessity due to an increase of human needs in today's lifestyle. Placement of the communicating elements of a house (e.g. windows or doors) is also a complex task, with multiple conditions and possibilities. (Caldas & Norford, 2002) presents a GA that is capable of placing windows in a building following architectural design principles.

(L. Li, 2012) breaks a complex architectural problem into combinatorial and numerical problems using GA. The GA search of the solutions space allows the architects to explore and select the best solutions based on different criteria. High-rise buildings structure or form-finding design could also be optimized using GA (Curriculum, 2012). GAs are also able to generate solutions for urban planning based on urban design concepts (Celani, Beirão, Duarte, & Vaz, 2011).

2.3. Genetic Algorithms

GAs have been a case of study for several years and the optimization problems based on layouts are not new. One of the most explored problems about layouts is the Facility layout problem (Drira, Pierreval, & Hajri-Gabouj, 2007) that consists in the usage of an available area to place facilities having constraints in the fitness function. Kar Yan Tam also implemented a GA to solve a Facility layout problem by taking into account two important components: area and shape constraints (Kar Yan Tam, 1992). Building temporary site-level facilities is common in construction areas during projects. To solve such problems, a set of facilities is allocated into a set of areas having into consideration the requirements and facilities constraints (H. Li & Love, 1998; Mawdesley, Al-jibouri, & Yang, 2002) . Manufacturing problems have been also addressed and solved with GAs such as crossdocking operations of manufacturing plant (Hauser & Chung, 2006), loop

layout design in flexible manufacturing systems (Cheng & Gen, 1998) or cellular manufacturing design and layout (Wu, Chu, Wang, & Yan, 2007).

Chapter 3. Background

To enable construction flexibility for timely production, some European countries decided to use prefabrication of off-site houses. Prefabricated methods of production can save time, money and allow high-quality construction. This type of construction uses vertical and horizontal studs, which give a stable frame for the interior and exterior walls. Walls can be fabricated off-side and the whole building construction could be done in a couple of weeks. As one of those countries, Wood Mass-Customized houses can be bought by anyone using catalogues in countries such as Poland or Portugal. The customization of this type of houses is however very limited and only some details can be changed by the future owner. For the design of modular housing according to an architectural language of design a system based on shape grammars was proposed, in 2016, by Kwiecinski et al. (Kwiecinski et al., 2016). The authors' main goal was that of allowing future owners to build houses that really meet their needs and simultaneously obey the language of design (Stiny, 1980). Such grammar represents the Polish understanding of family houses, which should be detached houses with a rectangular floor plan and with only the garage sticking out of the rectangular perimeter (Figure 1). A house can be positioned either parallelly or perpendicularly to the access route. However, for this dissertation, the positioning will be assumed to be perpendicular to the access route.

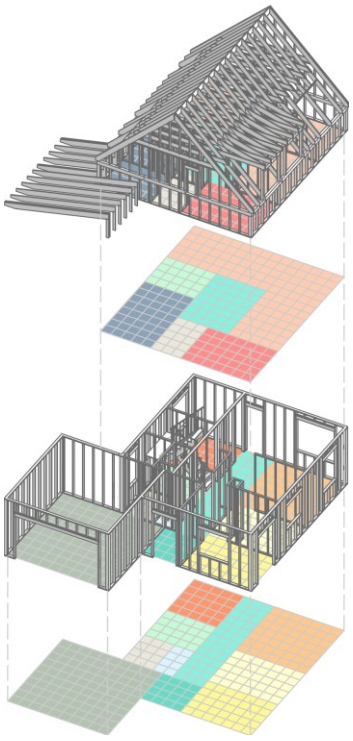

Figure 1 – Floor plan resulting from wood frame structural guidelines

3.1. A grammar for Polish modular homes - KSG

A Shape Grammar is a grammar with basic shapes as characters and where the rewriting rules define combination or substitution of shapes, may it be basic shapes or already combined ones.

The shape grammar created by Kwiecinski et al. (Kwiecinski et al., 2016), and which will be hereinafter designated as KSG, presents the shape rules divided in constraints for different stages of design (Figure 2): generation of the initial grid, generation of rooms belonging to the entrance zone, generation of rooms belonging to the semi-public zone and generation of rooms belonging to the private zone. The whole process is completed when all the required rooms are present, and all the conditions are met.

The initial process consists in collecting details from the future owner to formulate the house design brief. At this stage, either users or designers are asked to select the depth of each room between a minimum and a maximum size. In the following description, each rectangular grid represents a possible floor plan composed by multiple connected squares and where each square area is of $60\ x\ 60$ cm, which reflects the chosen sizes of the rooms.

This grammar can be used to generate house layout solutions based in the user's information and according to the architect's set of design rules. This process enables costumers to participate in the process by acquiring designs that meet their goals and requirements. As such, the grammar must provide more than one possible layout. Multiple layouts allow for the costumers to choose the one that he/she most likes and that better fits their needs. Since costumers' can compare the solutions, this comparison might lead to new input requirements due a change of mind.

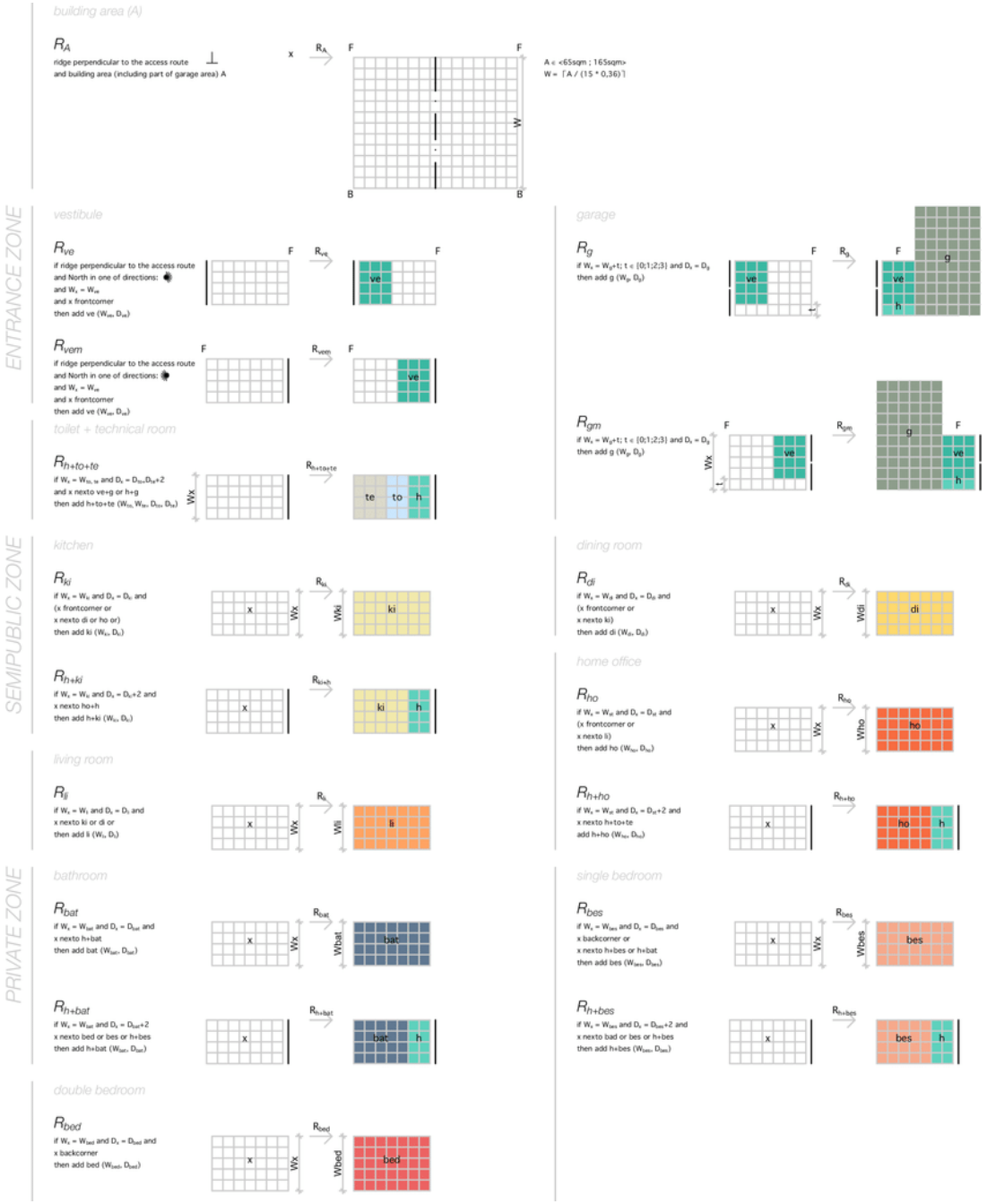

Figure 2 – An illustration of KSG shape rules.

3.2. Layout typology

At beginning, all shape rules from KSG were transformed into constraints to be incorporated into the fitness function of the GA, process that occurred in collaboration with the architect, K. Kwiecinski. This transformation was mandatory since the GA can't interpret the shape rules as such.

In the computational approach, the first step consisted in deciding which of the architects possible floor plans should be used as the goal. At the time, and because another procedural (iterative) approach was to be experimented with in parallel, it was decided that one store house with one master (double) bedroom and two single bedrooms was to be achieved.

For implementing the correspondent architectural rules imposed by the architect, 3 main areas were defined:

- Entrance - that contains the vestibule, a technical room, a garage and a toilet;
- Public area - containing the kitchen, dining room and living room;
- Private area - containing a double bedroom, two single bedrooms and bathroom.

