Shape Evolution

An Algorithmic Method for Conceptual Architectural Design Combining Shape Grammars and Genetic Algorithms

submitted by **Orestes Chouchoulas** Bsc (Hons)
for the degree of PhD of the University of Bath
2003

Centre for Advanced Studies in Architecture
Department of Architecture and Civil Engineering
University of Bath

Orestes Chouchoulas

Abstract

It is recognised that, while computers are used widely as tools for architectural representation, they are underused as architectural *design* tools. Shape Evolution is proposed as a generic design method and software tool for supporting the initial (concept design) stages of architectural design. It aims to inspire the architect towards more innovative solutions to design problems by offering unanticipated, evolved designs that both respond to the architect's stylistic agenda and satisfy the functional requirements of the brief. Shape Evolution combines a shape grammar, used mostly as the vehicle for aesthetics and style, with a genetic algorithm that optimises designs with respect to their functional performance. The key interface between the shape grammar and the genetic algorithm is a string that encodes the sequence by which shape grammar rules have been applied to generate a given design. This string, which uniquely identifies each design, is used as the genotype for the genetic algorithm. This allows the genetic algorithm to operate by modifying the sequence of rules that generated a design, not the geometry of a design directly. Consequently, modified designs are valid in the design language defined by the shape grammar and retain the stylistic characteristics chosen by the designer.

A prototype of Shape Evolution is developed in conjunction with a simple shape grammar for the design of apartment blocks and a set of appropriate evaluation routines. Tests are performed using four different sets of optimization goals. The results demonstrate that the combination of the shape grammar and the genetic algorithm works as desired, producing useful designs that also still satisfy the

designer's stylistic requirements as defined through the shape grammar. This is the case despite the need to filter out invalid individuals produced by disruptive genetic operators, and despite the fact that the genetic algorithm is found to perform inefficiently. Suggestions are made on how to eliminate these inefficiencies. The use of Shape Evolution in a proposal for housing design competition is mentioned. Issues of interface and optimized performance notwithstanding, Shape Evolution is shown to have great potential as an architectural design tool.

Acknowledgements

Several people contributed in important ways to the work presented in this thesis and I would like to extend my gratitude to them. Specifically, I would like to thank Alan Day, supervisor of this project, for giving me the opportunity to pursue a project that was personally meaningful, and for his advice and support throughout the process. John Rollo deserves to be acknowledged for providing the initial stimulus for this work through his undergraduate lectures on methodology of design and shape grammars. Several researchers have helped by exposing me to their work through papers and discussions at various stages and by offering guidance during the initial uncertain steps. These include Terry Knight, Jose Duarte, and Benjamin Loomis at the Massachusetts Institute of Technology, Catherine Teeling at the University of Greenwich, Kristina Shea at the University of Cambridge, and Rob Woodbury at the University of Adelaide. I am particularly thankful to John Gero for his feedback on multi-objective optimisation and Alwyn Barry for his insights on the inner workings of genetic algorithms and his help in evaluating the results of the Shape Evolution prototype. The prototype itself would not have been possible without the C++ hacking expertise of Alexios Chouchoulas and his readiness to share it. Fabien Coupat's assistance with data processing for the appendices was vital and greatly appreciated. Finally, I would like to thank all my friends, local and remote, for offering support, understanding, distractions, and constructive abuse during this project and for making my PhD experience singularly enjoyable.

Contents

List of Figures

"You could probably do this without a computer,

but, man, would that suck!"

– David Roth,

Killing the Buddha with Genetic Algorithms,

June 2003

1
Introduction

1.1 Architectural Design is Wicked

Design, and architectural design in particular, is a complex process. It involves the application of a plethora of diverse skills in order to produce architectural form in response to a brief. It needs the consideration of the programmatic, functional, structural, and environmental requirements of the specific building, the intricacies of the site, the management of resources, issues of environmental performance and sustainability, but also matters of aesthetics, semantic signification, artistic value, and sociological and psychological impact. Architects are expected to be masters of all trades, equally well-versed in philosophy as in construction. This can be evidenced in the variety of subjects taught in architectural schools, attempting to cover all these subjects to some extent.

To add to this, design is a messy and unpredictable activity. The weaving together of these multiple threads does not happen in a well-ordered sequence, but is riddled with multiple feedback loops, modifying both the solution and the formulation of the design problem. Architects are constantly moving back and forth between the brief and the proposed design. Alterations in the programmatic requirements and the production of spaces that respond to these requirements are not discrete tasks happening separately and sequentially in the meeting room and at the drawing board but are simultaneous combined tasks in the studio.

In that light, it is not surprising that producing an accurate model of the design process is a non-trivial exercise. Suggested models of the design process since Asimow (1962) have identified its iterative nature and its reliance on feedback loops and revision (Lawson, 1990).

Most models divide the process into episodic stages but at the same time provide routes from each stage to previous stages. Indeed, the amount of possible routes through the proposed flowcharts precludes the prediction of any actual sequence used in practice, limiting the usefulness of these models. Indeed, their usefulness is most often limited to analysis of case studies. Designers hardly use these models to guide them when tackling a design problem in practice. The complexity of design, and the dependence of its success on many disparate (and some initially unknown) factors have contributed to Simon (1973) classifying design as an ill-structured or "wicked" problem. Wicked problems, described by Rittel and Webber (1973) are formulated tentatively, and tend to be reformulated while solutions are being sought.

Designers are able to escape this seemingly vicious circle in ways covered by a veil of mysticism. Often associated with some little-understood innate artistic talent or inspiration of some unknown ethereal origin, creativity involves ways of thinking that defy recipes. As opposed to convergent thinking, which progresses in steady steps towards a clear goal, designers employ lateral, divergent ways of thinking, such as launching into tangential possibility exploration (Lawson, 1990). Design students learn the design process by osmosis, rather than in a formal and structured fashion, and are able to use it before they can fully understand it (Wade, 1977). While obfuscating an accurate understanding of sequence in the design process, this skill allows an informed exploration of the space of possible solutions to a design problem. This stochastic exploration is of key importance in wicked problems like architectural design, as the solution space is infinite, with moving, fuzzy boundaries.

1.2 Gallery of Methodologies

In design, infinite options are an impediment to progress. To a large extent, the process of design is about constraining choice and revealing the ultimate solutions by chiselling away unwanted options. It is often unclear whether a particular design decision can be objectively considered better than another. Because of the amount of possible ways to address the same (usually underspecified) brief, design decisions seem almost arbitrary. This arbitrariness is frustrating to designers, who are striving to create optimal or at least "good" solutions. To remove this arbitrariness, designers need to find compelling overarching principles to direct them.

Architectural design methodologies have, over the ages, provided just this kind of constraint, allowing architects to focus their efforts on a sliver of the solution space. From the divine apocalypse providing the dimensions of Solomon's temple to Alberti's rules in the "Ten Books on Architecture," and from modernism to deconstruction, designers have made an effort to limit their scope to what they deem a more appropriate portion of the possible designs. These design movements are herein viewed as methodologies because their theoretical underpinnings are defining the ways by which design problems are solved and guide the designers in particular routes through the decision tree.

Obviously, in architecture the very first set of constraints is provided by clients, as a collection of performance criteria that the building must satisfy. There are design methodologies, such as functionalism, that give these programmatic requirements the chief role in decision making. The same is true in designs driven by optimisation, such us hospital plan layouts. Other methods might use psychology to suggest spaces that produce desirable effects in their users. However, in most design projects there exist various secondary or implied design goals that may not be present in the clients' specifications.

Designers are sometimes celebrated as artists, chosen by clients on the evidence of their own design sensibilities as witnessed in the body of work they have completed

in the past. These sensibilities might be manifest consistently in their designs and therefore form the core of a designer's *style*. It can be argued, for example, that Frank Gehry's building designs are informed by his desire to produce sculptural forms, beyond the satisfaction of purely programmatic requirements. These artistic, philosophical, or stylistic choices are clearly subjective, vary greatly from designer to designer, and are susceptible to changes in fashion. They can be said to represent the *zeitgeist* as perceived by each designer, but might also be based on personal memories and experiences. No matter what informs them, these design sensibilities provide first principles against which the designer judges potential solutions to design problems.

Furthermore, the ultimate shape of an architectural design will often be influenced substantially by political, social, and economic considerations. For example, Rococo, Nazi and communist buildings have features that associate them to the types of regimes they represent; religious architecture intends to elevate the spirit as well as house a congregation. It might also be desirable to design buildings that are less than optimal in terms of economising resources, for the benefit of providing lavish displays of economic power. Alternatively, a building that represents ecological thinking might help boost a company's public image.

Mathematics have also been employed in the search for design perfection. How does one decide the size of a room the dimensions of which are not constrained by need or site? Architects have looked for the answers in the dimensions of an idealised human form, harmonic ratios, and the Fibonacci series. Algorithmic design is a more recent expression of the need to rationalise aspects of a design, relying on quantifiable programmatic constraints to define form, taking away the guesswork.

Programme, function, psychology, style, fashion, art, philosophy, creed, economics, politics, and mathematics all shape designs, but their role in the process of design is to constrain decisions. The most direct way to achieve this is to follow design recipes, in other words learn from precedents and opt for that which is tried and

tested. Alberti's *Ten Books*, Christopher Alexander's *Pattern Language*, and Feng Shui all give instructions on how to design spaces. The advice is often presented dogmatically and without the reasoning behind it, but in all it represents rules of thumb leading to *good practice* and decent buildings.

The choice of method by which to guide design decisions is a very personal one and it often defines the designer. Furthermore, a particular project may be calling for the use of a specific methodology, whereas other methodologies might be entirely inappropriate. Therefore, evaluating each of these methodologies individually would be ineffective. The choice is part of the designer's creative contribution. What is useful, for the scope of this research, is to place algorithmic design in this context, and realise what it offers in terms of helping the designer reach his ultimate solution.

1.3 Optimisation, Search, Evolution

Computers, though widely accepted as very powerful tools in many disciplines, are generally underused in the field of architecture. While the vast majority of architectural practices are using computers, they are treated, in most cases, as elaborate drawing boards. Their purpose is to mimic traditional tools such as pen and paper. They are made to perform the same tasks, only in ways that are more efficient. Recently, emphasis has been placed on the ability of computers to represent and allow the specification of complex three-dimensional forms, as for example in Perrella (1998). This addresses some of the shortcomings of traditional drafting tools, but it does so by replacing them with new drafting tools. There seems to be room for improvement where it comes to harnessing computational power to assist architects in a more "intelligent" and active manner. True Computer-Aided *Design* systems, as opposed to Computer-Aided *Drawing* systems, although technically feasible, are still scarce.

Computers are good number-crunchers, which suggests that their strength lies with dealing with the quantifiable aspects of design. There are a number of software

packages available currently that are based on this principle of calculating quantities. As such, these programmes are well suited to engineering applications, like calculating light levels and reverberation times in a space, or sizing structural members. However, this is engineering *analysis* software. It is typically employed in the later stages of the design process in order to evaluate or fine-tune the design; its impact on the final design is vital, but minimal. Pairing analysis with a generative mechanism would easily allow the quick generation of appropriate or feasible designs. This combination of analysis and synthesis was in the focus of CAD research in the sixties, mostly in the field of generating spatial configurations by optimising for one or more performance criteria (Cross, 1977).

However, optimisation does not solve design problems; it improves potential designs that are already resolved to a great extent. Optimisation only modifies designs by making small quantitative changes and cannot arrive at a better solution by making a qualitative jump. Therefore, the utility of this process is lessened during the concept design stages and it has little impact on the most important design decisions.

Seen from an algorithmic process point of view, optimisation is just one of the ways to see the route from concept to finished design. Design can also be seen as a search operation. In a so-called problem space, each point represents every possible solution (Newell et al., 1967). For design problems, each point in a *design space* is a potential design. In this model, the way by which the ultimate solution is derived is by searching through the problem space for the optimal or satisfactory solution (Kanal and Cumar, 1988).

The practical problem with this view is, of course, that the problem spaces for most design problems are infinite. What is required is a very robust search algorithm that can search through very large, or infinite spaces in an efficient manner. Since in infinite spaces not all possible solutions will be checked, the search algorithm will need to be stochastic. Furthermore, the search parameters need to be semantically cogent enough for the search to be meaningful in design terms. That implies a

thoroughly worked out representation system that would sufficiently specify the design solutions.

Another way to consider design in an algorithmic framework is design as evolution. According to this view, the ultimate design is produced by a process mimicking natural selection, by evaluating the performance of each contender design in an artificial environment. By making small changes and re-evaluating designs over a number of generations there is a gradual improvement to the appropriateness of the design. Genetic algorithms (Holland, 1975) and simulated annealing, and its application in design in the form of shape annealing (Cagan and Mitchell, 1993) employ this method.

Although evolution in biology is not goal-driven, it exhibits characteristics that resemble a computational process (Pinker, 1997). Thus useful results may be produced by modelling an evolutionary system and manipulating parameters accordingly. Specifically, beyond an appropriate modelling of the operations involved in evolution, such as mutation, a careful definition of the environmental conditions will be necessary to ensure good results.

1.4 Criticism of Algorithmic Design

Criticism to the use of computers in architectural design has been plentiful since its possibility arose. Lloyd Kahn, editor of the *Shelter* books in the early seventies dismissed computer-designed architecture as an effort that "will only produce environments that machines or machine-like people will want to inhabit" (Kahn, 1973). This attitude may have been partly technophobic, and seems to share common threads with the earlier fear of dehumanisation that would have been brought about by industrialisation. However, there is also the concern that computers, calculating machines that they are, are only capable of contributing to the design process as much as "an army of clerks" (Alexander, 1965). How can a rich, unpredictable process such as design be encompassed by something as deterministic as calculation? Similarly, the poor quality of early computer graphics

led architects to reject CAD drawings as crude, simplistic, and incapable of nuance at least as late as the late nineties in this author's experience. This criticism has its roots in the difficulty in prognosticating regarding the future capabilities of computer technology, and specifically in the counterintuitive observation that a simple difference of degree such as the increase in computational power can yield a qualitative difference. However, there are threshold points beyond which computers are no longer thought of as powerful calculators anymore. To the large percentage of people for whom using computers is a part of life, the enabling power of their digital tools seems to be far beyond calculation. It is now easy to accept that the use of computers in the design of a building does not necessarily create drab, gridded, clinical spaces. Progress in the understanding of complex systems has shown that unexpected patterns can emerge from simple deterministic rules. Computer generated drawings have the capacity to be crude and simplistic, or beautiful and evocative, the primary factor being the skill of the draughtsman. Computation has superseded this early criticism.

During the course of this research there has been criticism of algorithmic design methodologies from colleagues who contend that the resulting designs are devoid of meaning. Indeed, human designers infuse their work with semantic content drawn from their extensive cultural, aesthetic, and philosophical baggage. The outcome of the design process is presented as a clear consequence of an initial vision. This results in a lively diversity, which, in the view of this author, is to be celebrated. However, one must not confuse the method for the designer. Computers, usually the means by which algorithmic processes are employed, are merely tools. The computer's output is always deterministically dependent on the programmer's and the user's input. For the purposes of this research, algorithms are just tools among many at the designers' disposal. The choice of method and tools by the designer is a fundamental design decision that is highly semantic in itself and greatly affects the finished design's significance. Simon (1971) remarked that decisions about the design process may inform what we call *style* as much a teleological concept of the

final design's miscellaneous properties. The accurate and scientific way by which algorithms produce results may suit some designers better than more free-form or artistic approaches. A designer opting for an algorithmic approach method is making a statement that defines him or her.

A further common concern is whether the logical conclusion of the use of algorithmic design processes will lead to the complete removal of the human designer from the loop (Watanabe, 2002). Certainly, the human designer will be needed for the near future to decide on the use of algorithmic processes to begin with. But more importantly, the role of the designer is likely to evolve in tandem with the designer's tools. De Landa (2001) sees the designer working with genetic algorithm-based tools as a metallurgist, guiding the evolution of form in the same way as a craftsman controls the intensive properties of a material in order to tease the form from it. He calls for architects to become aware of computer programming, biology, thermodynamics, mathematics, and other disciplines that can provide the means to harness processes and guide the emergence of designs (De Landa, 2002). He further identifies that this necessitates a shift away from considering design as the forming of a single artifact and towards "populational, intensive, and topological" thinking. This view of the designer as a controller of processes is also shared by Leach (personal communication, 18 February 2003).