The relative positioning rules for these areas and for the house rooms are the following:

- The public area must be placed at the front end of the house while the private area should be placed at the back.
- Each room must obey the following rules:
  - The vestibule must be placed next to the garage;
  - The toilet and technical room must be placed next to the garage;
  - The kitchen must be placed either at the front of the house or next to the living room;
  - The dining room must be placed at front or next to the kitchen;
  - The living room must be placed next to the kitchen or dining room;
  - The single bedroom must be placed next to another single bedroom or a bathroom;
  - The bathroom is placed next to a single bedroom;

- The double bedroom must be placed at the back.

The tree in Figure 3 shows a possible sequence for a backtracking step-by-step procedure based on KSG (Kwiecinski et al., 2016). The tree considers a $17x14$ modular house with the following rooms and dimensions (fixed): vestibule, technical room and storage - 3 × 3v, garage - 9 × 6, toilet - 3 × 2, kitchen - 2 × 7, dining room - 4 × 7, double bedroom - 5 × 7 and single bedroom - 4 × 7, living room - 5 × 7 - and bathroom - 3 × 5.

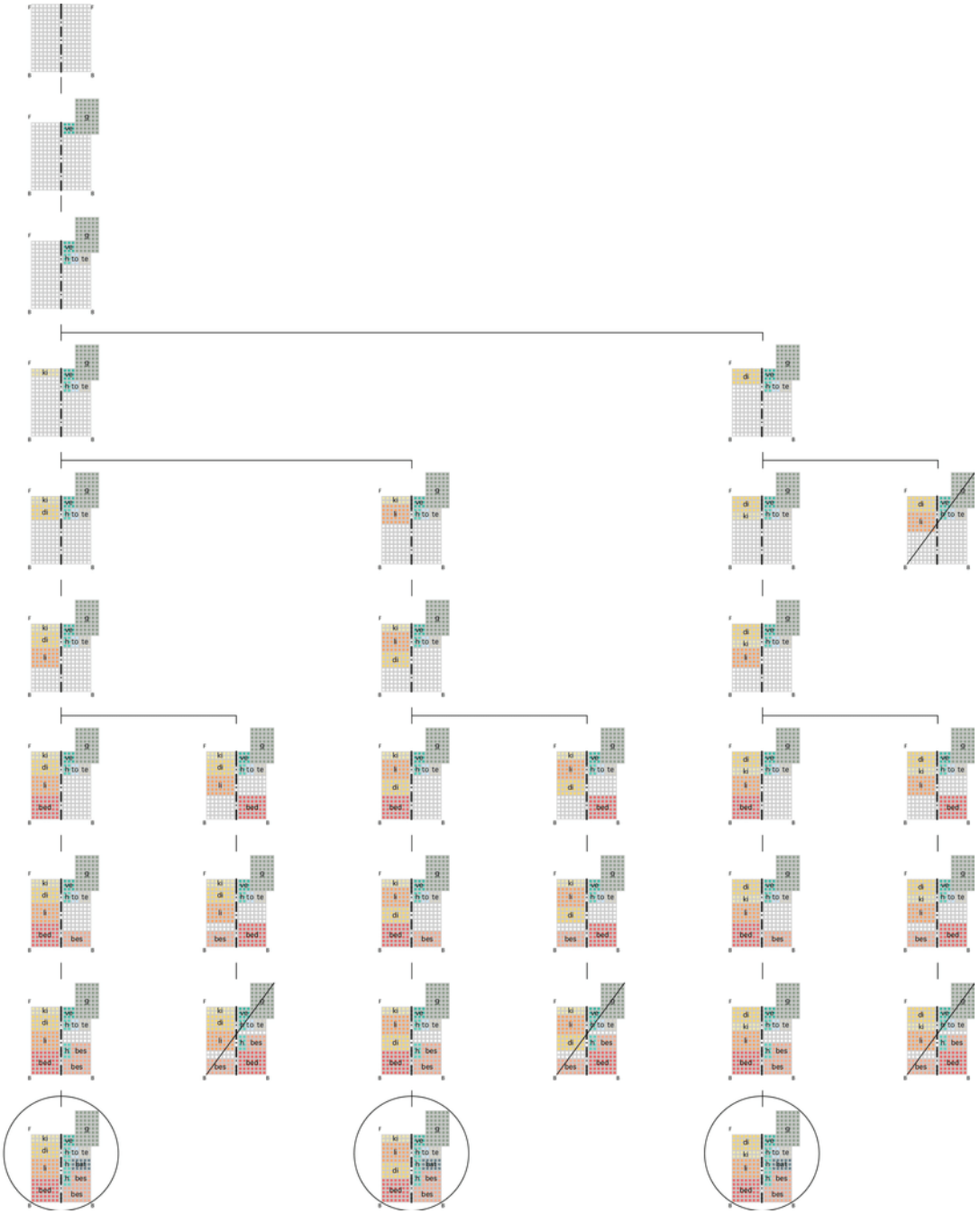

Figure 3 - Example of generation process of house layout design

3.3. Genetic Algorithms

Evolutionary Algorithms (EA) are a subset of Evolutionary Computation (EC) and are inspired in biological evolutions. An EA embodies mechanisms such as reproduction, selection, mutation, recombination and a fitness function. The fitness function has one of the most important roles in an EA since it defines the problem solution as the solution for an optimization problem and is the key element for guiding the evolution.

These algorithms can be used to solve either single-objective problem or multi-objective problems (*Evolutionary Algorithms for Solving Multi-Objective Problems*, 2007), but are especially renowned for its capability to solve multi-criteria hard problems.

John Holland proposed Genetic Algorithms (GA) as a paradigmatic method to tackle computational complex search spaces (J.H. Holland, 1995). One of the most important operators in GA is the fitness function that represents how adapted an individual is to the environment or the quality of each individual for a given target. Godfrey concluded, in 2014, that population size and crossover have high impacts in the goodness of the solutions (solutions closer to the objective) comparing with other operators if there is a constrained time environment (Godfrey, 2014).

A solution (for a given search problem) is a chromosome (or individual) – a binary or real valued string - and a set of chromosomes is called population.

The classic approach for genetic algorithms (Beyer, Beyer, Schwefel, & Schwefel, 2002) is described by Algorithm 1.

Algorithm 1 - Classic genetic algorithm

```
Randomly generate an initial population
Select k best fitted individuals
While stop criterion not met:
    Select parents for reproduction
    Crossover
    Mutation
    Select k individuals for a new generation
```

A large size of individuals in the population may cause large generational chromosomic diversity and consequently a higher rate of exploration of the search space. This in turn might increase the probability of finding an optimal solution.

The "natural" process leads to a higher probability of reproduction between higher adapted individuals, which should be able to reproduce. Such a process is achieved by analogy using a fitness function and selection and crossover operators. Selection can occur in two distinct steps: in the selection of individuals to reproduce - parental selection - and for the selection of a new evolutionary population.

As for crossover, there are several types recombination operators but the most common ones are: *single-point*, *multi-point* or *random respectful crossover* (RRC) (Goldberg & E., 1989; Radcliffe, 1991).

Mutation is another of the key operators in a GA scheme, since it explores the emergence of solutions and the exploration of the search space. The probability of an individual suffering a mutation has a direct impact in the performance of the GA. An interesting comparative study is that of (Cazacu, 2017) where three mutation operators are tested. The operators differ in the probability distribution being used: Uniform, Polynomial and Gaussian. Note that the probability of a chromosome suffering mutation should be independent of the probability to select a gene to be mutated. According to (Cazacu, 2017), when the probability of an individual being mutated is around 20%, more accurate solutions are found. The author concludes that the mutation operators tested have the same performance for structural problems. The best solution would be an evolutionary schema with the mutation probability going to zero at the last generations (Cazacu, 2017).

"Exploration and exploitation are the two cornerstones of problem solving by search." (Črepinšek, Liu, & Mernik, 2013). The process of visiting a new region of the search space looking for new solutions is called exploration. Exploitation is the process of visiting the regions found by the exploration and try to optimize the solutions comparing the neighbors. Thus, exploration is connected with crossover and mutation operators while exploitation is related with selection (Eiben, Eiben, & Schippers, 1998). A good ratio between exploitation and exploration will make the algorithm successful in advancing towards the objectives (Goldberg & E., 1989; Michalewicz, 1996).

Chapter 4. G-Shaper: A genetic algorithm application for automatic layout design of modular residential homes

The first GA attempt to produce an automatic layout based on the language of design defined was presented in (Almeida et al., 2016), with an approach called G-Shaper that implements the genetic algorithm described previously (Algorithm 1). In order to tune the GA approach, several operators were tested (e.g. different crossover techniques and selection mechanisms) to reach one optimal solution, a house layout, that not only obeys the design rules but also presenting the best performance possible.

Once it was decided to have the rooms' dimensions fixed, the problem to be solved is that of the placement of rectangular objects (rooms) within an overall rectangular area while optimizing a given function (fitness). Since the rectangles' orientations and their dimensions are fixed, the formal definition of the problem to be solved is the two-dimensional single large object placement problem (2SLOPP) (Gerhard Wäscher, Heike Haußner, 2007) obeying further relative positioning restrictions.

The search space for a $d \times w$ rectangular house with R rooms, each with dimensions $d_i \times w_i$ $(i = 1,2, \dots, R)$, is given by

$$\prod_{i=1}^{R}(w - w_i)(d - d_i) \qquad (1)$$

Thus, the search space is combinatorial in nature and, in fact, the general SLOPP is a Cutting and Packing NP-hard problem.

In order to tackle this challenge, a genetic algorithm strategy to deliver feasible layout designs (designs that obey both the typological specification, as well as the shaper rules) will be next described. This chapter begins by describing the encoding used to represent the house's layout, followed by a detailed overview of the operators and experiments performed. The chapter ends with the presentation and discussion on the results.

4.1. Chromosome and Gene representation

In Cutting and Packing existing genetic algorithm approaches for placement of rectangular areas, we can find different proposals for chromosome encodings:

- (Kröger, 1995) represents the layout's guillotine cuts (i.e. cuts that are orthogonal to the rectangle's sides dividing the rectangle in two smaller rectangles) by using a tree structure. Each tree node represents a rectangle that

encloses sub rectangles to be packed. The leaves represent the basic rectangular components (Figure 4).

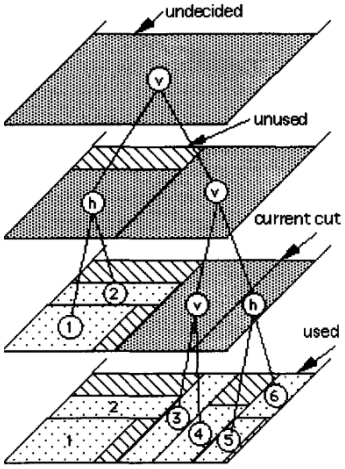

Figure 4 - Guillotine chromosome encoding (from Kröger, 1995)

- A different approach is the sequential rectangle's positioning, represented using the permutation of the identifiers for each of the rectangles (Hadjiconstantinou & Iori, 2007). The permutation represents the order in which the items must be packed.

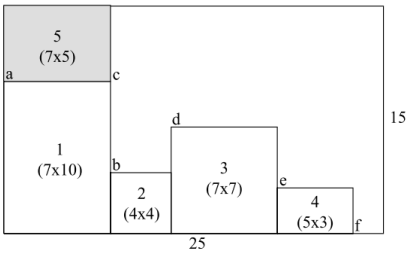

Figure 5 - Rectangle's positioning chromosome encoding (from Hadjiconstantinou & Iori, 2007)

- (Michalek, Choudhary, & Papalambros, 2002) use a grid representation for the entire layout by defining the layout area as a set of grid squares and using an algorithm to allocate each square to a particular room.

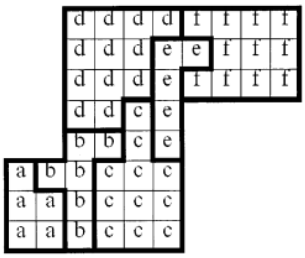

Figure 6 - Example of fix grid layout from (Michalek et al., 2002)

Based on (Michalek et al., 2002), a new encoding is proposed that, although closely related to the one described by Michalek et al., instead of representing the entire layout the grid is used to represent each the position of the chromosome gene (the respective room) in the layout (Figure 7). The entire layout for the house is represented by a $d \times w$ rectangular binary grid, where d stands for the depth and w the width of the layout. Each house has a predefined number of rooms, R, and each room, r_i ($i = 1, \dots, R$), has predefined dimensions $d_{r_i} \times w_{r_i}$.

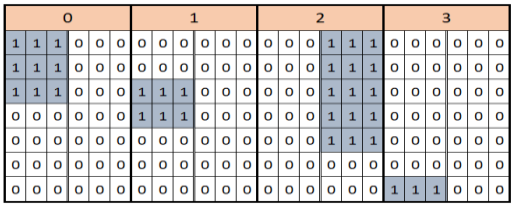

Figure 7 - Chromosome representation

A chromosome X is an array (Figure 7) that represents one modular house with multiple genes, r_i. Each gene is represented by a binary grid with all positions of the house, thus, the representation of each room is its $d_{r_i} \times w_{r_i}$ number of cells in the grid layout that are occupied. An occupied position is marked by 1 on the binary grid (Figure 7). The first position assigned for each room is the top-left position. Based on that position, the remaining positions are filled based on the given width, w_{r_i}. and depth, d_{r_i}, which depend on the type of room.

4.2. Initial and first evolutionary population

Assigning the "correct" size for the evolutionary population is not easy but is somewhat critical: if the population is too small size it may fail to converge quickly into an optimal solution, increasing the number of iterations (time) needed to stop the evolution.

An initial population with P randomly generated chromosomes (individuals) is created using an Uniform distribution (Goldberg & E., 1989). The random generation means that only the spatial positioning of each room on the grid is randomly generated, because of the predefined dimensions d_{r_i} x w_{r_i} are fixed. However, since every house must have its rooms on each side of a central axis (to allow for a central communication corridor), a room's positioning is randomly generated either at left or at right of the central axis.

The 10% most fitted chromosomes of the initial population go into the first evolutionary population, that is, the population that is actively going to evolve with the GA is composed by $\frac{P}{10}$ chromosomes.

In the human population, a DNA combination is unique for each human on the planet. G-Shaper also implements a mechanism that doesn't allow for equal chromosomes in either of the populations. This unicity mechanism evaluates and compares each new chromosome with the remaining existing population gene by gene, preventing duplicated chromosomes to join the population. If a chromosome is a duplicate, a new one is randomly generated.

4.3. Fitness function

The fitness function is responsible for measuring the adequacy of individuals (chromosomes) to reach the expected solution in optimization problems. In G-Shaper, the fitness function measures the adequacy of the individual to the architectural rules to define a desired floor plan (problem solution). This function is intended to be maximized and embodies the set of shaper rules defined for the layout.

The GA needs to evolve chromosomes into, at least, one feasible layout, that is, one that complies with all the design rules and there is no overlap between rooms positioning, which will be looked at as an optimal solution, or a solution with maximum fitness value. As such, it was decided that the failure to obey a rule or the existence of overlaps should be used a penalization towards the optimal value.

To define a quantifiable fitness function, several tests were made using different penalization implemented as negative quantities in order to maximize fitness, whose optimal value would be zero. For that, the design rules presented in Section 3.2 were implemented by turning them into decision statements. As an example, take the rule: "the double bedroom needs to be placed at the back of the house". If the double bedroom is not at the back, -x penalty points were added to the total penalization of the house i. However, the approach was failing to converge into a maximum, and a new schema was implemented. The final fitness function is calculated by the next described process.

An optimal solution will have a value of one, meaning that all shaper rules are obeyed and there are no overlaps between rooms. Any violation of a rule $j, j = 1, \dots, 9$, from (Section 3.2) incurs in a penalty weight of 100. The other important penalization is the

existence of overlaps between rooms' positioning. Since this is highly undesirable a penalization of 100 points per unit grid overlapped was imposed. As an example, in Figure 8, the blue and red rooms have both 3 x 3 side dimensions. The orange squares represent 3 overlapped grid units thus adding a 300 (3 x 100) points penalty.

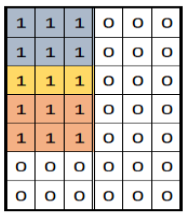

Figure 8 - Two rooms that overlap.

Let $H = \{h_i \mid i = 1, \dots, N\}$ be the evolutionary population (each h_i is a chromosom) and $p_j(h_i)$ the penalization of j^{th} shaper rule of house h_i, and $o(h_i,)$ the total of overlap penalties. A penalty function is defined as:

$$P(h_i) = \sum_{j=0}^{J} p_j(h_i) + o(h_i) \qquad (2)$$

The fitness function is defined by $F: H \rightarrow]0,1]$ such that,

$$F(h_i) = \begin{cases} \frac{1}{P(h_i)}, & P(h_i) \neq 0 \\ 1, & P(h_i) = 0 \end{cases}. \qquad (3)$$

4.4. Selection

As previously stated, G-Shaper uses two different type of selection operators: the parental selection for recombination and offspring generation, and the selection of individuals for the new evolutionary population.

4.4.1. Parental selection

This step is responsible for choosing two parents from the current population for use in a crossover. By using a *uniform* distribution, every chromosome has equal probability of being selected – *uniform selection*. This is the simplest approach towards this mechanism and was the first one implemented in G-Shaper, mostly because it's a technique to prevent selection from being too much biased towards higher fitness chromosomes.

Nevertheless, *The Theory Of Evolution* by Charles Darwin (Darwin, 2003) states that the most adapted individuals have higher probability of being selected for the crossover. Thus, the second choice was that of selecting individuals using *ranking*, that is, the

individuals are sorted by their fitness value. The most fitted individuals will can thus be chosen for reproducing – *ranking selection*.

Still following the principles of Darwin's Theory, *Roulette Wheel Selection* (RWS) (Lipowski & Lipowska, 2012) was chosen as another classic parental selection operator. RWS is based on the concept that the probability of an individual i having $F(i)$ as the fitness function value to be selected for crossover in a population with N individuals is described by:

$$P_{select}(i) = \frac{F(i)}{\sum_{j=0}^{N} F(j)},\ i \in \{1,2,..,N\}. \qquad (4)$$

4.4.2. Evolutionary selection

The evolutionary selection is responsible to select the individuals towards the new evolutionary population. In the following, the individuals from the current population (that is, not the new offspring) will be generally called by parents.