Far from being "just a formal game" (Stiny, 1998), algorithmic and rule-based processes allow designers to manipulate their designs on a meta level. Instead of modifying geometry and then testing the effects of the modifications, with algorithmic tools it is possible to define the effects desired and let the appropriate geometry emerge from that. This is a significant change in the way designers work and think. It has the potential to improve the artificial world, firstly by allowing designers to be aware of the performance-related effects of their choices, and eventually by letting them manipulate the effects directly.

1.5 Objectives of Shape Evolution

What is interesting is how these algorithmic tools might be used and what potential benefits they might bring to the process of design. The aim of this project is to investigate the use of computers in a way that engages more with the design process, assisting in *design generation* as opposed to merely *design representation*, and to ultimately propose a software tool for architectural concept design.

To be practical for its potential users, such a tool will need to match and surpass the utility offered by more traditional, tried-and-tested methods. It should, for one, be generic, i.e. applicable to a wide variety of design problems. It should exploit the large computational power of computers by dealing automatically with quantifiable characteristics of design, thus helping users avoid following dead-end routes to impractical or unfeasible solutions. This way, the tool could allow its users more time for creative exploration, which it should also support. But how?

The process of architectural design is typically supported by a large amount of background information, including anything that might have been absorbed during the seven-year educational process required before a new architect may qualify. However, it is recognised that the production of innovative design solutions is not dependent on knowledge of a direct mechanism that leads from requirements to solutions (Rosenman, 1997a). Indeed, preconceived notions tend to limit one's capacity for innovation and efficiency. Take, for example, a simple layout problem defined as follows:

> Given a 10×10 grid, place as many 1×1 squares (houses) on that grid as possible. The squares need to have at least one adjacent space free. All the free space on the grid must be contiguous.

The problem approximates the placing of houses on a square site. The first constraint ensures that all houses are accessible and will be referred to as the

adjacency constraint. The second constraint, referred to as the *contiguity* constraint, ensures that every house on the site is accessible from any point on the site by ensuring there are no closed courtyards.

The expectation is that human solutions, representing the use of prior knowledge to tackle this problem, will sacrifice efficiency for clarity and order. Computer solutions using an approach that does not rely on knowledge are expected to be more efficient and random-looking. The following are three typical human solutions provided by architectural design students:

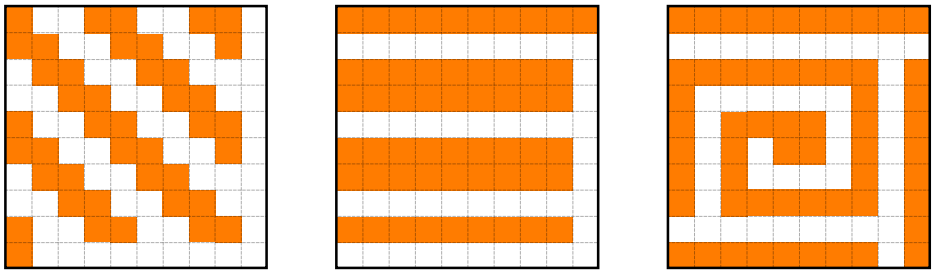


Figure 1.1: Typical student solutions to the toy problem

These are three of the best solutions:

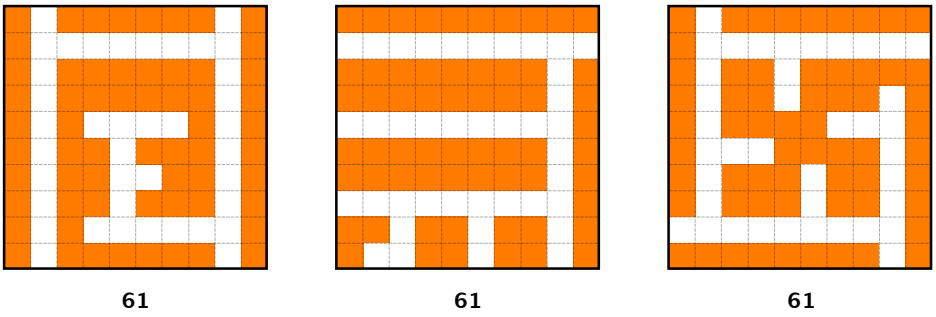


Figure 1.2: Best human solutions to the toy problem

These results seem to justify expectations. It is obvious that some of the typical solutions are guided by a desire to adhere to a salient construct (stepping, rows, and a spiral construct are represented in the typical results). The best solutions are sacrificing clarity of layout concept for small areas of singular adjustments to achieve greater efficiency.

Furthermore, much in the way that exposure to more kinds of music alters one's appreciation and taste of music, exposure to new kinds of design solutions informs the designer's collection of sought-after qualities in designs (Simon, 1996). In this light, creative exploration can be supported by having the tool propose atypical or unanticipated (yet functional) solutions to the user, allowing him or her to break out of creative ruts, and encouraging innovation. For that to happen, there needs to be a sufficient separation between the user's input and the programme's output so that the results cannot be easily predictable. Furthermore, the tool should be reasonably fast and user-friendly, to facilitate the evaluation of numerous solutions by the designer, and subsequent tweak-and-run cycles that may be necessary.

Finally, it should be able to accommodate the designer's established stylistic and aesthetic preferences. This would ensure that the designer gets more gratification out of the tool, but also that the products of the tool do not share common characteristics that can be directly associated with its use. That would, of course, be perceived as a limitation of the tool. In an infinite problem space there exist infinite solutions that satisfy the functional requirements. The choice of the right solution among those should be made by the designer, not dictated by the tools.

This is in line with a view of architectural design as a largely serendipitous, rather than goal-oriented, process: an investigation of alternatives. The brief must be satisfied of course, but the architectural solution is not connected to the brief in a deterministic cause-effect relationship (as some modernists have claimed). It is expected that different users employing this tool for the same project should reach remarkably different solutions.

The tool's function would be to quickly suggest a number of concept designs that when refined will produce functional building designs. As such, it is positioned right after the problem definition in the design process. However, it would eliminate several feedback loops by ensuring that most of the functional resolution happens concurrently with the concept design.

1.6 Overview of Thesis

The next chapter will open the discussion with an overview of work related to the use of algorithms in architectural design, with a focus on evolutionary strategies. Chapter 3 will introduce the general structure of Shape Evolution as a method and discuss the innovative way by which a shape grammar and a genetic algorithm can be meaningfully combined. Chapter 4 will delve further into the mechanisms of Shape Evolution, and describe how prototypical implementation for the design of apartment blocks works. Tests using this prototype are described and their results are analysed in chapter 5. Finally, in chapter 6, the prototype will be evaluated and improvements will be proposed, topics for further research will be outlined, and general conclusions will be drawn.

2
Review of Relevant Work

2.1 Introduction

This thesis describes the development of a prototypical computer implementation of Shape Evolution, a tool to support the concept design stage of architectural design. Shape Evolution combines aspects from two strands of computational design research: one deals with formal syntactic approaches to analysis and generation, and the other with evolution and self-organisation. This chapter outlines and reviews research done in these fields, setting the scene and leading up to Shape Evolution.

Early efforts to produce computer-based architectural design tools will be presented and reviewed, followed by brief explanations of genetic algorithms and the shape grammar formalism, both of which have received a lot of attention in the field of design research. Existing design tools based on an evolutionary paradigm will be reviewed next, followed by systems combining evolution with meaningful representational frameworks, much as Shape Evolution proposes.

2.2 Computer-Based Architectural Design Tools

There have been numerous attempts to create computer-based, generative tools for architectural design. Several different approaches and various methods have been employed, each with its strengths and limitations. Indeed, such efforts go back to

the early days of computers and computer-aided design, with an effort by Moseley (1963) to produce improved (i.e. lower-cost, in this case) building layouts through a quantitative analysis of circulation requirements. Moseley's linear programming implementation generated concept massing designs that focused on the placement of components and vertical circulation. Her approach made several crude assumptions in order to fit the example problem to the method employed.

Referencing Moseley's work, Whitehead and Eldars (1964) addressed some of its limitations on a similar space layout problem. Their work produced suggested hospital plans based on data on the movement of nurses, surgeons, and other hospital personnel. The authors claim that careful planning can reduce movement between rooms by 25%, which corresponds, by their calculations, to a 8.5% saving in salaries. The building users' movement is therefore taken as a prime consideration for internal planning decisions, if the stated goal is to produce a more economical design. Perhaps the most problematic aspect in their programme is that it relies on the collection of movement data from similar, existing buildings under use, rather than computer simulation. This data is then used to determine the level of interaction of each room with other rooms. The synthesis process consists of placing the room with the highest level of interaction in the centre of the plan, placing the room that most interacts with it adjacent to it, then the room that shares the most interactions with the two rooms already placed is positioned next to both of them, and so on, until the least "interactive" spaces are arranged in the perimeter. The placement of rooms beyond the second one is done though a trial-and-error process that evaluates the "cost" (in resulting journey length) of each possible positioning and selects the most cost-efficient. Cross (1977) notes that, due to the lack of a determinate solution strategy for problems of this kind, the generated solutions are not optimum, but near-optimum.

A similar approach was taken by the STUNI programme (Willoughby et al., 1970, Willoughby, 1970), initially geared towards producing designs for the campus of Stirling University. STUNI incorporated several improvements over the Whitehead and

Eldars offering, including, crucially, some amount of user interaction. Furthermore, STUNI introduced a wider range of criteria informing the arrangement of elements, including site conditions.

The Basic Architectural Investigation and Design Programme One, or BAID-1 (Auger, 1972), which also took site conditions into account, was developed at the University of Leicester. Its initial purpose was to enable medium-rise high-density housing developments as an alternative to bland high-rise buildings by making it easier to produce configurations compliant to British planning controls, the difficulty of applying these controls having been identified as one of the key obstacles towards more differentiated medium-rise projects. Specifically, the programme ensured that designs receive the minimum amount of daylight and skylight incident on windows specified in the building regulations. The programme produced designs by placing building elements randomly, and then checking for compliance with regulations. If the placement of an element failed the compliance test a new random position was chosen. In this way, the programme effectively conducted a random search of the problem space, looking for acceptable solutions. It follows that the design produced at each run would just be an instance of a successful design, and not an optimum. Auger himself commented that the products of BAID-1 should be considered stimuli to the designer's own ideas. This way of using the programme was served by the increased level of interactivity, allowing the user to freeze the positions of building elements that have been placed to his or her satisfaction, then letting BAID-1 come up with a new random configuration for the remaining elements. Of course, a significant amount of work needed to be done by the designers in order to interpret and adapt the computer output into realisable designs. Auger's flowchart for the design process using BAID-1 delegates the analysis of computer output to the human designer, after which follows "manual completion."

These early efforts, and projects that followed them like CEDAR (Chalmers, 1972) and SPACES (Th'ng and Davies, 1975), which included the synthesis component SPACES 2, were limited in scope and dealt with very specific design problems from

very specific viewpoints. They also shared the requirement for the involvement of the user for manual completion of the design. That places them firmly in the concept design stages of the design process. However, Lawson (1990) comments that whatever solution the computer might have offered that satisfied the constraints tackled by the programme would have to be altered significantly by the designer in an effort to make the design more responsive to other criteria. In this way, the computer is given the upper hand in the design process, with the human designer subsequently attempting to reshape the computer output into something useful. This seems to contradict the original intention of these programmes, which was to provide support to the designer and allow him to make more educated decisions. Clearly, such programmes must support the specification of any arbitrary number of criteria.

Another common thread connecting these projects is that they concern themselves with the space layout problem (Cross, 1977). This is of course a key problem in architectural practice. It is also ideal for investigation using computer programmes, because most space layout problems are intractable, i.e. there is no deterministic solution strategy that guarantees optimal solutions.

At the same time, the Massachusetts Institute of Technology was home to a much more radical and ambitious approach to the use of computers in architectural design. Instead of focusing on tools that support the design process, the goal was to produce programmes that solved design problems by demonstrating designer-like intelligence. The first fruits of this effort were the part of the URBAN series of programmes, of which URBAN 5 (Negroponte, 1967) was the culmination. URBAN 5 was meant to act as a partner to the designer. While the operator used a rudimentary graphical interface to design by configuring cubic modules, the computer "monitored" the process, interrupting with suggestions or warnings, or providing information when asked, all based on predetermined criteria. Negroponte describes the interaction with the machine as being a conversation, the aim being to produce "responsible architecture" through a system that allows the consideration of many

important criteria affecting design decisions at the same time. Telling, perhaps, of the ambition to eventually develop this programme into a complete artificial intelligence, was the effort put into ensuring that the input of criteria and the "conversation" between the machine and its user during design exploration was conducted in English. However, like most early efforts at writing programmes that displayed human-like intelligence, URBAN 5 did not: its responses to design issues were burdened by the preconceptions of its programmers (Negroponte and Groisser, 1970). Made up of specific algorithms, it was incapable of applying its "skills" generically.

This lead to the creation of the Architecture Machine Group in 1968, with the stated goal of creating an intelligent machine for the purposes of architectural design. Mirroring the situation in mainstream artificial intelligence research, the group made progress in sub-problems such as human-machine interfacing and visual recognition, clearly indicating their intention to eventually produce a robotic architect (Cross, 1977), believing that this was the way towards innovation in environmentally responsive architectural design. The Architecture Machine Group was later subsumed into the MIT Media Lab.

This course of research has very ambitious aspirations. Indeed, some argue that the goal of creating an artificial intelligence is fundamentally unattainable (Dreyfus, 1972), arguing that Pinker's view of the mind as a computation machine (Pinker, 1997) is flawed. Artificial intelligence research is steering away from attempting to emulate human-like intelligence by stringing together a large number of specialised machines. Spurred on by discoveries in complexity theory and cognitive science, intelligence is now thought to emerge out of complex interactions between relatively simple components (Fogel, 1995, Johnson, 2001). However, artificial intelligence research has also produced some very practical methods for dealing with uncertainty, ambiguity, and fuzziness that have found applications in many fields. The methods discussed in section 2.5 have their roots in this work.

Commercial CAD packages have recently started offering advanced tools that go beyond the representation of designs. Bentley's *Microstation* includes a feature called Dimension-Driven Design, which allows the parameterisation of design elements, and therefore provides a quick way of exploring design alternatives. This has been used by the architectural practice of Foster and Partners for the Swiss Re Headquarters and the Gateshead Music Centre (Constantinou, 2001). *Catia* software, created jointly by Dassault and IBM for the aeronautical industry, and famously appropriated by Frank Gehry for projects like the Guggenheim Museum in Bilbao (Mitchell, 1999), includes tools for engineering analysis and integration with Computer Aided Manufacturing (CAM) equipment, as well as advanced modelling tools. *Genesis*, the design system used by Boeing, combines geometric modelling, assembly hierarchies, a high-level specification of constraints, evaluation criteria, and design generation facilities in the same representation (Heisserman et al., 2000). These solutions, however, support the design process in its later stages, long after key decisions on form have solidified. Consequently, while they might be able to improve on the minutiae of a particular design, they cannot bring radical innovation in the process or allow a rethinking of the design at the conceptual level.

2.3 Genetic Algorithms

Genetic algorithms were developed in the 1970s by John Holland (Holland, 1975) in an effort to formally understand biological adaptation in nature. Much like the organisms they were meant to study, genetic algorithms have since taken a life of their own and have been proven robust in tackling optimisation problems as well as exploring very large search spaces (Goldberg, 1989). This makes them appropriate to the solution of design problems when these are represented as a search through a design space. As mentioned already, this analogy is derived from the definition of a problem as a search through a space of knowledge states, some of which represent solutions to the problem (Newell et al., 1967). In the same fashion, design problems can be thought of as a search through a design space whose elements are designs. Some of these designs are appropriate solutions to a design problem and some are

not. The design process can then be seen as a search for the optimal solution (Kanal and Cumar, 1988). Genetic algorithms can also cope with infinite problem spaces, which design spaces often are. For example, all the designs within a design language specified by a shape grammar can be elements of a design space. Some of these designs are better in addressing the design problem at hand.