One possible approach is to select N chromosomes that will survive and make part of the new evolutionary population can be decided by *ranking,* where the N most fitted individuals from the set of parents plus offspring are chosen. *Elitism* is another mechanism that copies half of the most fitted parents ($N/2$) and half of the most fitted offspring ($N/2$) towards the new evolutionary population. A hybrid method (e*litism&ranking)* moves the 10% most fitted parents directly into the new population. The remaining individuals are chosen from more fitted ones between the parents that were left and the offspring.

Algorithm 2 - Single-point crossover generating children $\boldsymbol{C}[\boldsymbol{c_1}, \dots, \boldsymbol{c_R}]$ *and* $\boldsymbol{D}[\boldsymbol{d_1}, \dots, \boldsymbol{d_R}]$

1: Select parents $A[a_1, \dots, a_R]$ and $B[b_1, \dots, b_R]$;
2: Randomly choose 1 crossover point $[cp_1]$;
3: **for** $i \leftarrow 0,\ i < R,\ i \leftarrow i + 1$ **do**
4: **if** $i < cp_1$ **then**
5: $c_i \leftarrow a_i$;
6: $d_i \leftarrow b_i$;
7: **else**
8: $c_i \leftarrow b_i$;
9: $d_i \leftarrow a_i$;
10: **end if**
11: **end for**

4.5. Crossover

The crossover main's objective is to generate offspring by combining genes from selected parents, trying to improve new individuals based on past ones. G-Shaper generates two offspring from two different selected parents, always making sure that the generated offspring is composed of unique chromosomes in the population, that is, since offspring can possibly end up with the same chromosome as another individual on the population, the mechanism to delete duplicated individuals (see 4.2) is also applied in crossover.

Different crossover algorithms were implemented to test G-Shaper. For each one, two parental chromosomes are selected, A and B, and two children are generated by recombination: C and D.

The first recombination technique is the so-called *Single-point* crossover. A one crossover point, cp_1, is randomly chosen between $\{1, \dots, R\}$, where R represents the number of rooms of the house. Offspring C receives A genes occurring before cp_1, and genes from B from then on and vice-versa for D (Algorithm 2***).***

Algorithm 3 - Multi-point crossover generating children $\boldsymbol{C[c_1, \dots, c_R]}$ *and* $\boldsymbol{D[d_1, \dots, d_R]}$

1: Select parents $A[a_1, \dots, a_R]$ and $B[b_1, \dots, b_R]$;
2: Pick a random $k \in \{1, 2, \dots, R-1\}$;
3: Randomly choose k crossover points $[cp_1, \dots, cp_k]$;
4: Fix $cp_0 \leftarrow 1$ and $cp_k + 1 \leftarrow r$;
5: $alt \leftarrow 0$;
6: $j \leftarrow 1$;
7: **for** $i \leftarrow 0,\ i < k+1,\ i \leftarrow i+1$ **do**
8: **while** $j < cp_i$ **do**
9: **if** $alt = 0$ **then**
10: $c_j \leftarrow a_j$;
11: $d_j \leftarrow b_j$;
12: **else**
13: $c_j \leftarrow b_j$;
14: $d_j \leftarrow a_j$;
15: **end if**
16: $j \leftarrow j + 1$;

17: **end while**
18: $alt \leftarrow 1 - alt$;
19: **end for**

The second implemented crossover is the *Multi-point* crossover, that randomly generates k crossover points between $\{1, \dots, R-1\}$ that is $\{cp_1, \dots, cp_k\}$. Between each crossover point, the parents will exchange the genes for the offspring according to Algorithm 3.

Algorithm 4- RRC crossover generating children $\boldsymbol{C}[\boldsymbol{c_1}, \dots, \boldsymbol{c_R}]$ *and* $\boldsymbol{D}[\boldsymbol{d_1}, \dots, \boldsymbol{d_R}]$

1: Select parents $A[a_1, \dots, a_R]$ and $B[b_1, \dots, b_R]$;
3: **for** $i \leftarrow 0,\ i < R,\ i \leftarrow i + 1$ **do**
4: **if** $a_i == b_i$ **then**
5: $c_i \leftarrow a_i$;
6: $d_i \leftarrow a_i$;
7: **else**
8: $c_i \leftarrow Generate\ new\ random\ position$;
9: $d_i \leftarrow Generate\ new\ random\ position$;
10: **end if**
11: **end for**

Finally, a third crossover technique was implemented, the *Random respectful crossover, RRC* (Algorithm 4). While the parents have the same gene RRC replicates it towards the offspring but, whenever it differs, two new genes are randomly created for each of the offspring.

With *single-point* and in *multi-point* crossovers, the search space is mostly exploited since the selected parent genes are kept, even if in different arrangements. However, RRC keeps most of the exploitation but allows some exploration of diversity.

As already mentioned, after the recombination for the generation of two new individuals, each child is evaluated by comparing its chromosome only with the remaining population with the same fitness value. If there is a duplication (i.e, the two houses are

equal), the offspring is ignored, not being allowed to join the remaining offspring population.

4.6. Mutation

Mutation is a mechanism that induces diversity on the population by changing the genes, allowing for some slack of exploration of the search space.

After crossover, offspring may suffer mutations. To establish that, a number is drawn using uniform probability. If it is found to be in the range $[0.001, 0.02]$, a random gene $G = \{g_i \mid i = 1, \dots, R\}$ is selected to be mutated. This operator changes the selected gene (g_i) by randomly generating a positioning of the room on the binary grid.

4.7. Experiments and Results

The following results were produced using a directly coded Java implementation of the previously defined GA and each of the operators. The results report to experiments with 30 individual runs from start. The following tests run on a virtual machine using Windows Server 2016 with 8GB ram, 4 virtual CPUs and Intel Xeon E312xx processor.

The first step to be executed on a new GA application is to select the most adequate operators and perform parameter tuning. G-Shaper was tested over the different combinations of selection and crossover operators. The evolution stops when a first optimal house is found.

After exhaustive experimentations, the operators that showed better performance as selection operators are:

- *Roulette Wheel Selection* for the parent selection operator;
- *Ranking* for the new evolutionary population selection.

4.7.1. Crossover and evolutionary population study

The following tests were executed to test the different crossover operators and the influence of the size of the evolutionary population. As previously mentioned, the selection operators are *RWS* for parenthood and *ranking* for the selection of a new evolutionary population. G-Shaper stops after finding one optimal house but it takes a considerable number of cycles to do so. In order to understand if some chromosomic specialization might be occurring, it was decided to explore the population diversity using the average fitness and not the maximum average of the population.

As it can be seen on Figure 9, the average fitness for 100 and 200 individuals in all operators is increasing with the number of iterations. In terms of average fitness, RRC is always surpassed by *multi-point* and *single-point* during the iterations. *Multi-point* (200) crossover consistently achieves higher values over the remaining operators. Interesting is also the fact that multi-point (100) follows from very close.

It's possible to identify that *RRC* has a notorious difference in the average fitness compared with the remaining operators. This could be explained by the fact that *RRC* creates more diversity in the population, reducing the specialization of the individuals but also helping to reduce the possibility of genetic drift. Thus, these results were expected and, for *RRC* with 200 individuals in the population, the diversity is higher, making the average of the population fitness achieve lower values.

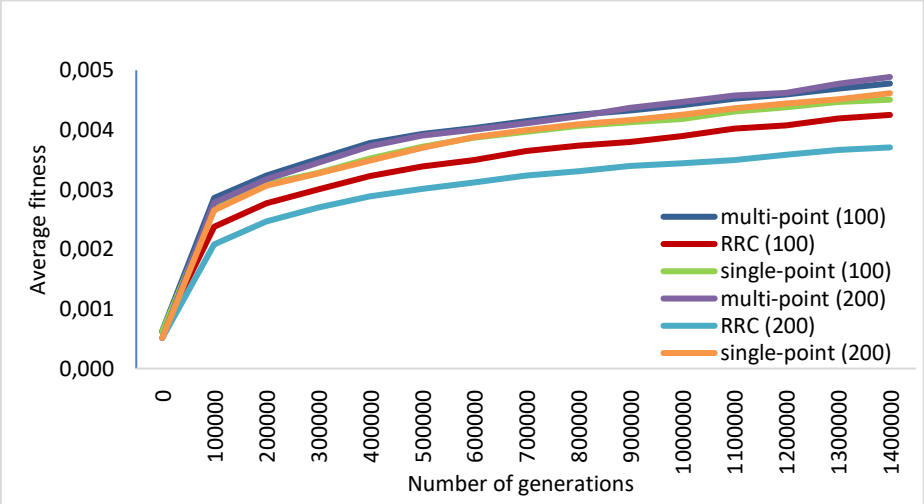

Figure 9 - Average fitness of populations for different crossover operators and population sizes.

4.7.2. *Particular results for the e*volutionary population size

As previously seen, the evolutionary population dimension influences the average fitness values but also impacts the number of generations needed to terminate the evolution. Figure 10 shows the average number of generations needed to achieve the termination criterium, i.e. the first optimal house, while Figure 11 compares the population both size and the average population fitness.

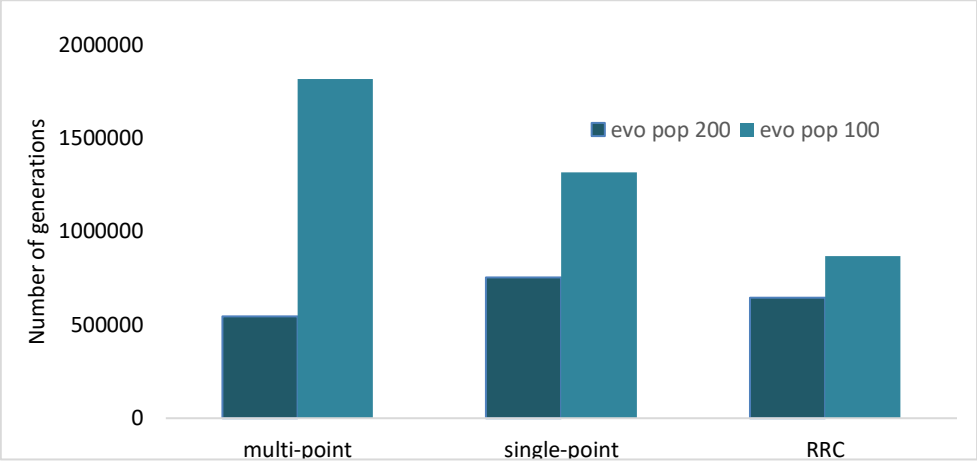

Figure 10 - Average number of iterations per crossover operators.

Regarding the average number of iterations, it's possible to see that the change between 100 to 200 individuals has an important effect over all crossover operators iterations. The crossover operator that shows more impact by changing the number of individuals is *multi-point* crossover. Comparing *RRC* and *multi-point*, it's possible to verify that *RRC (100)* was able to achieve the objective with less generations than *multi-point (110),* even consistently presenting lower average fitness values in its populations. The diversity present on *RRC* populations, while making the individuals in average lesser fitted, allows to reach one optimal solution much faster.

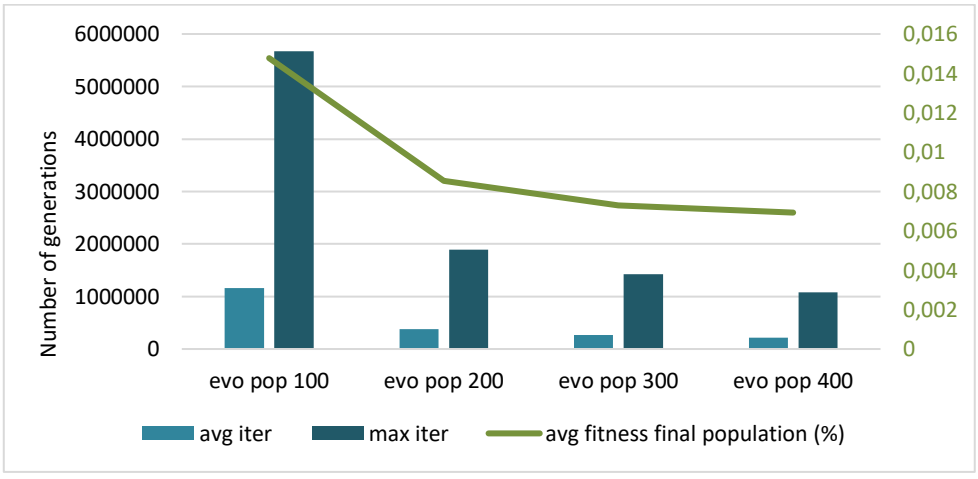

Figure 11 - Population's sizes vs average population fitness

As the population increases, less iterations are needed to find the optimal house (Figure 11). An interesting comparison is to verify the number of iterations with the average fitness

values at the final population. With the increase of the population the average fitness decrease, which serves to confirm that the higher population size, the wider is the diversity.

Note that the number of maximum iterations seems to decrease polynomially (($y \approx -5.31 \times 10^6 x^3 + 4.84 \times 10^7 x^2 - 1.46 \times 10^7 x + 1.56 \times 10^7$)) with the increase of population chromosomes, as shown in Figure 12.

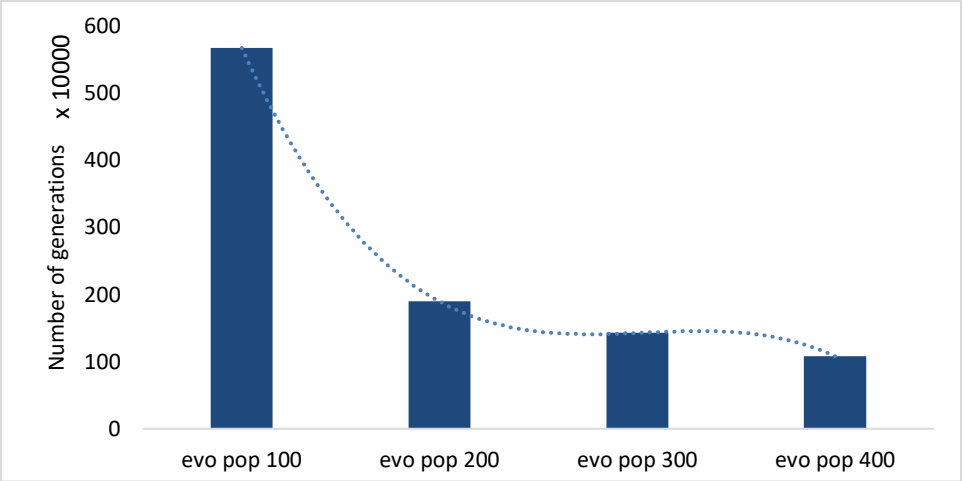

Figure 12 - Maximum number of iterations reduction

4.8. Discussion

At this point, it is possible to say that the first research question RQ 1 is positively answered: a GA approach is a viable approach to produce automatic layout designs for modular houses compliant with architectural rules of design.

The genetic algorithm approach presented can generate a feasible layout is a matter of close to 20 minutes. Moreover, the balance between average fitness values and total number of iterations makes *RRC* as the better crossover choice for following experiments.

One of the main characteristics of this approach is that G-Shaper, although being capable of producing a solution for this real-world problem, this was a simple design and thus, the approach still presents some disadvantages. In order to allow for scalability, it would be desirable to reduce the time to generate the layout. On another note, looking at the average growth of the population average fitness values, G-Shaper, after a quick improvement, still presents a tendency to stabilize, which may incur in genetic drift. Thus, a mechanism to enforce chromosomic diversity should be further explored.

Finally, from the user's perspective, to receive only one solution is not the most user-friendly and does not allow for an active choosing of the favorite house from a set of possible solutions. Thus, the layout design system should able to achieve multiple solutions and display them to the final users to help the decision.

Chapter 5. Shaper-GA: Automatic shape generation for modular house design

Shaper-GA, that is described in (Taborda, de Almeida, Santos, Eloy, & Kwiecinski, 2018), is the proposed "evolution" of G-Shaper. Shaper-GA can produce multiple optimal solutions based on the architectural rules of Kwiecinski et al. (Kwiecinski et al., 2016). The following sections describes the changes that were made to G-Shaper.

The first change that had to be made consists in achieving different final house layouts. Moreover, the language of design considers the possibility of, for the same overall area of layout, allowing variable depths per room type. The fact that variable depths are to be considered, the problem formulation turns from a 2SLOPP into a two-dimensional strip packing problem (2D-SPP) (Bortfeldt, 2006) (Thomas, 2013) where the objects have fixed width and variable depth.

The next sections describe a new gene encoding and optimization procedures that lead to the new GA approach, Shaper-GA.

5.1. Chromosome and Gene representation

The first change performed intended to optimize the time performance of G-Shaper (Section 4.1) and consisted in changing the gene representation.

Similarly of the previous approach, the chromosome encoding on Shaper-GA still is represented by an array X with R genes. Each house still (h_i) has fixed width (w) and depth (d) and can be visualized as an abstract rectangular binary grid with $w \times d$ cells. However, for the gene representation, instead of a rectangular binary grid, in Shaper-GA relative positioning (coordinates) are used instead. With this representation change, the computation time in all genetic algorithm operators was expected to decrease. The encoding of one room can be described as:

Table 1 - Room i (gene) encoding

Parameter	Description
Name	Name of room (e.g.: living room)
x	1 or $\frac{w}{2} + 1$
y	$\in [1, d - d_i]$
Width	w_i (fixed)
Depth	$d_i \in [m_i, M_i]$ (predefined range)

In Table 1, x and y are the grid cells coordinates of the up-left vertex of the room (rectangular shape), representing the initial point of positioning for room i. Since the rooms cannot cross the central axis (that divides the house width in two equal sub-rectangles), rooms are always placed on the left side of each of the sub-rectangles ($x = 1\ or\ x = \frac{w}{2} + 1$). The rooms' width, w_i, and depth, d_i, are also used to calculate the area (A_i) occupied by room i: $A_i = w_i\ x\ d_i$, i.e., to identify which grid units are used for the rooms positioning (important for the search of overlaps) .

5.2. Speed-up of search space exploration

The results of G-Shaper (Section 4.7, Figure 9) show that average fitness evolution was somewhat "slow" and possible tendency for genetic drift should be avoided. To improve that situation, an optimizer was created, based on an adjustable mutation rate and thus increase the diversity of the population (Thierens, 2002) . Such approach should be used only in specific occasions, where it is possible to perceive that the value in the average fitness of population has not been changing, situation that was found to happen with G-Shaper.

Having G as current generation, if the average fitness of population doesn't increase at least 0.5% between generations $G - 1$ and G, the optimizer duplicates the mutation rate. Note that one must bound the maximum possible mutation rate, otherwise there is some probability of overdoing it and creating an all-new initial population, situation that is not desired. Shaper-GA maximum mutation rate was set to 20% (Cazacu, 2017). Once the average fitness of population increases, the mutation rate goes back to 2%.