The main strength of genetic algorithms as problem solvers is that they do not require an explicit optimal solution generation method, relying instead on a generate-and-test process. They are therefore well suited to solving intractable problems. In the place of an explicit solving strategy, genetic algorithms merely require a way to evaluate solutions. That is achieved by an objective function that can assign scores to solutions according to how well they address the problem. If the problem can be expressed in terms of quantifiable goals then an objective function can be produced. The only other element required by a genetic algorithm is an initial, random selection of solutions to work on, i.e. random starting points in the problem space.

Genetic algorithms operate on representations of solutions, not the solutions themselves. In keeping with the biological metaphor, solutions are called *phenotypes* and their coded representations are called *genotypes*. Phenotypes can be encoded as genotypes in several ways: strings, *n*-dimensional arrays, lattices, topological graphs, etc. The facility to work with such diverse data structures enables genetic algorithms to tackle a great variety of problems. To demonstrate the process of encoding solutions as genotypes, the shape of the features of a face can be encoded in a four-bit binary string, where the digits represent the shapes of the head, eyes, nose, and mouth sequentially. The value **0** specifies a round shape, whereas the value **1** specifies a rectangular shape. Example faces and their corresponding genotypes are shown in Figure 2.1 below.

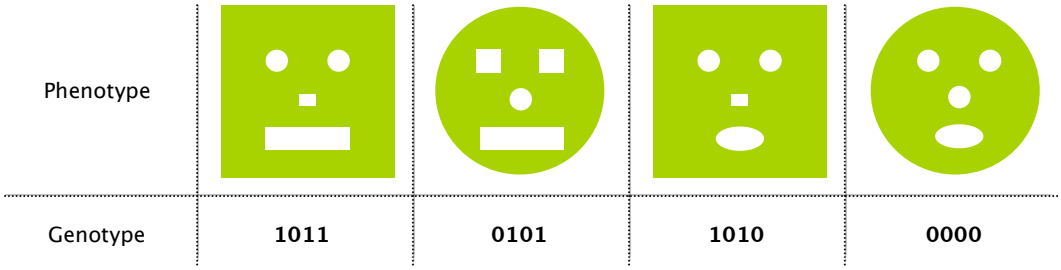


Figure 2.1: Stylised faces and the genotypes that describe them

The process by which the coded genotype is translated into the phenotype is called embryogenesis. In this case of the stylised faces, each digit corresponds to a feature of the phenotype. Borrowing the terminology from biology, a segment of the genotype that contains the information for a particular feature of the phenotype is called a *gene*, whereas the values that a gene can take are known as *alleles*.

Vital to genetic algorithms are the *crossover* and *mutation* operators. These operate on the genotypes, and mimic their counterparts in natural evolution. Crossover is the process of splicing together the genotypes of two "parent" solutions to produce offspring that combines the characteristics of both parents. Take for example the two eight-digit binary genotypes 110|01110 and 011|01011. The vertical line represents the crossover point. Splicing these two genotypes together gives us the "child" genotypes 110|01011 and 011|01110. Mutation is applied randomly to the offspring, changing parts of the genotype. For example, a mutation can change the genotype 11001011 to 110**1**1011.

This flowchart illustrates how a simple genetic algorithm works:

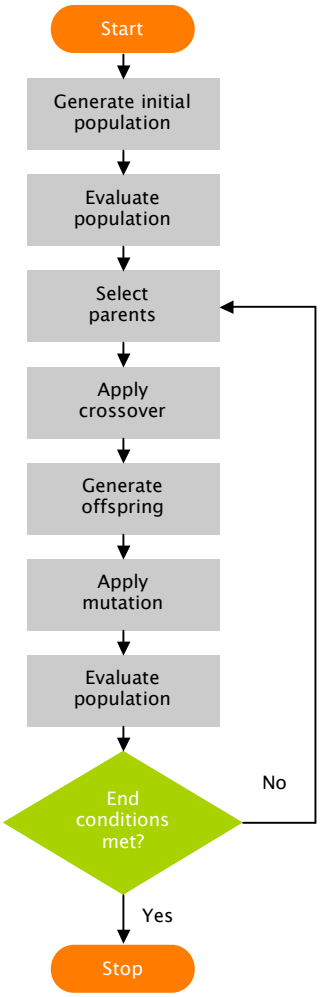


Figure 2.2: Flowchart describing a simple genetic algorithm

An initial population of genotypes is generated at random. Each member of this population is evaluated for fitness by use of the objective function and a fitness value is attached to it. Higher scoring individuals are selected as parents and produce offspring by crossover. A small amount of mutation is then applied to the offspring. That produces the new generation of solutions, which in turn is evaluated for fitness and the system loops. The loop is broken when the termination conditions are met. The termination conditions might be the generation of an individual that surpasses a fitness threshold value (a “sufficiently good” solution), or the completion of a specified number of generations.

The genetic algorithm works by gradually improving the fitness of solutions with each generation. Like in natural selection, the characteristics of unsuitable individuals are not perpetuated while fit individuals are allowed to pass on their

features. Holland's Schema Theorem (1975), later elaborated by Goldberg as the Building Block Hypothesis (1989) attempts to explain this process more rigorously through the study of *schemata*.

Schemata are in essence templates representing groups of genotypes that exhibit similarities. For example, using four binary bit string genotypes, the schema 01**, where the asterisk is a wildcard symbol represents the genotypes 0100, 0101, 0110, and 0111. If these genotypes represent the stylised face phenotypes mentioned above, the schema 01** describes all faces with a round head and rectangular eyes. Schemata have two significant characteristics: The *order* of a schema is defined as the number of fixed bits in the template. For example, the schema **0*10** is of order 3, and the schema 1101**1* is of order 5. The *defining length* of a schema is the distance between the furthest two fixed bits in it. So the defining lengths of the schemata **0*10**, 1101**1*, 0******1, ****1*** are 3, 6, 7, and 0 respectively.

The Schema Theorem states that highly fit schemata of low order and short defining length will be found in exponentially increasing copies in each population. These schemata are referred to as *building blocks*. The Building Block Hypothesis proposes that genetic algorithms produce better scoring individuals by combining building blocks into whole genotypes. There is an ongoing debate about the validity and usefulness of the schema theorem (Mitchell, 1996, Christiansen and Feldman, 1998), but it serves at least to give an insight into the reasons why genetic algorithms seem to work.

The general flowchart and operators described above pertain the simple genetic algorithm (Goldberg, 1989). There exist several elaborations of the basic algorithm, which can be more appropriate to different types of problems. Bentley (1996) lists some of these advanced types of genetic algorithms. These tend to modify the way genotypes are encoded and manipulated through the used of more exotic versions of the genetic operators mentioned above.

2.4 Shape Grammars

The shape grammar formalism was introduced in the early seventies by George Stiny and James Gips (1972). The linguistic metaphor that permeates shape grammars is the result of their foundation on the work of Noam Chomsky (1957, 1965) on generative and transformational grammars in linguistics. Their function is to specify classes of designs through an algorithmic understanding of the processes that generated them.

A shape grammar consists of a vocabulary of shapes (with or without labels), a set of shape rules, and an initial shape. The rules are presented as transformations of a shape or collection of shapes (shown on the left hand side) to a new shape or collection of shapes (shown on the right hand side). Applied recursively on an initial shape, the rules produce designs that are said to belong to a language (Stiny and Gips, 1972, Stiny, 1975).

To illustrate the shape grammar formalism, consider the simple shape grammar defined in Figure 2.3 below. The vocabulary of shapes consists of a single rectangle of ratio 1:2. There is a single rule, which places another rectangle adjacent and perpendicular to the first. The rectangle, being the only shape in this grammar, also serves as the initial shape. Figure 2.4 shows a few designs belonging to the language defined by this simple shape grammar.

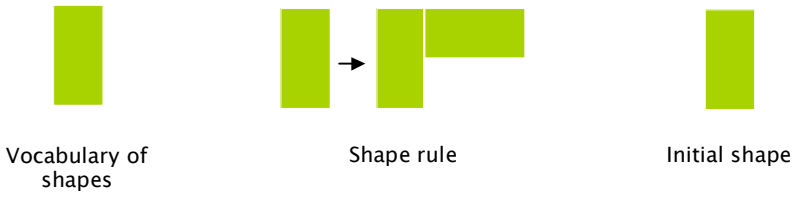


Figure 2.3: An example of a simple shape grammar

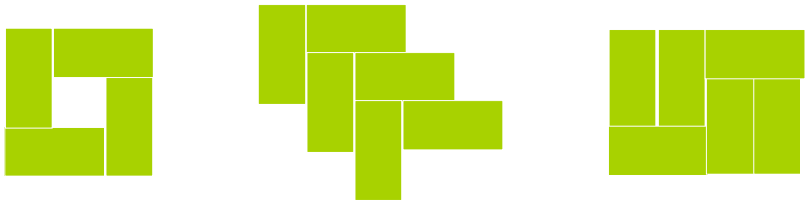


Figure 2.4: A few designs derived from the simple shape grammar

The shape rule states that if recognised in the design, the shape on the left hand side should be transformed to the configuration on the right hand side. However, the left hand side may be recognised in any of its isometric transformations, i.e. its scaled, translated, rotated, and reflected forms, or combinations of these (Mitchell, 1990). With the spatial relationship defined by the rule in the example shape grammar, and as a result of the symmetry properties of the rectangle, the rule can be applied to each rectangle in the design in four different ways, as shown in Figure 2.5.

Figure 2.5: The four possible ways of applying the rule in the simple shape grammar

To control and direct the application of rules in light of this ambiguity, labels may be employed. Labels specify where and how a rule will be applied, removing ambiguity caused by symmetry. Figure 2.6 illustrates the use of labels to define explicit shape rules. In this case, the white dot is the label. It marks one of the corners of the rectangle, thus removing its symmetry properties under reflections and rotations. Four different explicit rules can then be defined for the placing of a new rectangle in the same spatial relationship defined in Figure 2.3.

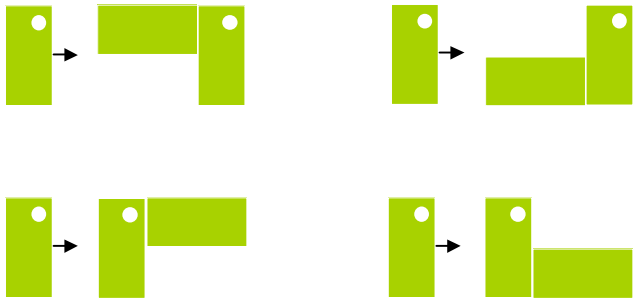


Figure 2.6: The use of labels to define explicit shape rules

A key characteristic of the shape grammar formalism is that the vocabulary shapes are not treated as discrete objects, but as collections of geometrical entities. This makes it possible to recognise and consider non-predefined shapes in the designs, formed by the relative positioning of these geometrical entities. These emergent shapes can then be used in shape grammar calculation, i.e. they can have rules applied to them. The term *emergence*, when used in a shape grammar context, refers exactly to the formation of these new shapes (Stiny, 1994, Knight, 2003). Figure 2.7 shows a simple shape grammar that demonstrates emergence. In the spatial relationship defined by the single rule of this grammar the overlapping of two 1:2 ratio rectangles creates a shape where six 1:2 ratio rectangles can be recognised: the two original rectangles plus the four smaller ones created by an exclusive-or logical operation between the two original rectangles. Those smaller rectangles are *emergent* shapes. Because the left hand side of the shape rule consists of a 1:2 rectangle, it can be applied to the emergent rectangles, producing designs like the one presented in Figure 2.7.

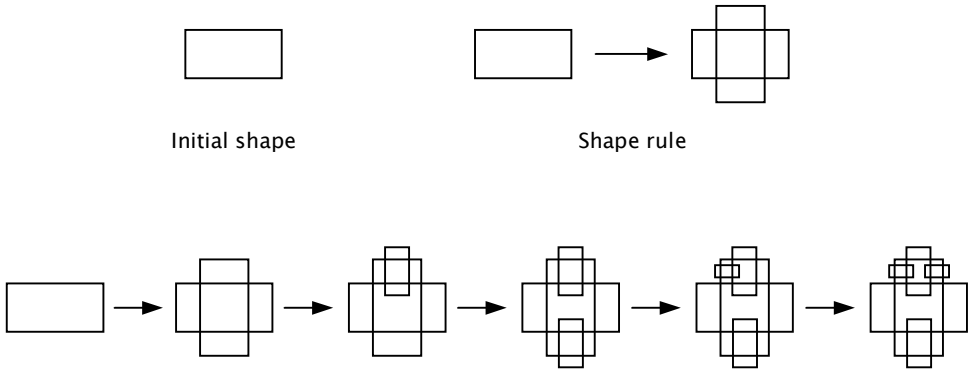


Figure 2.7: A simple shape grammar to demonstrate emergence

Initially developed to provide insights into the cognitive mechanisms of the design process (Gips, 1979), shape grammars have, since their inception, used both in analysis and synthesis. In terms of analysis, they have been called to serve as descriptors of style. If shape grammars are to be sufficient in defining a style, they need to satisfy three "tests of adequacy" (Stiny and Mitchell, 1978). First, they must make apparent and describe the "underlying commonality of structure and appearance" among instances in a corpus of work. Second, they should provide a mechanism for identifying whether a particular design is an instance of the style under consideration. Finally, they should supply a method for generating new designs within that style.Shape grammars have been successful in these respects as applied analytically to the definition of historical styles including Palladian villas, Queen Anne houses, Frank Lloyd Wright Prairie houses and window designs, Japanese tearooms, Mughul gardens, and Hepplewhite chairs (Knight, 1994).

Another stated aim of the shape grammar formalism is to provide the mechanism for the creation of *new* design languages, i.e. new design styles. That is achieved simply by creating a vocabulary and a set of rules from scratch. The use of shape grammars in a generative, instead of an analytical capacity is of particular interest. Its advantage is that a very simple grammar with a limited vocabulary and few rules can create significantly complicated and unanticipated results (Rowe, 1987). As such, shape grammars are potentially a very useful tool for design innovation. Furthermore, a look at shape grammar applications attests to the great diversity of

design languages that they are capable of defining. Indeed, Stiny (1975) shows that they are theoretically capable of producing any possible design. These characteristics of shape grammars make them ideal for use in a generic design tool with innovation as one of its main goals. There are fewer actual examples of generative applications of shape grammars, with Alvaro Siza's Malagueira housing project being prominent among them (Duarte, 2001), although they have been used extensively for students' projects in schools of architecture.

There has been ongoing research on the various aspects of the shape grammar formalism since its inception, 30 years ago. Still, a complete computer implementation of a generic shape grammar system has been elusive. This is mainly due to the algorithmic complexities of coding emergence, as seen for example in one of the sub-problems dealt with by Tapia (2000). As a result, many of the current shape grammar parsers tend to disregard emergence altogether (Chase, 2000a). The most promising shape grammar parser currently in development is the two-dimensional *GEdit* by Mark Tapia (Tapia, 1999, Knight, 1999, Chase, 2000a). Tapia is currently working on a new three-dimensional parser, which at the time of submission of this thesis is not yet very usable. However, a generative design tool based on the parsing capabilities of Tapia's programme could potentially produce very successful results. Knight (1999) and Chase (2000b) also point out that interface problems stand in the way of the acceptance of current shape grammar parsers by non-programmers. Despite these difficulties, research on shape grammars has plenty of momentum, and it seems only a matter of time before a usable generic tool emerges.

2.5 Evolutionary Generative Design Systems

In recent years, and further spurred on by the publication of *Emergence* (Johnson, 2001) and the evocative work on complexity theory coming out of the Santa Fe Institute, architects have been keen to employ techniques that do not take a structured, goal oriented approach to the production of form. Instead, form is

expected to emerge out of a set of given conditions in an organic way (Rahim, 2000, Rahim, 2002). Computer scientists have long been working with systems that display emergent behaviour, using autonomous agents, self-organising elements, cellular automata (Wolfram, 2002), and evolutionary models. Evolution is particularly relevant as it "exemplifies the 'explore, evaluate, and refine' subprocess of architectural design and its overall non-linear nature" (O'Reilly and Ramachandran, 1998).