5.3. Generation of several optimal solutions

One of the most notorious differences between G-Shaper and Shaper-GA is the capability to produce several optimal houses, that is, different house layouts. Different houses mean that the internal composition of the house differs, since the layout area is unchangeable. The rooms can be placed in different positions (structural composition) or can change their depths (parametric composition).

Structural composition is directly implemented in the fitness function by the shape rules to be considered. The analytic exploration of the KSG rules using fixed room's dimensions only showed three different possible layouts for the implemented architectural rules

(Figure 3). To provide a variety of solutions to final users, each of the possible layouts can also be generated with different rooms' depths (parametric composition). To implement the parametric composition several steps were implemented.

Upon consulting the architect, K. Kwiecinski, possible rooms dimensions for the example being used were provided (Table 2) in terms of the range of possible values (integer number of grid units).

Table 2 - Depth (in grid units) for each room

Room *i*	Minimum depth: m_i	Maximum depth: M_i
Kitchen	2	8
Living room	4	9
Dining room	4	6
Double bedroom	5	7
Single bedroom	4	7
Bathroom	3	5

In order to make the rooms depth a variable dimension, that is, that evolution could incur in resizing some or all of the rooms, a *resize* function was needed. It was decided that, besides the obvious possibility of generating a room whose depth was also randomly drawn from the allowed range, resizing of a room could also occur (in probability) in two evolution points: in recombination or mutation operations. The resize function that was implemented is described by Algorithm 5, where d_i stands for the depth of room i:

Algorithm 5 - Resize function

1. Randomly generate $r \in [0,1]$;
2. If $r < 0{,}5$
3. If $d_i > m_i$ then $d_i \leftarrow d_i - 1$;
4. Else
5. If $d_i < M_i$ then $d_i \leftarrow d_i + 1$;

Towards the goal of generating several house layouts, Algorithm 5 is used for the mutation of a gene. When the gene is selected for a mutation one of two things may be

randomly decided: either to generate a new random position for the room, or to resize it. In the latter case, only the depth of the room will be changed (either increasing or decreasing the depth). Thus, the mutation operator has now two possibilities: the resize or the normal mutation, as described by *Algorithm* . Notice that, if an optimal solution is selected for mutation, only the resize option will be performed; otherwise both options have equal probability of being selected.

Algorithm 6 - Mutation operator for a room of house h_i

1. Randomly generate a value $c \in [0, 1]$;
2. If $0.001 \leq c \leq P$ $(P \in [0.02, 0.2])$
3. Pick a random room, d;
4. If $\boldsymbol{F}(\boldsymbol{h_i}) = 1.0$
5. Resize d;
6. Else
7. Randomly generate $r \in [0, 1]$;
8. If $r < 0{,}5$
9. Generate new position for d;
10. Else:
11. Resize d;

As previously mentioned, Shaper-GA uses *RRC* recombination. Thus, the resize function is also used in this crossover operator, namely when parents' genes are different. In G-Shaper, different parental genes in *RRC* random generate a new room's position for the offspring, whilst in Shaper-GA the rooms can be either positioned randomly or keep the same position and having its depth resized.

5.4. Experiments and Results

To explore this new approach, two different GA versions were implemented for Shaper-GA: *Standard* and *Resizable from beginning* (RfB) version.

For the initial population generation, the *standard* version generates each house assuming that all the rooms have predefined width and depth (standard dimensions), while in RfB, each room is generated by assigning a random depth in the allowed range for each type of room, $d_i \in \{m_i, M_i\}$ (Table 2).

Both Shaper-GA versions (standard and RfB) were implemented using Java. Each experiment consists in 30 independent runs. All the following tests run on a virtual machine using Windows Server 2016 with 8GB ram, 4 virtual CPUs and Intel Xeon E312xx processor.

Based on the previous work (section 4.7 and 4.8) experiments and results, the following parameterization was used to perform all the tests:

- Initial population: 1000 individuals
- Parent selection operator: RWS
- Crossover operator: RRC
- Next evolutionary population selection: elitism, ranking and elitism&ranking
- Stopping criterion: 5 optimal solutions

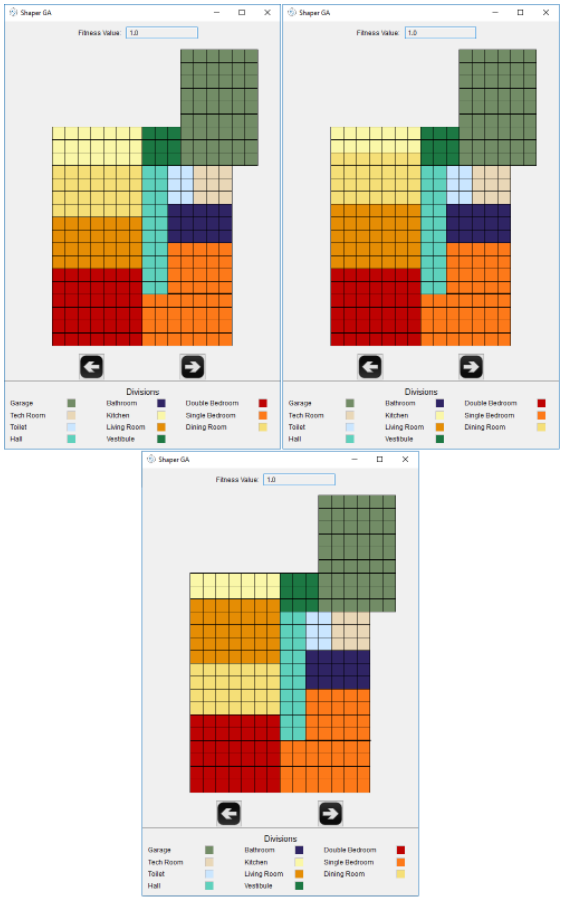

Figure 13 - Three different solutions from Shaper-GA

Figure 13 shows three different examples of the five optimal solutions produced by Shaper-GA. Each of the layouts in Figure 13 presents either different depths for the same

rooms or different positioning. Comparing the top-left and top-right layouts it's possible to see differences on the depths of the kitchen and the living room. The top-left layout presents 4×7 grid units for the living room and 3×7 grid units for the kitchen. The second layout (top-right) has 5×7 grid units on the living room and 2×7 grid units on the kitchen. Comparing both layouts, it's possible to say that they are equal on the rooms arrangement but not on rooms size. Note that the bottom layout of Figure 13 shows a different arrangement of rooms positioning comparing with the top layouts.

Comparing the performance for both versions of Shaper-GA concerning average fitness of the evolutionary populations and showing only the first 1 million generations for different selection schemes for the next evolutionary population (*ranking*, *elitism* and *elistism&ranking*), it is possible to say that, for the *standard* version, the performance of the different schemes is very similar (Figure 14). However, for the RfB version, and for the first 600 000 generations, *elitism* and *ranking* show higher average fitness than *elitism&ranking*. However, the latter shows higher average fitness results after 600 000 generations.

An interesting analysis comes from comparing directly the average fitness values for the *standard* and RfB versions using only *elitism&ranking* (Figure 14**)**. The *standard* version presents higher fitness values (0.203 approx.) than the RfB version (0.139 approx.). Again, that difference can be explained by a wider population diversity present in the evolutionary population for RfB. Thus, in *standard* version, the population is more adapted to the objective.

To understand population's diversity, *Figure 14* uses average fitness to evaluate both Shaper-GA versions. Comparing the average fitness for both versions with G-Shaper (Section 4.7) it is possible to conclude that populations' average fitness is considerably higher for Shaper-GA then for G-Shaper. Such results show that the mechanism for forcing higher search space exploration rates described in Section 5.2 is successful.

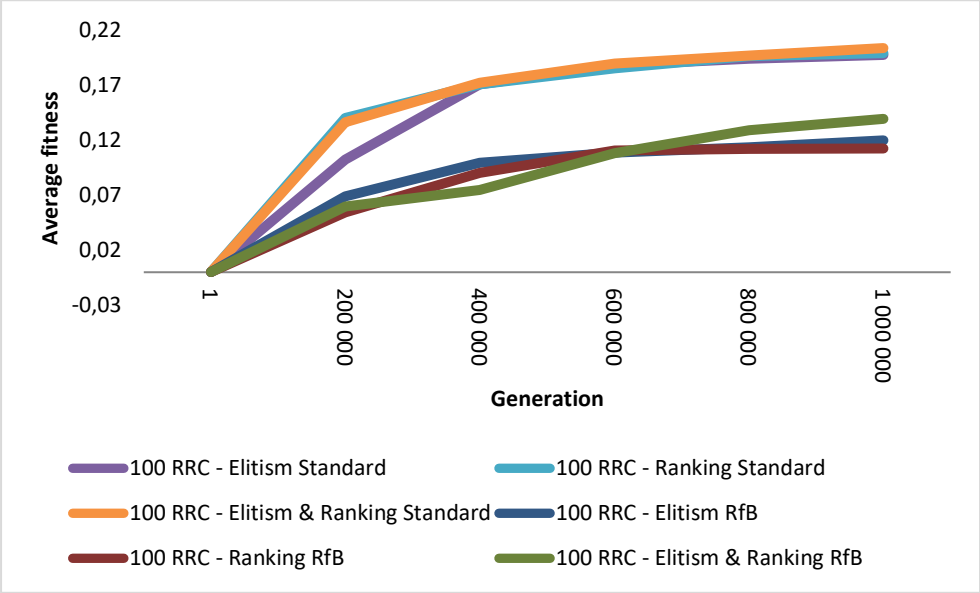

Figure 14 - Average fitness for Standard and RfB versions

One of the problems identified in G-Shaper was the execution time needed to get one optimal house. Figure 15 compares the average execution time per run between G-Shaper and both Shaper-GA versions, *standard* and RfB, until reaching 1 million iterations and using *elitism&ranking* as the selection operator. G-Shaper is only evolving until it achieves one optimal solution, while Shaper-GA only stops at five. Even with such difference, both versions of Shaper-GA outperformed G-Shaper, making the new structure of chromosomes and optimization schemes a feasible approach to increase and expand the performance of computation. The best results on running time are achieved using Shaper-GA *standard* version (approximately 2 minutes in average before algorithm termination).

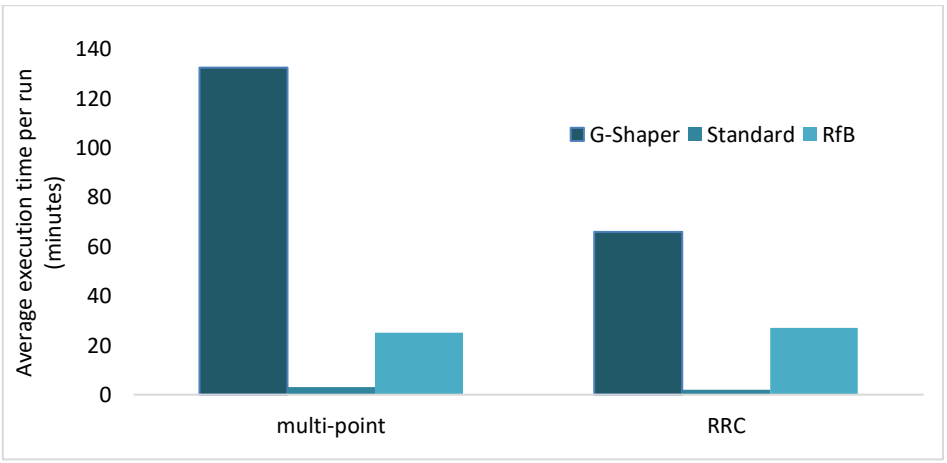

Figure 15 - Average execution time per run with different versions and crossovers

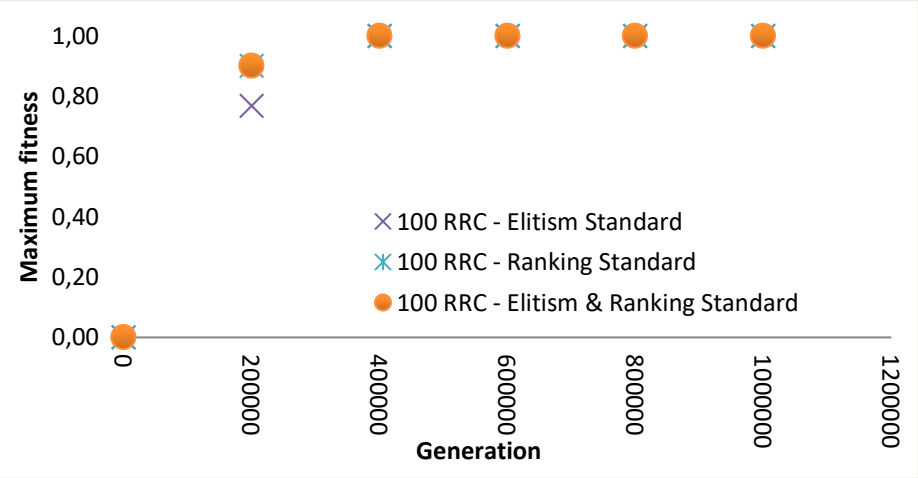

Figure 16 - Average maximum fitness - Standard version

Figure 16 represents the average of the maximum fitness over 30 runs for the *standard* version. It's possible to conclude that any of the operators is capable of producing at least one optimal solution at 400 000 generations. Those results are also linked with those pictured in *Figure 14*: it's possible to see that the average fitness of the population shows a higher slope until it reaches 400 000 generations, after which the slope is substantially decreased.

5.5. Discussion

The new room (gene) encoding together with the adjusting optimization scheme developed for Shaper-GA decreased more than 85% of the running time comparing with G-Shaper. The fastest G-Shaper runs, in average for approximately 65 minutes before it finds one optimal solution in the search space. With Shaper-GA *standard* version, five optimal solutions are generated in, approximately, 2 minutes, it is possible to say that the second research objective (RQ2) was answered positively.

The capability to achieve different layout solutions was also reached with both versions of Shaper-GA. Comparing *standard* and RfB versions, the former outperformed the latter both for running time, establishing the *standard* version as the best approach to produce several optimal solutions.

The last versions were implemented using dynamic room's dimensions. The third research question (RQ3): How to implement dynamic room's dimensions?, can now be replied in a positive way, with the creation of a resizable function that were used in *RRC* crossover and mutation operators.

Chapter 6. Conclusions

6.1. Main conclusions

The fact that future owners of modular houses were not able to customize their houses was a gap identified by Kwiecinski and Slyk (Kwiecinski & Slyk, 2014), that gave the first steps towards filling this gap by proposing a shape grammar to formalize an automated language of design (Kwiecinski et al., 2016). G-Shaper and Shaper-GA were GA systems created intending to contribute to close that gap in an automated fashion.

G-Shaper (Almeida et al., 2016), detailed in Chapter 4, was the first attempt of producing an automatic layout design tool for modular houses. Using a genetic algorithm, it was able to produce one feasible solution complying with the architectural rules of design suggested by the architect. G-Shaper answered positively to RQ1 (Is the Genetic Algorithm approach viable for generating effective house layouts, that is, obeying the architectural rules of design?) in a limited time (approx. 1 h). Experimentation results show that the usage of GA is a valid and promising approach for an automatic house layout design system but suffers from tendency to genetic drift.

Shaper-GA (Taborda et al., 2018), detailed in Chapter 5, is the evolution of G-Shaper to optimize house layout generation in two fronts: time and several layouts deliverance. This tool also presents a different genetic algorithm architecture to return more than one optimal layout solutions complying with the design rules. Shaper-GA is able to deliver different layouts in a matter of a few minutes. Also, the possibility to generate several optimal solutions was achieved using variable depths for rooms.

6.2. Future work

Shaper-GA can only be considered as a complete automatic house layout design system when the tool is being able to place components (e.g. doors, windows) following the architectural language of design. The current work uses only one architectural language of design (Kwiecinski et al., 2016) but to have a robust solution for the enterprise market, multiple architectural perspectives should be tested and validated. Also, this tool can easily be integrated in a website using a richer interface to reach multiple users in different sides of the world.

As a pervasive important objective for the future is the possibility of emergence of design, that can be scanned since it presents a most wanted goal search for architects.

Multiple variables are not being taken under consideration (i.e. exposure to sun) or orientation towards the road access. A generic and complete framework should be created to accommodate all of those inputs

References

Almeida, A. De, Taborda, B., Santos, F., Kwiecinski, K., & Eloy, S. (2016). A genetic algorithm application for automatic layout design of modular residential homes. *Proceedings of the 2016 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2774–2778. Retrieved from https://ciencia.iscte-iul.pt/export/pub/30239

Ayala-ramirez, V., Ponce-Pérez, A., Perez-Garcia, A., Pérez-Garcia, A., Ponce-p, A., & Arturo, P. (2005). Bin-Packing Using Genetic Algorithms. *Electronics, Communications, and Computers, International Conference On*, *0*(Conielecomp), 311–314. https://doi.org/http://doi.ieeecomputersociety.org/10.1109/CONIEL.2005.25

Baker, B. S., Coffman, Jr., E. G., & Rivest, R. L. (1980). Orthogonal Packings in Two Dimensions. *SIAM Journal on Computing*, *9*(4), 846–855. https://doi.org/10.1137/0209064

Berkey, J. O., & Wang, P. Y. (1987). Two-Dimensional Finite Bin-Packing Algorithms. *Journal of the Operational Research Society*, *38*(5), 423–429. https://doi.org/10.1057/jors.1987.70

Beyer, H.-G., Beyer, H.-G., Schwefel, H.-P., & Schwefel, H.-P. (2002). Evolution strategies – A comprehensive introduction. *Natural Computing*, *1*(1), 3–52. https://doi.org/10.1023/A:1015059928466

Bortfeldt, A. (2006). A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research*, *172*(3), 814–837. https://doi.org/10.1016/j.ejor.2004.11.016

Caldas, L. G., & Norford, L. K. (2002). A design optimization tool based on a genetic algorithm. *Automation in Construction*, *11*(2), 173–184. https://doi.org/10.1016/S0926-5805(00)00096-0

Cazacu, R. (2017). Comparative Study between the Improved Implementation of 3 Classic Mutation Operators for Genetic Algorithms. *Procedia Engineering*, *181*, 634–640. https://doi.org/10.1016/j.proeng.2017.02.444

Celani, G., Beirão, J., Duarte, J. P., & Vaz, C. (2011). Optimizing the "characteristic structure": Combining shape grammars and genetic algorithms to generate urban patterns. *ECAADe 29*, 491–500.