Some interesting work that is evocative of the applicability of these techniques to design problems should be mentioned. Richard Dawkins (1986) developed a simple genetic algorithm, guided by user selection of preferred designs, generating two dimensional drawings ("biomorphs") reminiscent of plants and animals. Particularly suggestive of the possible applications of these systems in design was William Latham's artwork (Todd and Latham, 1992, Todd and Latham, 1999), produced using evolutionary techniques, and displaying forms that resembled biological organisms. Subsequently, a plethora of such examples of "genetic art" appeared in the 1990s, usually aiming only to produce interesting abstract images (Rowbottom, 1999). In a different vain, Karl Sims's "creatures" (Sims, 1994a, Sims, 1994b), although composed entirely of simple parallelepipeds, displayed remarkably life-like behaviour, which was evolved by breeding successive generations in a virtual environment.

Evolutionary tools have in fact been used extensively in design, particularly in engineering, but only in order to optimise existing designs. Bentley (1996) lists numerous examples of such applications in engineering design. It should be noted that evolutionary techniques have also been employed for the optimisation of parts of architectural design. "Evolve" (Cawthorne and Sparreboom, 1995) produces variant forms of a building design starting with the input of simple CAD drawings. The alternatives are evaluated in terms of einvironmental performance and construction cost, but the choice is made manually by the user. More variations on the newly selected design is then displayed and the process is repeated until the user finds a design satisfactory. Elezkurtaj and Franck (2000) have demonstrated a very effective

and usable system for optimising floor plans using a genetic algorithm and a constrained topology of rooms. Impressively, their system allows changes in the constraints while the optimisation is running, and presents the new results in real time. More recently, a genetic algorithm has been employed for the optimisation of whole building forms, starting with a user-provided schematic design and driven by criteria relating to daylighting and energy use (Caldas, 2002). These efforts, however, do not allow the generation of concept designs from scratch, thus depending on unsupported human design for the initial, and potentially crucial, design decisions. A detailed description of these efforts is therefore beyond the scope of this literature review (but they are revisited in section 6.2 where they are compared to the completed Shape Evolution prototype).

Frazer's work is perhaps one of the earliest examples of the application of evolutionary techniques to architectural design specifically (Frazer, 1995). His projects involve geometric transformations and evolution of designs, revelling in the unexpectedness of the resulting designs. The "Universal Interactor," designed by Frazer and his students at Unit 11 of the Architectural Association in London, used three-dimensional cellular automata controlled by genetic algorithms. The evolutionary process is guided by data collected using an array of exotic input devices, including sound and light sensors, infra-red movement detectors, wind-sensing piezo-electric "grass," touch-sensitive body suits, and jewellery that can pick up the movement of finger joints. All this measured data conspires to influence the complex deformations of a spherical surface. This "datascape" is meant to represent an ever-changing environment that creates the evolutionary pressure for the genetic algorithm at the heart of the system. However, despite the use of a genetic algorithm, the Universal Interactor performs what is in essence a random search through a constrained problem space, given that the design is shaped by a system of arbitrary interpretation of irrelevant information. The forms produced are of course unanticipated and innovative, and valuable if only for that reason. However, more semantic content is infused into the design by the loose-wire

aesthetic of the data-collecting hardware than by the data itself. Although there is nothing stopping this system from being applied to any design problem, the lack of meaningful relationships between input and output would make the production of useful concept designs unlikely.

The analogy of natural evolution is used again in the computer-aided design of Makoto-Sei Watanabe's subway station at Iidabashi in Tokyo (Watanabe, 2002), claimed by its designer to be the first built project whose design was generated by a computer programme. Watanabe develops a method termed "induction design" as part of a larger project that aims to reveal and model the processes that create similar patterns in complex systems of different scales, with a focus on urban growth. The Iidabashi project concerns the design of a ceiling-suspended lattice of steel tubing with integrated lighting, over an escalator connecting the underground train platform to the surface. The design of the lattice starts as a random pattern of overlapping line segments, representing tubular steel members, which is refined through hard and soft regulation. Hard regulation concerns the range of allowable angles between steel members, ensuring that the members are forming a lattice by being connected at nodes at both ends, and constraining the three-dimensional form of the lattice in the space available. Soft regulation allows the user to specify areas that need to be clear of members, enabling the structure to bypass and embrace existing structure, or areas that need to be denser in order to perform better structurally. Initially implemented in *Mathematica*, the system uses a genetic algorithm optimises the forms, with user feedback providing the evaluation of the design population, and therefore "guiding" the design process (Tanaka and Kiriyama, 2000). The desire to produce unanticipated results is here made secondary to the user/designer's preference. However this method seems to be far from generic. While its principles might be applicable to other design problems, it would be necessary for a new programme to be written specifically for each particular problem.

Bentley first produced a generative system that can be applied generically to different design problems and does not depend on a priori definitions of high-level components (Bentley, 1996, Bentley, 1999). This system, called GADES (Genetic Algorithm DESigner) employs a phenotype representation using clipped stretched cubes as primitives for the approximation of any solid geometrical form. These forms are "sculpted" by a genetic algorithm using a multi-objective ranking system to select fit designs. Bentley tests his system by generating designs for tables, heat sinks, penta-prisms, boat hulls, and aerodynamic cars, demonstrating its capability to cope a wide spectrum of engineering design problems. In architectural design, GADES has been employed for the evolution of hospital floor plans. However, the use of an entirely generic phenotype representation, though sufficient for engineering design, would fail to produce anything but functionalist architectural solutions. To get around that using this system, one would need to figure out a way to encode stylistic and aesthetic requirements as evaluation criteria. Beyond simple requirements such as symmetry or adherence to proportional systems, this would be a highly non-trivial task.

2.6 Combination of Generation and Evolution

Clearly, evolutionary systems, as exemplified by those using genetic algorithms at their core, are very powerful and versatile. They enable the designer to consider any number of criteria simultaneously without the need to define a priori solution strategies. They provide the means for optimising designs through an extensive examination of alternatives. However, if these tools are to infiltrate architectural design studios they need to be generic. At the same time, they need to produce designs that are meaningful to their designers and pertinent to each specific design problem. The key to providing these features seems to lie with the choice of representational system for the designs themselves, and also the structure of the genotypes representing these designs when using a genetic algorithm (Woodbury, 1993). There have been a few recent examples of design tools combining generative

systems, which provide the representation framework, and an evolutionary system, which evaluates and modifies designs.

Chan, Frazer, and Tang (2002) attack this problem by using a hierarchical representation combined with a genetic algorithm. Their system is based on a model of design that describes it using hierarchical topologies. These hierarchies are abstractions of design solutions. Indeed, they are abstract enough to allow them to be application-independent. To demonstrate this system, the design of wine glasses is considered. That example makes it apparent that, although there is a single, generic hierarchical core to this system, further representations and mapping are required to apply the system to a particular design problem. Genotypes and phenotypes, and the process that maps them to each other, still need to be defined at every hierarchical division point in this framework. It would seem logical, if the generic representation is a hierarchy, for the hierarchy itself to be the genotype. But despite the initial development of promising concepts, that is not the path taken in this project.

Cagan and Mitchell (1993) introduced shape annealing, a method combining shape grammars and simulated annealing, an evolutionary optimisation algorithm that uses multi-objective evaluation of solutions. The very simple shape grammar defines a language for the generation of truss-like structures composed of linear members. Parameterisation allows the variation of structural member thickness depending on load conditions. This was demonstrated for the design of unusual yet efficient trusses, transmission towers, and domes (Shea and Cagan, 1997, Shea and Cagan, 1999a, Shea and Cagan, 1999b). This was further developed by Shea (2000) into eifForm, a generative structural design system capable of generating free-form structural systems for a wide range of situations and load conditions. EifForm, though still under development, is showing the signs of a mature and stable system that has been applied to provide innovative structural solutions for challenging situations. Recently, eifForm benefited from being used for the design of a structure that actually got built. As part of the *Swarm Tectonics* workshop, coordinated by Neil

Leach at the Academie van Bouwkunst in Amsterdam, and in collaboration with architects Spela Videcnik and Jeroen van Mechelen, a canopy was designed and built by students to accommodate the end-of-year party in June 2002 (Shea, 2003). This project demonstrated how the eifForm system was capable of accommodating a real context and architects' suggestions for modifications to the design. Though this system is intended for the design of structures, this intention is actualised only because of the shape grammar used and the evaluation criteria employed. Both are easily changeable, and therefore little stands in the way of eifForm being turned into a generic design system, given the availability of a shape grammar interpreter which could be added to the system. Shape annealing works on a single design as opposed to a population, as is the case with genetic algorithms. This has the benefit of allowing a very direct manipulation of the design, but for that sacrifices the ability to perform a broad, parallel search of the design space, thus potentially missing more readily appealing or radically different candidate designs.

Rosenman and Gero (Rosenman, 1997b, Rosenman and Gero, 1999) have actually used a design grammar to provide the representation for a genetic algorithm in a system for the production of architectural floor plans. However, the prime interest in this project is the development of a hierarchical framework for dividing design tasks to discrete stages, thus allowing the use of simple genotypes. There seems to be little concern for using the grammar in order to generate designs "in language." This, however, was of central importance in a working paper describing the SGGA (Shape Grammar Genetic Algorithm) system (Loomis, personal communication, July 28 2003). Loomis suggests a very meaningful combination of a shape grammar and a genetic algorithm that parallels the intentions of this thesis. Unfortunately, this project was abandoned, never having gone further than a statement of intent.

By the time Loomis's working paper appeared, work on Shape Evolution had gone far enough to establish that this approach is more than promising. The combination of shape grammars and genetic algorithms, if done in a way that allows both components to offer their benefits in full, can fulfil all the requirements for a useful

architectural design tool as outlined in section 1.5. However, as evidenced by the review of relevant literature, this route has not yet been taken far enough to demonstrate the potential power of such a system.

3
Shape Evolution Overview

3.1 Combination of a shape grammar and a genetic algorithm

Genetic algorithms are robust search algorithms that can be used very effectively for finding solutions that satisfy a set of quantifiable standards, such as the functional requirements for an architectural project. Shape grammars are good at providing an aesthetic and structural specification for the generation of forms. Combining the two produces a system that allows the specification of a design space using a shape grammar but also provides the means for navigating this space effectively by use of a genetic algorithm. Furthermore, it has been observed that while specific determinate algorithms might be faster at finding solutions to specific problems, genetic algorithms will offer good solutions to a wider range of similar problems (Fogel and Council, 1995). This makes the use of genetic algorithms ideal for use in a design tool that purports to be generic.

This combination should be able to quickly produce a number of "appropriate" solutions to a design problem. The designs are expected to be appropriate in two ways. Firstly, their style is dictated by the shape grammar, which is specified by the designer, and therefore the designs produced will fit in the designer's corpus. Secondly, the genetic algorithm selects the designs that best satisfy functional

constraints (e.g. site boundaries and building regulations for an architectural design problem).

To achieve this, and retain all the benefits afforded by shape grammars and genetic algorithms, it is not sufficient to merely let the genetic algorithm get to work on a shape grammar definition. The iterative nature of the evolutionary approach forces a precise identification of how the parts of the system will interact and share information. Clearly, the intention to use shape grammars means that the designs produced by the tool should be valid in the language defined by that shape grammar. If those designs are to be evolved using a genetic algorithm, the operations on them must retain their shape grammar generated nature. In other words, every design in every population during the evolutionary process must be a valid design in the language.

A separate, sequential application of a genetic algorithm on a shape grammar, does not guarantee the generation of designs that remain parts of a predetermined language. If, for example, the genetic algorithm modifies designs by applying three-dimensional scaling and shearing, the rule-based generation process that produced the original design becomes irrelevant. In all probability the original shape grammar will not be able to produce the modified design, which would therefore not qualify as part of the designer's specified design language. In short, it must be ensured that every evolutionary cycle modifies designs without taking them outside the design space defined by the shape grammar.

In fact, as genetic algorithms' operations are affected on a genotype rather than the designs themselves, it is up to the genotype to represent designs produced by the sequential application of shape grammar rules. The most obvious way of achieving this is by directly encoding this rule application sequence as the genotype.

3.2 Shape Code

The generation process involves a chain of shape grammar rule applications starting with an initial shape. The shape grammar is defined by a usually small number of shapes in the vocabulary and a usually small number of rules. Consequently, it would be easy to code the generation process for a design into a simple string. That string would encode which initial shape was used, which rules were applied to which subsequent shapes, and in what sequence. This string, called a *shape code*, has been employed by Koutamanis (2000) for the purpose of cataloguing and retrieving designs.

By way of explanation, take the example shape grammar shown in Figure 3.1. There are three rules, R_1, R_2, and R_3, defined using a label. The bottom row of images shows the sequence of generation for a simple design. The shape code for the resulting design is **R_1 R_3 R_2 R_2 R_3**.

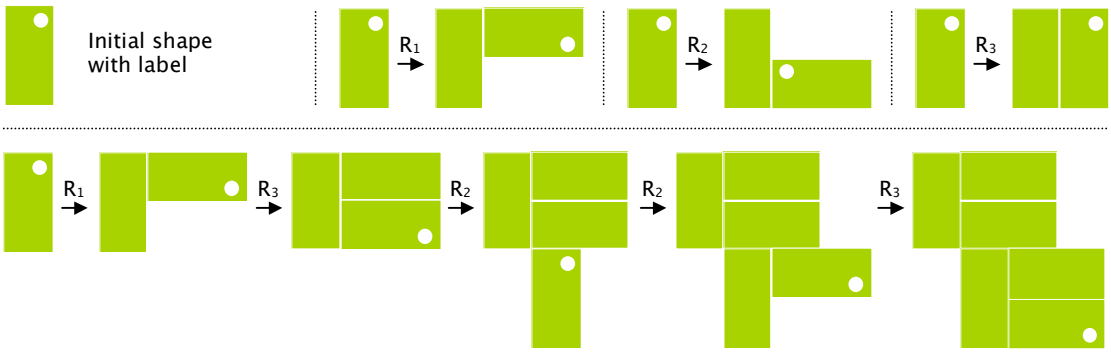


Figure 3.1: Example of simple design generated by a shape grammar

Note that in this case the rules remove the label from the original shape and place it on the added shape. That way there is no ambiguity about which shape a rule is applied to from reading the shape code alone. Grammars that do not use labels or allow the application of rules on any recognisable shape in the design can be accommodated by shape codes that are "deeper" than a simple string. For example, in the shape code **A (B C (A B) D (C D))** the rules in parentheses are applied to the shape added by the rule immediately preceding them. Therefore, this shapecode

describes a sequence in which: rule A is applied to the initial shape; rules B, C, and D are applied to the shape added by rule A in the previous step; rules A and B are applied to the shape added by rule C in the previous step; rules C and D are applied to the shape added by rule D two steps back. Since the generation of designs by use of a shape grammar requires a known set of shapes and a known set of rules it is possible to encode any shape grammar produced design into a data structure. This data structure can be used as the genotype for a genetic algorithm.

Using the shape code directly as the genotype offers three very significant benefits. Firstly, it ensures that the all designs during the evolutionary process are valid in the language. It also means that the genetic algorithm's operations on the genotype, such as crossover and mutation, alter the selection and sequence of shape grammar rules used for the generation of a design: potentially very meaningful alterations. In addition, the process of embryogenesis is simply a process of parsing the sequence contained in the shape code to produce geometry. Then the designs can be evaluated with respect to their physical attributes.

3.2.1 Shape Code Ambiguity

It is important to note a key issue in the use of the shape code that limits its usability. Every rule sequence, and therefore every shape code, generates a unique design. However, there exist many possible shape codes that could describe a specific design. Alternative shape codes for the same design might merely show a different order for the same rules. More rarely, the same design might arise out of a completely different rule sequence. This means that there is no one-to-one correspondence between a shape code and the design it generates. In other words, while it might be possible to parse a shape code to produce a design, the process cannot be inverted: a shape code cannot be unambiguously deduced from the design (Koutamanis, 2000).