Chen, B., & Deng, L. (2011). The Application of Genetic Algorithm in Layout Design of Oil Rig Driller Console. *Advanced Materials Research*, *216*, 171–175.

https://doi.org/10.4028/www.scientific.net/AMR.216.171

Cheng, R., & Gen, M. (1998). Loop layout design problem in flexible manufacturing systems using genetic algorithms. *Computers & Industrial Engineering*, *34*(1), 53–61. https://doi.org/10.1016/S0360-8352(97)00150-2

Črepinšek, M., Liu, S.-H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms. *ACM Computing Surveys*, *45*(3), 1–33. https://doi.org/10.1145/2480741.2480752

Curriculum, A. D. S. (2012). High-rise Building Optimization, *1*, 305–314.

Darwin, C. (2003). *On the Origin of Species, 1859*. Routledge. https://doi.org/10.4324/9780203509104

Drira, A., Pierreval, H., & Hajri-Gabouj, S. (2007). Facility layout problems: A survey. *Annual Reviews in Control*, *31*(2), 255–267. https://doi.org/10.1016/J.ARCONTROL.2007.04.001

Dyckhoff, H. (1990). A typology of cutting and packing problems. *European Journal of Operational Research*, *44*(2), 145–159. https://doi.org/10.1016/0377-2217(90)90350-K

Eiben, A. E., Eiben, A. E., & Schippers, C. A. (1998). On Evolutionary Exploration and Exploitation. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.4885

Evolutionary Algorithms for Solving Multi-Objective Problems. (2007). Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-36797-2

Fasoulaki, E. (2007). Genetic Algorithms in Architecture : a Necessity or a Trend ? *Design*.

Gerhard Wäscher, Heike Haußner, H. S. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, *183*(3), 1109–1130.

Gharsellaoui, H., & Hasni, H. (2012). An hybrid genetic algorithm for two-dimensional cutting problems using guillotine cuts AND LITERATURE RE-, (February 2014).

Glover, F. (1989). Tabu Search—Part I. *ORSA Journal on Computing*, *1*(3), 190–206. https://doi.org/10.1287/ijoc.1.3.190

Godfrey, M. (2014). Evolutionary Algorithm Parameter Fitness: An Exploratory Study, (July).

Goldberg, D. E. (David E., & E., D. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley Longman Publishing Co., Inc. Retrieved from https://dl.acm.org/citation.cfm?id=534133

Hadjiconstantinou, E., & Iori, M. (2007). A hybrid genetic algorithm for the two-dimensional single large object placement problem. *European Journal of Operational Research*, *183*(3), 1150–1166. https://doi.org/10.1016/J.EJOR.2005.11.061

Hauser, K., & Chung, C. H. (2006). Genetic algorithms for layout optimization in crossdocking operations of a manufacturing plant. *International Journal of Production Research*, *44*(21), 4663–4680. https://doi.org/10.1080/00207540500521147

Hevner, A., & Chatterjee, S. (2010). Design Science Research in Information Systems (pp. 9–22). https://doi.org/10.1007/978-1-4419-5653-8_2

Hevner, March, Park, & Ram. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75. https://doi.org/10.2307/25148625

Hopper, E., & Turton, B. (1999). *Theory and Methodology An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem*. Retrieved from www.elsevier.com/locate/dsw

Hopper, E., & Turton, B. C. H. (2001). A Review of the Application of Meta-Heuristic Algorithms to 2D Strip Packing Problems. *Artificial Intelligence Review*, *16*(4), 257–300. https://doi.org/10.1023/A:1012590107280

House, T. (2011). The House as a System. *House System*, 13–30.

J.H. Holland. (1995). *Hidden Order: How Adaptation Builds Complexity*.

Kar Yan Tam. (1992). Genetic algorithms, function optimization, and facility layout design. *European Journal of Operational Research*, *63*(2), 322–346. https://doi.org/10.1016/0377-2217(92)90034-7

Kröger, B. (1995). Guillotineable bin packing: A genetic approach. *European Journal of Operational Research*, *84*(3), 645–661. https://doi.org/10.1016/0377-2217(95)00029-P

Kwiecinski, K., Santos, F., De Almeida, A., Taborda, B., & Eloy, S. (2016). Wood Mass-Customized Housing A dual computer implementation design strategy. *Complexity & Simplicity - Proceedings of the 34th ECAADe Conference*, *2*, 349–358.

Kwiecinski, K., & Slyk, J. (2014). System for customer participation in the design process of mass-customized houses. *Fusion: Data Integration at Its Best, Vol 2*, *2*, 207–215. Retrieved from http://papers.cumincad.org/cgi-bin/works/Show&_id=caadria2010_039/Show?ecaade2014_156

Li, H., & Love, P. E. D. (1998). Site-Level Facilities Layout Using Genetic Algorithms.

Journal of Computing in Civil Engineering, *12*(4), 227–231. https://doi.org/10.1061/(ASCE)0887-3801(1998)12:4(227)

Li, L. (2012). The optimization of architectural shape based on Genetic Algorithm. *Frontiers of Architectural Research*, *1*(4), 392–399. https://doi.org/10.1016/j.foar.2012.07.005

Lipowski, A., & Lipowska, D. (2012). Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and Its Applications*, *391*(6), 2193–2196. https://doi.org/10.1016/J.PHYSA.2011.12.004

Lodi, A., Martello, S., & Monaci, M. (2002). Two-dimensional packing problems: A survey. *European Journal of Operational Research*, *141*(2), 241–252. https://doi.org/10.1016/S0377-2217(02)00123-6

Mancapa, V., van Niekerk, T. I., & Hua, T. (2009). *A genetic algorithm for two dimensional strip packing problems*. *South African Journal of Industrial Engineering* (Vol. 20). The Southern African Institute for Industrial Engineering. Retrieved from http://www.scielo.org.za/scielo.php?script=sci_arttext&pid=S2224-78902009000200012

Mawdesley, M. J., Al-jibouri, S. H., & Yang, H. (2002). Genetic Algorithms for Construction Site Layout in Project Planning. *Journal of Construction Engineering and Management*, *128*(5), 418–426. https://doi.org/10.1061/(ASCE)0733-9364(2002)128:5(418)

Maxence Delorme, Manuel Iori, S. M. (2016). Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, *255*(1), 1–20.

Michalek, J., Choudhary, R., & Papalambros, P. (2002). Architectural layout design optimization. *Engineering Optimization*, *34*(5), 461–484. https://doi.org/10.1080/03052150214016

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Berlin Heidelberg.

Oliveira, J. F., Neuenfeldt Júnior, A., Silva, E., Carravilla, M. A., Oliveira, J. F., Neuenfeldt Júnior, A., … Carravilla, M. A. (2016). A SURVEY ON HEURISTICS FOR THE TWO-DIMENSIONAL RECTANGULAR STRIP PACKING PROBLEM. *Pesquisa Operacional*, *36*(2), 197–226. https://doi.org/10.1590/0101-7438.2016.036.02.0197

Oosterhuis, K. (2012). Simply complex, toward a new kind of building. *Frontiers of*

Architectural Research, *1*(4), 411–420. https://doi.org/10.1016/j.foar.2012.08.003

Pál, L. (2006). A Genetic Algorithm for the Two-dimensional Single Large Object Placement Problem.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). *A Design Science Research Methodology for Information Systems Research*. *Journal of Management Information Systems* (Vol. 24). Retrieved from http://www.tuunanen.fi.

Qian, Z., Bi, Z., Cao, Q., Ju, W., Teng, H., Zheng, Y., & Zheng, S. (2016). Expert-guided evolutionary algorithm for layout design of complex space stations. *Enterprise Information Systems*, 1–16. https://doi.org/10.1080/17517575.2016.1150521

Radcliffe, N. J. (1991). Forma Analysis and Random Respectful Recombination. *Undefined*. Retrieved from https://www.semanticscholar.org/paper/Forma-Analysis-and-Random-Respectful-Recombination-Radcliffe/bc72643dddd31d37e13d3878a8f97f6bee1c6fce

Simon, H. A. (1996). *The Sciences of the Artificial Third edition*. Retrieved from https://monoskop.org/images/9/9c/Simon_Herbert_A_The_Sciences_of_the_Artificial_3rd_ed.pdf

Stiny, G. (1980). *Introduction to shape and shape grammars*. *Environment and Planning B* (Vol. 7). Retrieved from http://home.fa.ulisboa.pt/~miarq4p5/2011-12/Course_Programs/0_Projecto - Arch Design IV/1_Turma A/Shape_Grammars/1_Fundamentals of shape grammars/Stiny_IntroShapeAndShapeGrammars,.pdf

Taborda, B., de Almeida, A., Santos, F., Eloy, S., & Kwiecinski, K. (2018). Shaper-GA: Automatic Shape Generation for Modular House Design. In *Proceedings of the Genetic and Evolutionary Computation Conference on - GECCO '18* (pp. 937–942). New York, New York, USA: ACM Press. https://doi.org/10.1145/3205455.3205609

Thierens, D. (2002). Adaptive mutation rate control schemes in genetic algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)* (Vol. 1, pp. 980–985). IEEE. https://doi.org/10.1109/CEC.2002.1007058

Thomas, J. (2013). Advances in Computational Intelligence, *7902*(November). https://doi.org/10.1007/978-3-642-38679-4

Valenzuela, C. L., & Wang, P. Y. (2000). A Genetic Algorithm for VLSI Floorplanning (pp. 671–680). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45356-

3_66

Wu, X., Chu, C.-H., Wang, Y., & Yan, W. (2007). A genetic algorithm for cellular manufacturing design and layout. *European Journal of Operational Research*, *181*(1), 156–167. https://doi.org/10.1016/J.EJOR.2006.05.035