As a result, it is possible that there will be cases in which the modifications that the genetic algorithm operators carry out on the shape code might produce the same

design. The extent of this ambiguity will vary widely between different shape grammars, but for some design languages it is likely to affect the efficiency of the genetic algorithm, by making it re-evaluate designs that have already been considered.

This happens because the genetic algorithm's stochastic exploration happens in fact in a space of genotypes rather than the actual design space. Because of the lack of a one-to-one correspondence between these two spaces, the genotype space is larger and it therefore takes longer to search. This issue does not qualitatively affect the capability of the genetic algorithm to search through the design space, but it means that for certain shape grammars the search will take longer.

3.2.2 Invalid Shape Codes

Using the shape code as the genotype creates one further complication. Overlapping abstract shapes is usually not a problem, but when these shapes represent physical entities, as it happens when a shape grammar is used to generate architectural designs, there are limitations. That is especially true in the cases of shape grammars used to organise modular building blocks. In these cases there exist shape grammars where not all possible rule sequences are valid, and where conditions need to be met for rules to be applicable. However, the genetic algorithm's crossover and mutation operators can, over a number of generations, entirely change a shape code. This means that when manipulating some shape grammars, the genetic algorithm is likely to produce rule sequences that are invalid. Figure 3.2 shows one such sequence, based on the example shape grammar defined in Figure 3.1. This problem will in general not occur in shape grammars that work by subdividing shapes, such as the Palladian floor plan grammar (Stiny and Mitchell, 1978).

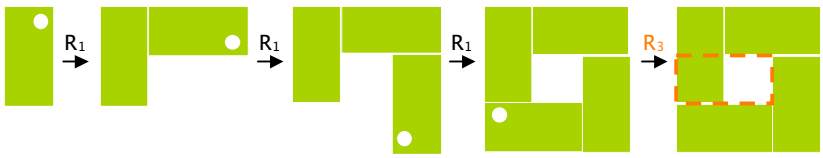


Figure 3.2: An invalid rule sequence

It is possible to minimise the significance of invalid shape codes, and their resultant designs, by severely penalising them in the evaluation process. Invalid designs are therefore evolutionarily disadvantaged and are less likely to be selected to contribute genetic material for the next generation. However, this method does not ensure the invalid codes are forced out of the genetic algorithm's population. Furthermore, for some shape grammars there will be a large number of invalid codes produced after each generation, saturating the population with undesirable designs.

Clearly it is better if invalid shape codes are weeded out before they ever make it to the evaluation algorithms. Provided that the initial population fed into the genetic algorithm is comprised entirely of valid individuals, it is only at crossover and mutation that invalid shape codes can be produced. Adding controls to these two operators so that they are only applied in ways that guarantee valid results solves this problem. This sort of peace of mind comes at a cost, namely a further decrease in the genetic algorithm's performance while searching for optimal solutions: more computer processing power will need to be expended by the controlled evolutionary operators. However, it is necessary to safeguard the integrity of the shape grammar framework in order to get results that are controlled, and therefore remain meaningful to the user of the system. The way the crossover and mutation controls work is explained in detail in sections 4.7 and 4.8.

3.3 System Overview

Shape Evolution is a prototype system combining shape grammars and genetic algorithms in the way described, implemented as a computer programme. It is meant to receive input from the designer and output concept design solutions that satisfy a

set of requirements and form part of the designer-created design language. Figure 3.3 shows a general flowchart of the system.

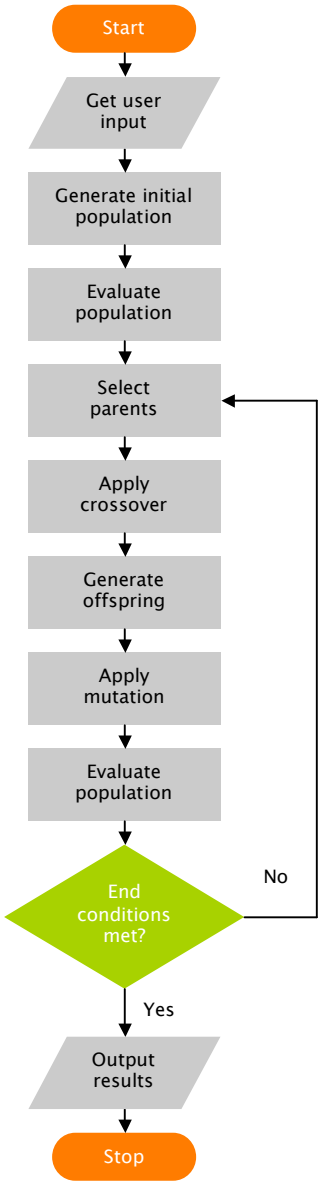


Figure 3.3: Shape Evolution flowchart

The programme starts by getting and parsing user input. That includes a definition of the shape grammar to be used, as well as goal values for quantifiable properties of the design. The genetic algorithm will be using those to assign a score to each design according to how close to the goal values it is. Further user input includes genetic algorithm variables such as population size, mutation rate, and the end conditions. The latter determine when Shape Evolution will terminate its run and can

be satisfied either after a certain number of generations or when designs with scores better than a user-defined threshold value are produced.

Shape Evolution then proceeds to generate the initial population using the shape grammar. The initial population can be random, but needs to consist of designs that are part of the design language. Every design in the population is then evaluated and has a score value attached to it.

The fittest (i.e. highest-scoring) members of the population are then selected. These designs will be providing the genetic material out of which the next generation will be produced. The genotypes of the selected parents are paired up randomly and crossover is employed to generate two new genotypes per mating. After being mutated probabilistically, these genotypes form the new generation of designs.

A further evaluation and scoring of the new population helps determine if the end conditions have been met. If not the selection-crossover-mutation cycle starts again to turn out a new population. If the end conditions are satisfied the best designs produced are presented to the designer and Shape Evolution stops.

What happens next is up to the designer. Some of the ultimate designs might be usable as they are, others might serve as inspiration, and others may be entirely useless. Provided the programme takes only a few seconds to satisfy the end conditions and stop, the designer might begin tweak and run loops by altering the input and observing the impact on the usefulness of produced designs. Experienced users of Shape Evolution should be able to make educated guesses about the effect of their input and thus control the system effectively.

3.4 The Apartment Block Problem

As a method, Shape Evolution is expected to be applicable generically on any design problem. However, the development of a prototype to provide proof-of-concept for this thesis would be better served by focusing on a particular example. For testing purposes, the formulation of the problem needs to provide opportunities for widely

variable requirements and resulting formal solutions, while at the same time remaining as simple as possible. On the other hand, the problem should be not be trivial enough to have obvious solutions.

A three-dimensional elaboration of the simple layout problem was formulated to serve as the example design problem for the Shape Evolution prototype. The problem seeks innovative and functional design concepts for a multi-storey apartment block. For the sake of simplicity, the apartment block will consist only of apartment units and the horizontal and vertical circulation spaces (i.e. corridors and stairwells/lifts respectively) connecting them. A necessary requirement for all designs is that all circulation is contiguous and that there is an accessible entrance on the ground floor.

A simple shape grammar producing designs of this sort can be constructed with only two shapes in its vocabulary: a multi-purpose circulation unit and the apartment unit itself. The circulation unit can be approximated by a 4m $\times$ 4m $\times$ 4m cube, an envelope that can easily accommodate vertical and horizontal circulation as seen in Figure 3.4. The apartment unit is a generously sized single bedroom flat at 16m $\times$ 4m $\times$ 4m. A possible internal layout for the flat is suggested in Figure 3.5. To allow more flexibility in the arrangement of the apartments, we can accept that the large window in the living room can be placed on any of the three walls. Also, the entrance to the flat can be moved a few metres along the wall without any problems. Finally, the entire plan of the apartment can be "handed," i.e. reflected along its long axis, allowing the entrance to be placed on the opposite wall. However, it is taken as given that one end of the oblong apartment will house the living room. Simplified abstract representations for these units will be used, as seen in Figure 3.6. The circulation block will be symbolised by an open orange framework and the apartment by a grey box, with a green end denoting the location of the living room.

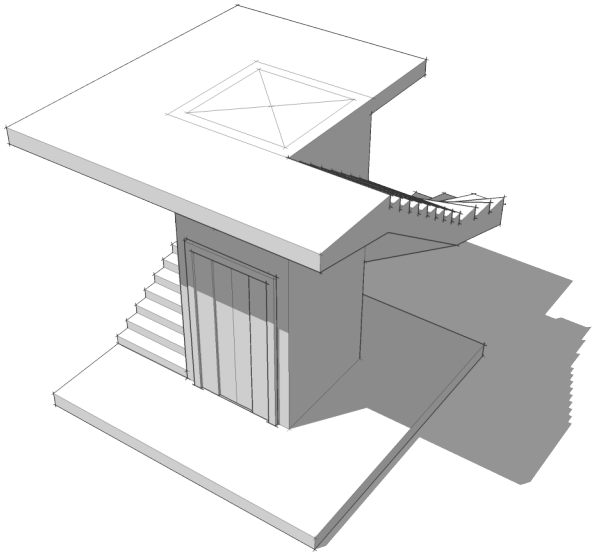


Figure 3.4: Example use of the circulation block

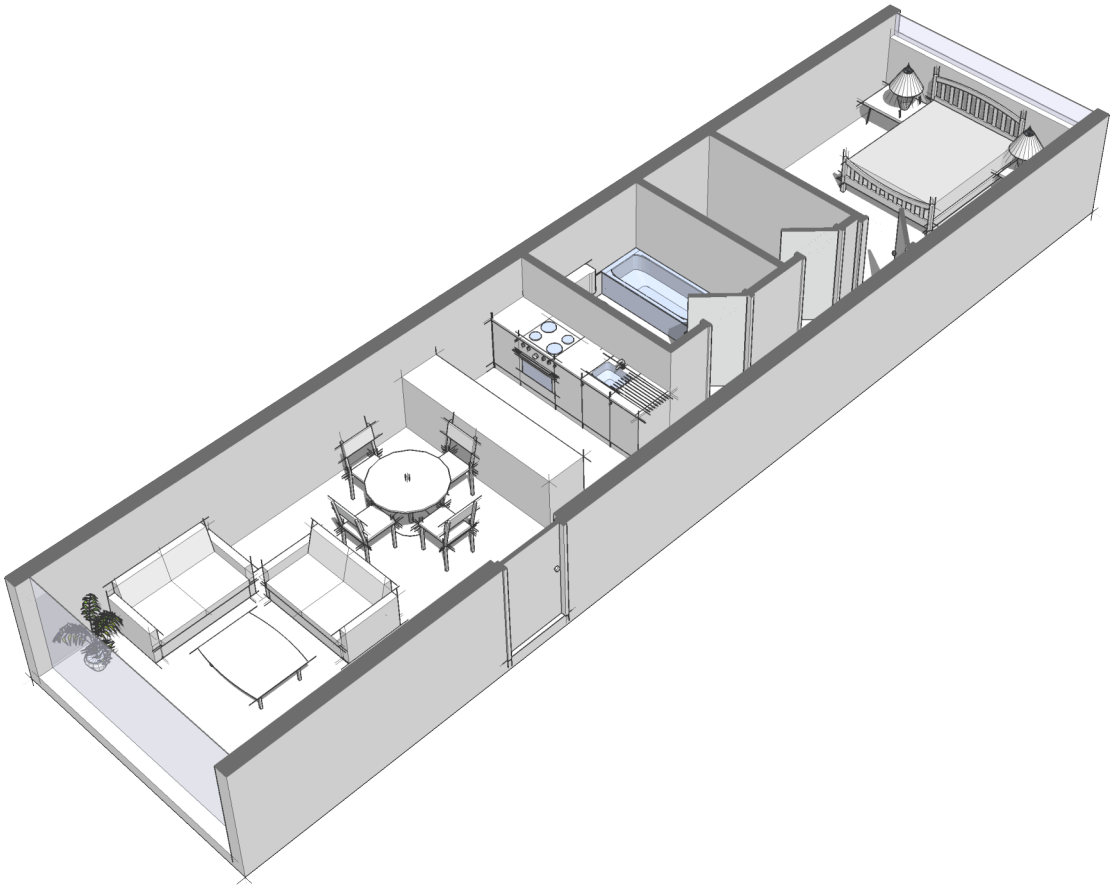


Figure 3.5: Possible internal layout for the apartment unit

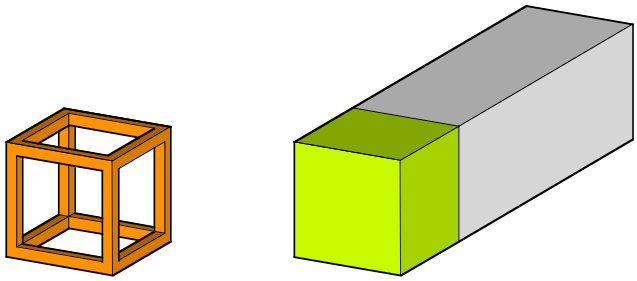


Figure 3.6: The two shapes in the apartment block shape grammar vocabulary

It seems reasonable to use a circulation block on the ground level, representing the entrance to the building, as the initial shape. To eliminate the possibility of the entrance being surrounded by built form and therefore rendered inaccessible, this building will not have any ground floor flats. The entrance must therefore carry visitors up one level in the first instance. Consequently, the initial shape for this grammar should be changed to a stack of two circulation blocks.

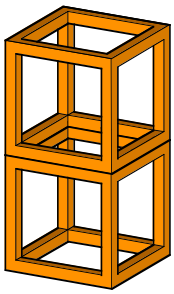


Figure 3.7: The initial shape for the apartment block shape grammar

In order to build upon the initial shape, there need to be rules that add shapes to the circulation block at the top of the initial stack. In fact, contiguous circulation and accessibility for every apartment can be easily assured at the level of the shape grammar definition by using a circulation block on the left hand side of every rule. That way, circulation blocks are appended to by either an adjacent apartment (ensuring access to that apartment) or an adjacent circulation block (ensuring contiguous circulation). Using two possible ways of attaching an apartment to the circulation route allows for extra variability in the resultant building forms. The apartment can be attached to the circulation either at its living room end, as shown in the first rule below, or four metres further away from that end, as shown in the

second rule. A third rule can attach another circulation block to an already existing one. In essence, the third rule creates a contiguous chain of circulation spaces, the circulation core of the apartment building. The first two rules then attach apartments to existing circulation. Figure 3.8 shows these three basic rules.

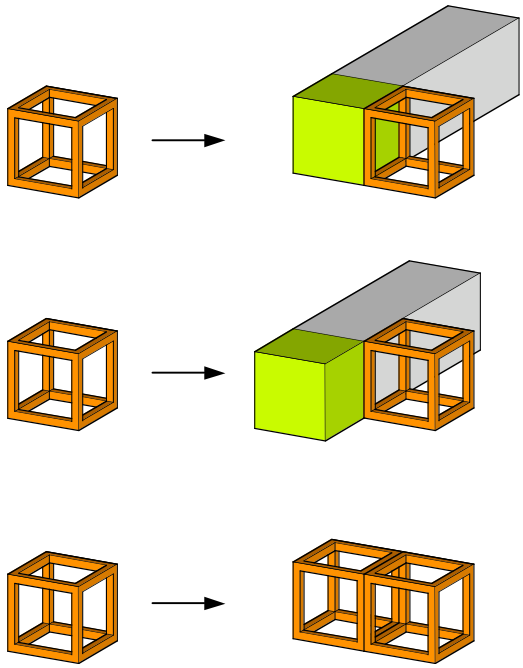


Figure 3.8 The three basic rules of the apartment block shape grammar

Most of the isometric transformations of these rules are applicable and appropriate, but some are clearly not. For example, transformations of rules that place the apartment units upright cannot be allowed. Explicitly defining only the usable transformations of these rules produces a total of 22 rules, as seen in Figure 3.9. Rules 1 to 8 are rotations and reflections of the first basic rule on the horizontal plane. Similarly, rules 9 to 16 are rotations and reflections of the second basic rule. Finally, rules 17 to 22 are extending existing circulation to all six possible directions. Note that the simple abstract symbol used to denote the circulation block has several symmetry properties that allow all possible transformations to be exhausted in six variations. The actual configuration of circulation within this abstract cube might vary, but this should not affect the definition of the shape grammar at this stage, as the “fleshing out” of the abstract representations produced by Shape Evolution is left as a task for the human designer.

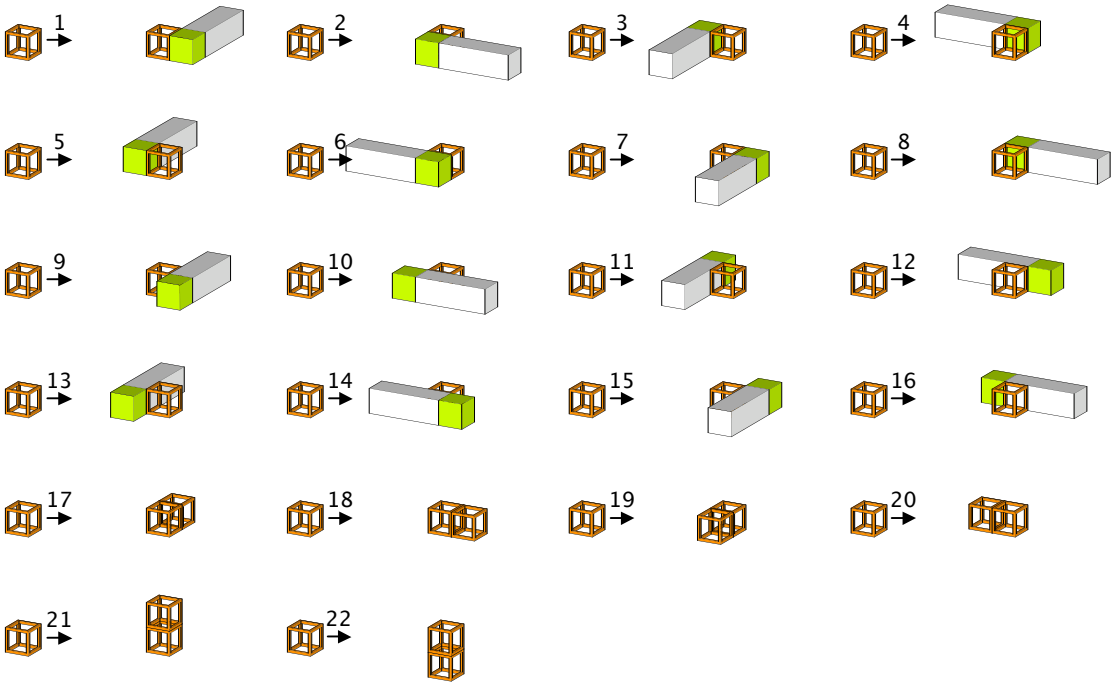


Figure 3.9: The 22 explicit rules of the apartment block shape grammar

Concept designs for apartment buildings can be produced by use of this shape grammar by starting with the initial shape and then applying rules sequentially. An example, showing the generation of a simple apartment building is shown below. Each new stage in the sequence is the result of applying the rule designated above each arrow on the last placed circulation block.

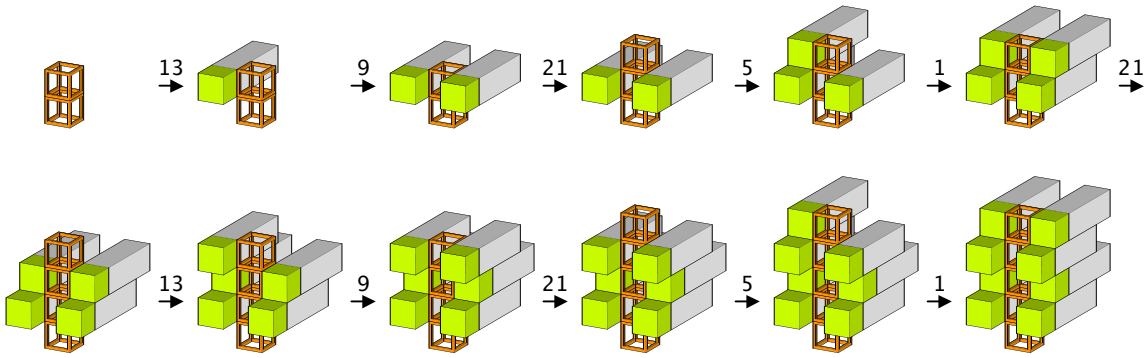


Figure 3.10: Example sequence of generation of concept design for an apartment building using the shape grammar

3.5 Evaluation Criteria

Concept designs produced using the shape grammar described above display characteristics that can be described quantitatively and that can be used to define a set of desirable characteristics for the optimisation of designs by Shape Evolution. The extents of the resulting buildings are an obvious starting point: height and footprint can be measured and constrained for. A count of the apartment units and circulation units used for each design is also useful, as it can provide a measure of layout efficiency as defined by the ratio of apartment floor area over total floor area. The stacking of apartment units may create opportunities for balconies when it is possible to step out from the living room end of an apartment onto the roof of built form from the level below. Given that apartments with balconies will usually have a higher value, a developer might wish to control the percentage of apartments with balconies in a building. For reasons of value, but also for controlling the engagement of the building with the context of the site, it might also be sensible to measure and optimise for views out of the living room towards particular directions. This measurement can also help avoid situations where an apartment's views are completely blocked by built form around it.

All these quantities can be measured from the produced designs, providing a record of each design's characteristics. Any combination of values for one, some, or all of these quantities can be used to describe a desired set of characteristics for the ideal apartment building. Each apartment block design can then be scored by comparing its measured characteristics with the goal quantities.

4
Shape Evolution In Detail

4.1 Computer Implementation of the Shape Evolution Prototype

The choice was made early on to code Shape Evolution in C++. This was done not only because of the speed that the language affords, but also because of the plethora of free, cross-platform programming tools and tutorials available on this particular language. Cross-platform portability was deemed important in order to keep the project's requirements minimal and to facilitate the further development of Shape Evolution beyond the requirements of this thesis.

Apple's Mac OS X was chosen as the development platform for Shape Evolution because of the excellent free development tools that are provided with it, combined with the flexibility of its Unix command line. *ProjectBuilder*, Apple's programming environment, uses the cross-platform gcc3 compiler, enabling an easy migration to different Unix platforms or to Microsoft Windows systems.

For development purposes, the output of Shape Evolution will be in files adhering to open standards. Documents will be produced as text files or HTML/CSS files, and three-dimensional geometry will be exported in the form of VRML files. Viewers for these file formats exist in abundance for all platforms, so their use will not be detrimental to Shape Evolution's portability.

As mentioned already, Shape Evolution is intended as a generic concept design tool. That implies that the user/designer will have to feed a large amount of input into the system in order to define all necessary variables. User input is required to define the problem space, to specify goal characteristics for the produced designs, and to provide the settings that control the running of the Shape Evolution process.

The main consideration for the prototype developed for this thesis is to show that Shape Evolution as a method yields useful results. For that reason, and to forego the difficulties inherent in providing a user-friendly interface for data entry, much of the designer input has been hard-coded into the source code for the prototype. More specifically, since the prototype will focus on finding solutions for the example problem described in section 3.4, all the information required to define the problem space (i.e. the shape grammar and appropriate representation frameworks) has been fixed into the code in advance. The same is true for the evaluation algorithms pertaining to the architectural typology generated by the prototype.

Furthermore, it was decided that designing and implementing a full-featured, generic shape grammar parser for Shape Evolution merited a long-term research project on its own and was beyond the scope of the current undertaking. Instead, a generative system specifically designed for the problem at hand was conceived, with capabilities limited to those required by the apartment block shape grammar. Emergence, for example, will not be accommodated by this generative system, as the shape grammar at hand does not allow for it.

It should be noted here that while the term “grammar” refers specifically to generative systems that do exhibit emergence, the apartment block shape grammar is a rule-based generator chosen for its simplicity that can comfortably be accommodated by the shape grammar formalism. Indeed, given a full-featured, generic parser, Shape Evolution would be capable of dealing with any shape grammar, no matter how complex.

As mentioned in section 2.3, several variations of the basic genetic algorithm have been developed. Each of these has been shown to provide better solutions to a particular genre of problems. It was decided early on that, as improvements in the performance of the genetic algorithm are secondary to an investigation of the fundamentals of its combination with a shape grammar for the purposes of this thesis, the prototype should use a bare-bones genetic algorithm architecture that can subsequently be elaborated as needed. Furthermore, the decision to opt for the simplest genetic algorithm to implement was supported by Davis's observation that genetic algorithms are very forgiving, as even poor implementations can yield acceptable results (Davis, 1991).

4.2 Representation of the Phenotype

Before the apartment block shape grammar can be encoded into a form that can be manipulated by software there needs to be a framework for representing the shapes and their relative positions. In its abstract form, the apartment block shape grammar is based on a 4m × 4m × 4m cubic module. This makes it easier to avoid programming a shape grammar parser or interpreter or a generic system for the description of three-dimensional form. Instead, the prototype can rely on a simple three-dimensional array that can be populated with numbers representing different kinds of cubic modules. Starting sensibly by representing a void by a zero, a circulation block can take the value 1 in the array, and the four cubes that make up an apartment can take the values 2, 3, 4, and 5, starting at the living room end.

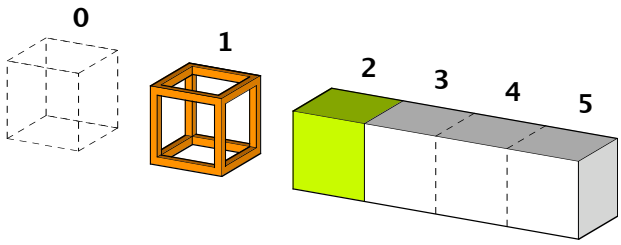
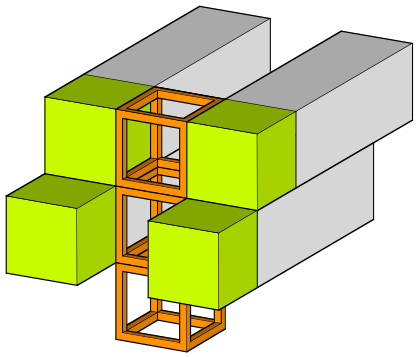


Figure 4.1: The numerical values assigned to each cubic module in the apartment block shape grammar

As an illustration of the concept of the three-dimensional array consider the design below, displayed with its array, shown as three two-dimensional sets of numbers.



Level 1:	Level 2:	Level 3:
0 0 0 0 0	2 3 4 5 0	0 2 3 4 5
0 1 0 0 0	0 1 0 0 0	0 1 0 0 0
0 0 0 0 0	2 3 4 5 0	0 2 3 4 5

Figure 4.2: An example apartment block design expressed as an array of values

Using this three-dimensional array encoding the rules is fairly straightforward. Rules can be defined by changes of the array values in specific positions relative to the position of the circulation block. So, if the directions of *i*, *j*, *k* are as shown in Figure 4.3 and the unit length in this Cartesian system is equal to the 4m module, then rule 1 can be described like so:

> For each array position with value 1 and coordinates (*i*, *j*, *k*), change the value at position (*i*, *j*–1, *k*) to 2, the value at position (*i*+1, *j*–1, *k*) to 3, the value at position (*i*+2, *j*–1, *k*) to 4, and the value at position (*i*+3, *j*–1, *k*) to 5.

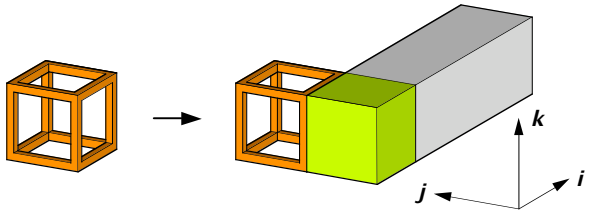


Figure 4.3: The first rule of the apartment block shape grammar with a guide showing the direction of the vectors in a Cartesian system

In this way, all 22 explicit rules used in the shape grammar for the prototype can be defined as operations on the three-dimensional array. Although it is trivial to make the size of the containing array (i.e. the three-dimensional extents of the space in which the resulting building is required to fit) user-configurable, for the purposes of the prototype this has been fixed to a cube with an edge length equal to 16 modules, i.e. 64m × 64m × 64m. This provides ample space for experimentation with apartment blocks of different sizes and footprint ratios.

For the purposes of presenting a design to the user in an intelligible format, Shape Evolution can convert the three-dimensional array to a Virtual Reality Mark-up Language (VRML) file. The conversion to VRML happens by reading the values in the three-dimensional array and appending a cubic module of the appropriate type in the right coordinates. This process builds a three-dimensional geometric model that represents the design by use of the abstract forms of the circulation block and apartment shapes. A VRML viewer can display this geometric model on the computer screen as seen from several preset viewpoints. The viewer software also allows the user to freely rotate the model or carry out a virtual walk-through in real time. VRML is a standard file format that can be imported into three-dimensional CAD or analysis software and incorporated in the designer's digital workflow, including the production of physical models using rapid prototyping equipment.

Figure 4.4: The VRML model of a random apartment block design as viewed in a VRML browser

Each three dimensional array is paired with a shape code that describes the rule-based generation of the design. It must be noted, however, that because of the shape code ambiguity issue mentioned in section 3.2.1 above, while the three-dimensional array can be derived from the shape code, the array is not sufficient to produce the shape code. For this reason, it must be ensured that the shape codes for all designs in the current generation are kept in memory throughout the Shape Evolution process.

4.3 Generation of the Initial Population

The genetic algorithm provides a mechanism to search the problem space that requires a starting point: a random initial population must be provided in the first instance. Despite the fact that it is random, the initial population needs to be comprised of designs valid in the apartment block language. In order to generate these designs, a specific, user-defined number of random rules is selected and applied. (In the interest of simplicity, the genetic algorithm implementation in the prototype cannot deal with populations containing genotypes of variable length. All shape codes must therefore include the same number of rules.) However, as mentioned in section 3.2.2 above, for certain shape grammars there exist rule sequences that are invalid. Such is the case with the apartment block shape grammar. A simple example of an invalid sequence in this grammar would be the application of rule 1 twice in sequence, as that would attempt to insert an apartment in exactly the same position twice, which would require overlapping shapes. The initial population generator must therefore be more discerning about how the "random" rule sequences are produced.

Instead of trying to resolve this issue by a knowledge-based system that defines which rules are applicable at a particular point in the sequence, the generator uses the three-dimensional information provided by the array representation to determine whether a rule application would create an invalid design. Notice that the simple rule application algorithm mentioned in the previous section disregards the original state of the changed values in the array. Adding a checking step ensures that all the array positions to be changed had originally a value of 0, i.e. that segment of space was void. The corrected algorithm for the application of rule 1 therefore becomes:

> For each array position with value 1 and coordinates (i, j, k), if the value at position (i, j−1, k) is zero and the value at position (i+1, j−1, k) is zero and the value at position (i+2, j−1, k) is zero and the value at position (i+3, j−1, k) is zero,

then change the value at position (i, j−1, k) to 2, the value at position (i+1, j−1, k) to 3, the value at position (i+2, j−1, k) to 4, and the value at position (i+3, j−1, k) to 5.

This ensures that the rule is not applied if it would require the overlap of shapes but it does not explain what should happen if this rule is not applicable. If this rule is not applicable, clearly a different rule should be tried. If the second rule to be tried is also selected randomly it is possible that a non-applicable rule might be attempted for the same position more than once. To eliminate this, a randomly shuffled stack of all 22 rules is created for each position. The rules then are attempted in sequence: if the first rule on the stack is not applicable the next rule is attempted. When the stack has been exhausted and no rule has been applied it means that no shape can be added to the last circulation unit. Since the last six rules of the apartment block grammar attach further circulation units in each possible direction, the stack may be exhausted only when the latest circulation unit is entirely surrounded by form in all six directions, a rather rare situation. Despite its rarity, this is an occurrence which must be detected, as if this happens before the required number of rules is applied, the design will have a shorter genotype, an anomaly which is not accommodated by the genetic algorithm at the heart of the Shape Evolution prototype. Therefore, the exhaustion of the rule stack triggers the rejection of the current design. The generator then starts from scratch until the initial population is filled with the required number of valid designs.

The corrected rule application algorithm shown above also disallows the application of a rule when the array positions it affects lie outside the preset limits. This ensures that the design is kept within the 64m × 64m × 64m envelope. It also ensures that no shapes are placed on the ground level apart from the entrance, i.e. the lowest of the circulation units in the initial shape.

As each rule is applied, the changes to the design's array representation are made, and the number of the rule that was applied is recorded in a one-dimensional array.

When the required number of rules is applied this one-dimensional array is the shape code for the current design. In other words, during the initial population generation stage three-dimensional arrays and the shape codes are generated in parallel for every design and kept in memory.

4.4 Evaluation Algorithms

The next step in the process is to evaluate the performance of designs in the initial population by comparing them to the user-provided optimisation goals. The evaluation routine is also called at every iteration of the genetic algorithm loop, as shown in Figure 3.3. In terms of programming the evaluation algorithms the task can be reduced to teasing information about a design's physical attributes from the already known data about the design: the three-dimensional array and the shape code.

4.4.1 Apartment Count, Area, and Volume

The shape code alone can provide a large amount of information. It is known that rules 1 to 16 are rules that add an apartment, whereas rules 17 to 22 add a further circulation unit. Counting the number of times a rule in the range 1 to 16 appears in the shape code gives the number of apartments in the design. Given that the initial shape is comprised of two circulation units stacked on top of each other, the number of circulation blocks in the design will be equal to the number of instances of rules 17 to 22 present in the shape code increased by two. For example, take the shape code for the design shown in Figure 4.4:

19 2 13 21 16 7 20 21 2 11 17 1 4 21 1 21 14 5 8 18 21 5 1 17 21 8 13 7 19 6 21 10

In this 32-rule shape code there are 19 rules that add an apartment and 13 rules that add a circulation unit. Therefore this design features 19 apartments and 15 circulation units, including the two in the initial shape.

This is fairly basic information on its own, but it can reveal plenty about the design to which it pertains. For example, given rough costs per apartment and circulation

unit, a value for the cost of the entire building can be approximated. More simply, the floor area of the building can be calculated by multiplying the number of circulation units by 4m × 4m, and adding the number of apartments multiplied by 4m × 16m. For the example shape code used above, this gives a total floor area of (15 × 4m × 4m) + (19 × 4m × 16m) = 1456m^2. This number can be multiplied further by 4m to give the volume of the building in cubic metres.

More usefully, the ratio of the apartment area over the total area can give a rough indication of the circulation efficiency of the building. In the prototype, the evaluation algorithm produces a value for the percentage of apartment units placed out of the total number of modules placed (this latter value can of course be derived by taking the length of the genotype and adding 2. for the two circulation blocks used in the initial shape). Note that while a single circulation block can take a maximum of four apartments around it, a schema which would give a maximum value of 80% apartment units placed, in actual designs that value can only approach 80% asymptotically, as the circulation block used as the building's entrance cannot have any apartments attached to it.

4.4.2 Building Height and Footprint

The shape code can also be used to derive the height of the building. All apartment additions, using rules 1 to 16, happen on the same level as the last placed circulation unit. The same is true for circulation unit additions using rules 17 to 20. Indeed, the only rule that might change the height of the building is rule 21; rule 22 moves to an existing level below. The initial shape alone takes the height of the building to 8m. Starting with that value, and adding 4m every time rule 21 is encountered while subtracting 4m every time rule 22 is used gives the height of the last circulation unit added. However, the maximum value reached in this process is the overall height of the building.

The height can also be derived from the three-dimensional array by simply starting to search the array from the top down for the first non-zero value. Adding 1 to the

k-coordinate of that position and then multiplying by 4m gives the height of the building. The same method can be used to find the extrema of the building in other directions. This information can be used to provide the rectangular footprint of the building, an important piece of data when trying to fit a design on a site.

4.4.3 Views

One way of taking into account the site conditions is by recognising which direction offers the more desirable views. Conversely, a particular direction might face into the blank wall of an existing building or some otherwise unsightly view. For this building the views originate from the living room end of each apartment. It is assumed that the layout of the apartment can be easily adjusted so that the large living room window can be placed on any of the three walls. The view evaluation algorithm starts at each apartment's living room and checks the values of all positions in a straight line from there to the extents of the three-dimensional array along the four horizontal directions. It is taken that an apartment affords views in a particular direction when there is nothing built between the living room and the array boundary at that direction, i.e. the values of all positions from the living room to the boundary are zero. This algorithm returns the number of flats that have unobstructed views in a particular direction, taking into account that a particular flat may allow views in two or three directions. It also returns the number of flats that have no unobstructed views to the site boundaries, meaning that parts of the building are surrounding those apartments' living rooms in all directions.

4.4.4 Balconies

The stacking of apartments on top each other creates opportunities for using the roofs of built structure below as balconies, to be accessed through the glazed wall of the living room. The evaluation algorithm checks for balconies by looking for voids in the four positions adjacent to the living room, and then checking for a non-zero value (non-void, or built form) directly below. The algorithm simply returns the number of apartments that provide opportunities at least for one 4m $\times$ 4m balcony.

4.5 Scoring

Having drawn a set of quantifiable information pertaining to each design in the population, the next step is to rank the designs according to how well they meet the user-defined goals. For the prototype Unix console application, the optimisation goals are input as a sequence of numbers in a text file. It is trivial however to create a script that provides a graphical front end for the user to select the desirable characteristics of the apartment blocks, subsequently translating this information to a text file that will be read by the prototype Shape Evolution application. A graphical front end of that sort, seen below, was created using AppleScript on Mac OS X.

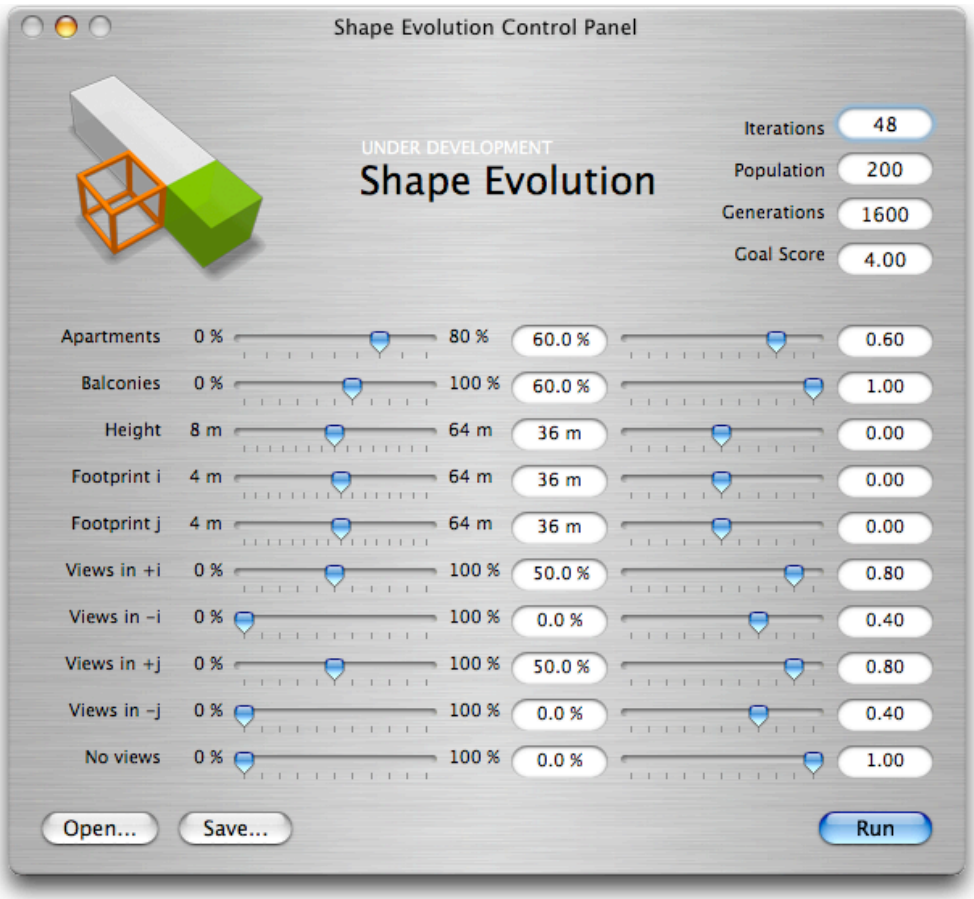


Figure 4.5: The graphical user interface developed for inputting the optimisation goals

This screenshot illustrates the way the user input is structured: for each evaluated quantity or fitness criterion the user inputs a goal value from a range appropriate to that quantity and a weighting in the range [-1, 1]. The weighting determines how

much influence this particular criterion should have on a design's score. Negative values for the weighting result in a higher score for designs that have a measured quantity the *furthest* away from the "goal" quantity; in other words using a negative value for the weighting allow for the specification of undesirable characteristics in the optimised designs.

The score component for a particular criterion is calculated by firstly normalising the measured value of that quantity and the goal to the range [0, 1]. The distance of these two values (i.e. the absolute difference) is then multiplied by the weighting. This gives a normalised and weighted measure of the distance between the measured value and the goal value for each quantity. The total score for each design is found by subtracting the sum of the scoring components for all criteria from the maximum possible score (the sum of all positive weightings). Higher scores denote more desirable designs.

The score S_n for an individual design n is given by the formula below, where S_{max} is the sum of all positive weights, *iterations* is the length of the genotype, variables with the index "goal" are the goal values for quantities revealed by the variable names expressed in percentile points or metres as appropriate, variables with the index "n" are the counts of the features revealed by the variable names in individual design n, variables with the index "weight" are the weights for each criterion, and i_{min}, i_{max}, j_{min}, j_{max}, and k_{max} are minimum and maximum extents of the design n in the directions i, j, and k. The function round(x) returns the integer nearest to x.

$$
S_n = S_{\max} - \left(\begin{array}{l}
\dfrac{\left| \operatorname{round}\left(\dfrac{apartments_{\text{goal}}}{100} iterations \right) - apartments_n \right|}{\lfloor 0.8 iterations \rfloor} apartments_{\text{weight}} + \\
\dfrac{\left| \operatorname{round}\left(\dfrac{balconies_{\text{goal}}}{100} apartments_n \right) - balconies_n \right|}{apartments_n} balconies_{\text{weight}} + \\
\dfrac{\left| \dfrac{height_{\text{goal}}}{4} - \left(k_{\max} + 1 \right) \right|}{14} height_{\text{weight}} + \\
\dfrac{\left| \dfrac{footprinti_{\text{goal}}}{4} - \left(i_{\max} - i_{\min} + 1 \right) \right|}{15} footprinti_{\text{weight}} + \\
\dfrac{\left| \dfrac{footprintj_{\text{goal}}}{4} - \left(j_{\max} - j_{\min} + 1 \right) \right|}{15} footprintj_{\text{weight}} + \\
\dfrac{\left| \operatorname{round}\left(\dfrac{viewsiplus_{\text{goal}}}{100} apartments_n \right) - viewsiplus_n \right|}{apartments_n} viewsiplus_{\text{weight}} + \\
\dfrac{\left| \operatorname{round}\left(\dfrac{viewsiminus_{\text{goal}}}{100} apartments_n \right) - viewsiminus_n \right|}{apartments_n} viewsiminus_{\text{weight}} + \\
\dfrac{\left| \operatorname{round}\left(\dfrac{viewsjplus_{\text{goal}}}{100} apartments_n \right) - viewsjplus_n \right|}{apartments_n} viewsjplus_{\text{weight}} + \\
\dfrac{\left| \operatorname{round}\left(\dfrac{viewsjminus_{\text{goal}}}{100} apartments_n \right) - viewsjminus_n \right|}{apartments_n} viewsjminus_{\text{weight}} + \\
\dfrac{\left| \operatorname{round}\left(\dfrac{noviews_{\text{goal}}}{100} apartments_n \right) - noviews_n \right|}{apartments_n} noviews_{\text{weight}}
\end{array} \right)
$$

It should be noted that this linear addition of score components results in scores that do not fully represent the value of each criterion. A design that satisfies one criterion fully but performs unacceptably poorly with respect to another criterion

might get the same score as other designs with a good balance between criteria. The incorporation of weighting factors for each criterion is meant to compensate for this and give the user the ability to specify priorities. Still, even after normalisation and weighting, the added score components are qualitatively different. The same increase in one component is likely is to have a different impact in another component. Pareto optimisation is a much more effective way to retain the integrity of each criterion in a multi-objective search. This is discussed further in section 5.3.2.

4.6 Selection

The next phase in the genetic algorithm is to select, from the evaluated and scored population, the "parents," i.e. the designs that will contribute genetic material to the next generation. There exist several selection schemes that can be used in genetic algorithms, each applying the probability of selection to its individual with varying degrees of "fairness" or bias (Prügel-Bennett, 2000).

Shape Evolution implements a simple binary tournament selection scheme, chosen for its conceptual simplicity and its capacity for giving a chance to certain low-scoring individuals to be represented in the next population. Two individual designs are picked at random from the population, their scores are compared and the one with the higher score is put in an intermediate population of the same size as the normal population. The process is repeated until the intermediate population is full. Notice that the opponents of each comparison do not become exempt from further fighting. This results in an intermediate population with more copies of the higher scoring designs and fewer or no copies of low scoring designs, thus giving higher reproductive capacity to the fittest individuals.

4.7 Crossover

In Shape Evolution the genotype is a simple string, the shape code. This allows the use of a simple single-point crossover: two parental genotypes are bisected at a random position, and the halves are swapped and rejoined to create the genotypes

for the offspring. Since the tournament selection algorithm already results in a well-shuffled intermediate population, the parental genotypes can just be paired sequentially: the first individual is paired with the second, the third with the fourth, and so on. To avoid unnecessary complications in this step the size of the population is limited to even numbers. (Having set that limitation, if there were a need for odd population sizes, it would suffice to simply copy the last remaining individual over to the new population without crossover.)

With a shape grammar in which all rule sequences are valid a crossover mechanism as already described would be sufficient. However, the apartment block shape grammar presents the problem of invalid shape codes. As mentioned in section 3.2.2, the crossover process generates new, potentially invalid shape codes. To address this issue, some extra elaboration of the crossover algorithm is called for.

After each crossover, the two new offspring designs are therefore checked by being ran through an embryogenesis algorithm that converts the genotypes to three-dimensional arrays containing information about the spatial configuration of the design. Much like the process that generated the initial population, the embryogenesis process checks if an attempted rule application would result in overlapping shapes. At the first instance of that happening it returns an error message. The crossover algorithm then attempts crossover at a new point. To avoid multiple attempts at the same, pathological crossover point, the crossover points are chosen from a shuffled stack of all possible points. If the crossover point stack is exhausted it means that the two parents undergoing crossover cannot produce valid offspring under a single-point crossover. However, this does not mean that they will not be able to produce offspring when paired with other individuals. “Failed” parents can therefore just be copied over to the next population as they are.

4.8 Mutation

The crossover process fills the new population with new valid individuals that are composed of genetic material provided by the parents, the individuals of the

previous generation. To introduce diversity into the new population and allow the exploration of new areas of the problem space mutation is applied probabilistically. Every bit of every genotype in the population is mutated according to a user-defined mutation rate. Mutation actually involves changing the value of a particular bit to a random rule number from 1 to 22.

Once again, this is an operation that changes a genotype to a potentially invalid one, so every time a gene is mutated the validity of the resulting genotype is checked by calling the embryogenesis process. If that process returns an error, i.e. the mutated genotype produces an invalid design, the mutation of the particular bit is simply cancelled.

This introduces a discrepancy between the number of bits that were actually mutated and the user-defined mutation rate. Indeed, the user-defined value actually reflects the number of *attempted* mutations. To allow for a more informed adjustment of that value during modify-and-test loops, the actual mutation rate is recorded at this point and presented to the user at the end of the run of the programme.

4.9 Embryogenesis

As mentioned previously, the embryogenesis process converts the shape codes to the three-dimensional array representation in order to enable the evaluation of designs. Embryogenesis is very similar to the initial population generator. The crucial difference is that it does not produce random genotypes from scratch; it reads the genotypes of the current generation and produces the corresponding three-dimensional array, which is then stored.

Furthermore, as mentioned above, embryogenesis ensures that no malformed individuals are entered into the next generation by checking for validity after the application of each rule. This step may affect the speed of Shape Evolution but ensures usable results.

4.10 Shape Evolution Output

Once the first new generation has been evaluated, its designs are scored, new "parents" are selected, crossover and mutation are applied, and another new generation of designs is produced. This is repeated for either a user-defined number of generations (the end condition used in the prototype), or until a generation is produced with individuals whose score surpasses a user-defined threshold value. The results of the Shape Evolution run are then presented to the user.

For the purposes of analysis, the prototype outputs an HTML document that collects and displays a significant of information about the last Shape Evolution run. Part of this information is the user-inputted variables: target values and weights for all criteria are presented, as well the population size, the number of generations, and the length of the genotype, i.e. the number of shape grammar rules used in each design. The user-defined value for the mutation rate is displayed alongside the actual mutation rate over all generations. The maximum score (a function of the optimisation weights) is also shown.

The prototype code keeps a record of the average and highest scores for every generation. This data is presented on the HTML file as a graph of both these values on the vertical axis over the generations on the horizontal axis. This makes evident the rate of progress of Shape Evolution towards better solutions, permitting a more informed adjustment of parameters if better results are required.

The prototype also records data pertaining to any number of "champion" designs. These are the highest scoring individual designs produced throughout the whole process. The champions' scores as well as the number of the generation in which they appeared are presented on the HTML file, together with a detailed report of their evaluated characteristics. Furthermore, VRML files of the champion designs are generated and linked to from the HTML report. Shape Evolution does not produce *ultimate*, fully optimised solutions, so it would be deceptive to offer a single design at this end of its run. The purpose of the programme is to inspire the designer by

suggesting *good* solutions. Offering a number of alternatives does that, and also allows the designer a choice that can be influenced by criteria that were not dealt with by the Shape Evolution process.

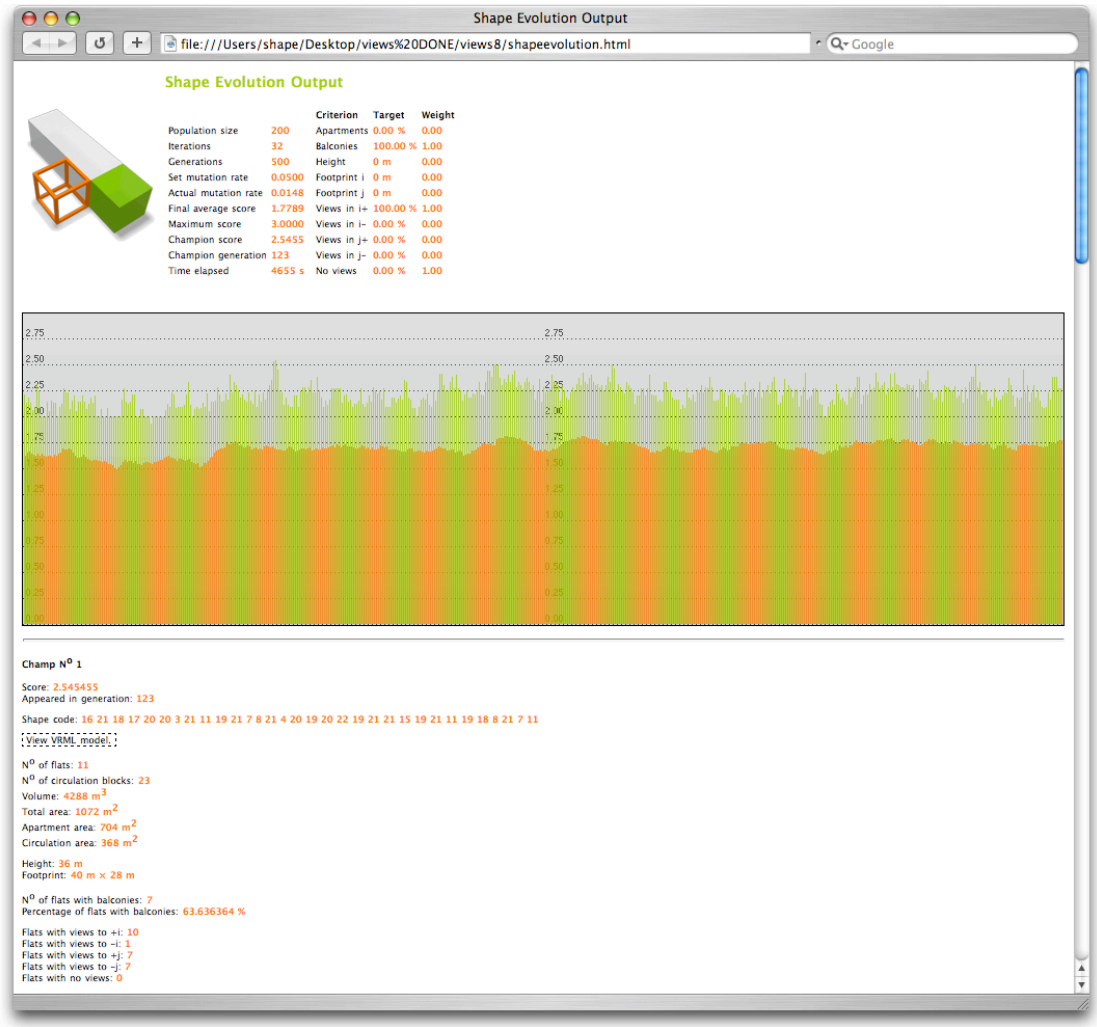


Figure 4.6: The beginning of an example HTML report output by the Shape Evolution prototype viewed in a web browser

5
Experiments and Analysis

5.1 Example Design Intentions

To test the Shape Evolution prototype plausible design situations were devised using a selection of the developed evaluation parameters. Four sets of optimisation goals were used, covering an array of criteria. All four share the common requirement for minimisation of the number of apartments with blocked views in all directions. This is expressed by setting the goal amount of apartments with no views to zero, and the weight of this criterion to 1, the maximum setting. The four sets of design intentions concerned the design of tower blocks, low-rise blocks, buildings maximising opportunities for views and balconies, and buildings satisfying a combination of several criteria.

For each of these design intentions, Shape Evolution was run fifteen times, using a range of different settings for population size and the set mutation rate. Specifically, values of 50, 200, and 500 were used for the population size, and values of 0.005, 0.01, 0.05, 0.1, and 0.5 were used for the set mutation rate. Shape Evolution was allowed to run for 500 generations for the tower, low-rise, and views and balconies problems. 1000 generations were evolved for the multiple criteria problem.

Tower block: The intention is to generate tall and thin buildings. This is encoded by setting the goal value for the building's height to the maximum value, 64 metres. Both dimensions for the footprint are set to 24 metres, equivalent to six cubic modules. The weights for the height and the two footprint criteria are set to 1. The genotype length is set to 48, meaning that 48 shape rules will be applied to the initial shape. The maximum score for designs under these conditions is 4.

Low-rise block: This time the requirement is to generate lower buildings. The goal height is set to 16 metres, equivalent to four storeys. The footprint of the building is left unconstrained. Instead, to avoid unnecessary spreading of the building horizontally, the goal relating to the percentage of apartment modules in the design was set to the maximum value, 80%. The effect of this is that designs with a more efficient use of circulation will be preferred. The genotype length is 24. The weights for height and percentage of apartments are set to 1, making the highest possible score less than 3 (since the maximum value for the percentage of apartments can only be approached asymptotically, as explained in section 4.4.1).

Views and balconies: The building is imagined in a setting with beautiful views in one particular direction. The intention is to exploit the views maximally. This goal is represented by requiring 100% of apartments to afford views towards the positive *i* direction. To further maximise enjoyment of the views, it is required that 100% of apartments have balconies. Weights for the views and balcony criteria are set to 1. The genotype length is set to 32. The maximum score for this set of criteria is 3.

Multiple criteria: The final set of intentions is used to test Shape Evolution in situations were many criteria are called into effect simultaneously. In this case, design goals from the previous test cases are combined. The building should be as high as possible, the percentage of apartments should be as high as possible, and as many apartments as possible should have balconies. The goal values for these three criteria are therefore set to their maximum values, at 64 metres, 80%, and 100% respectively. The goal value for apartments with views in the positive *i* direction is

set to 80% (allowing, perhaps, the 20% of apartments that don't have views in that direction to be sold or rented at a lower price). The weights for the criteria mentioned so far have been set to 1. A less important goal is to give the building an oblong footprint, measuring 24×64 metres in the *i* and *j* directions respectively. The weights for these two criteria have been set to 0.6. This set of criteria seems likely to be satisfied by large buildings; accordingly, the genotype length for this case is set to 64. Given that some of the criteria might be conflicting, and given the impossibility of attaining the apartment percentage criterion, the best scores for this set should be lower than 6.2.

5.2 Results

The results produced by the Shape Evolution prototype for the test cases are collected in appendices A, B, C, and D. This section will discuss the results in general as well as select particular instructive results to focus on.

One of the most telling indicators of the performance of a genetic algorithm is the rate by which the quality of the population improves. This can be visualised by plotting the average score in a population versus generations. Typically, a fast increase in fitness is observed in the first few generations, followed by decelerating progress as the algorithm focuses on a narrow part of the search space. The expected curve can be seen in Figure 5.1 (Parmee and Denham, 1994).

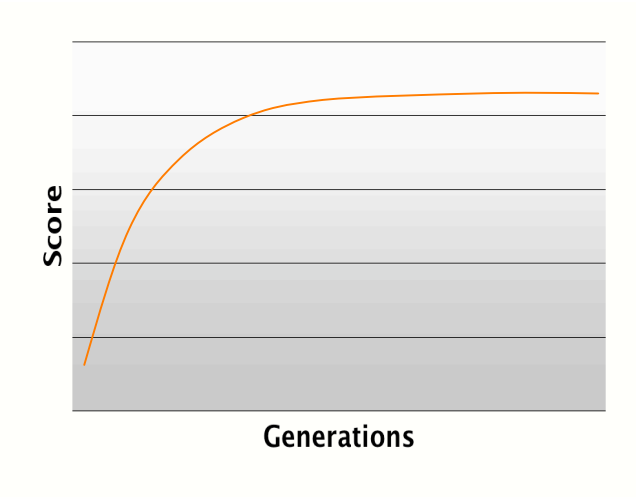


Figure 5.1: Typical curve of score plotted versus generations in genetic algorithm results

In general, the Shape Evolution prototype produced mixed results. While none of the test runs produced dramatically positive results, there were results where the average fitness increased over time in a way similar to the typical curve shown above. There were also cases in which the average fitness clearly *decreased* after a few generations. Since the choice of optimisation criteria greatly affected the capacity of the genetic algorithm to produce consistently improved results, the four different test cases will be examined individually.

5.2.1 Tower Block

The results for the tower block design scenario are presented in appendix A. The requirements for this scenario are fairly simple, asking essentially for a tall, thin building. The footprint and height criteria are not conflicting, and the small footprint is also making it easier for the criterion minimising the number of apartments with no views to be satisfied. As a result, several of the test runs in this scenario produced designs that attained the maximum score value of 4. Some of the maximally scoring designs are shown in Figure 5.2.

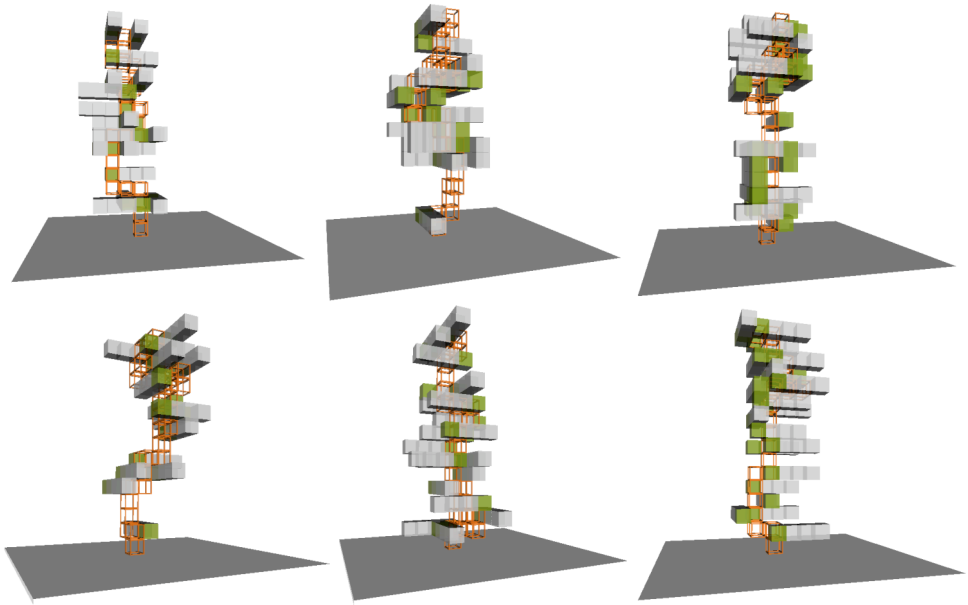


Figure 5.2: Tower designs with maximum score

These building concepts certainly satisfy all the functional design intentions as they were encoded for Shape Evolution. Furthermore, they present novel solutions within the shape grammar defined for this class of designs. In that respect, the tool has been entirely successful: the designer has used a shape grammar and a set of design criteria as input for the programme, and the programme output a range of possible compliant and stimulating designs. Of course, for the purposes of experimentation with the Shape Evolution prototype, aspects of building functionality have not been considered (such as structural concerns, or the vertical continuity of elevator shafts that might be used as part of circulation). Still, the requirements that *were* encoded have been resolved successfully.

Furthermore, the results of this test case display another positive trait: diversity. In most of the runs, the "champion" designs, i.e. the highest-scoring designs of all generations, are very different. This is of course important for Shape Evolution as a tool because its purpose is to open up different routes for the designer to evaluate and be inspired from. This diversity is a result of the conscious strategy to ultimately display as results fit individual designs from all generations, and not just the last one. As seen in the run using a population of 500 and a set mutation rate of 0.05,

the top individuals in adjacent generations might be very similar, stemming from minor mutations of the same ancestral design. The "all-time-champion" system minimises the effect of this phenomenon by also retaining strong designs from the "distant past" with schemata that somehow got extinct in the process.

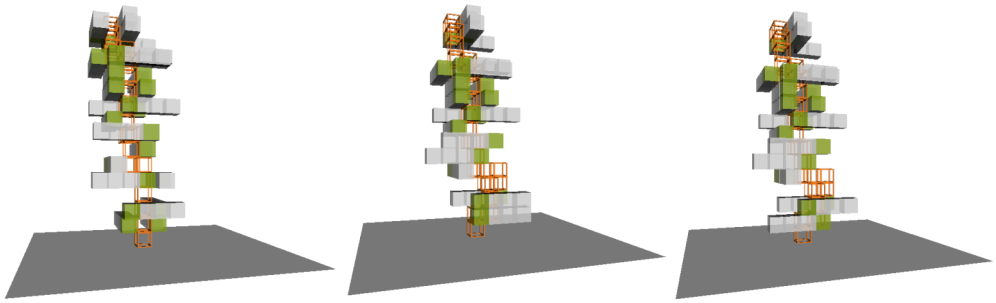


Figure 5.3: The top three tower designs from generations 459, 462, and 461 of the same run, using a population of 500 and a set mutation rate of 0.05, and displaying significant similarities

This kind of output can inspire ideas for further constraining the search of the design space. For example, in order to achieve the height requirement, several of the best tower solutions had floors entirely composed of circulation, which, in most cases, would be undesirable. This undesirability can easily be encoded as an evaluation criterion. Redundant vertical circulation can be detected simply by looking in the genotype for substrings beginning and ending with the allele **21** (the rule that adds a circulation block above the current one) and not containing any of the alleles **1** to **17** (corresponding to rules that attach apartments to circulation blocks). Genotypes that contain such strings can then be penalised during scoring.

The general image of the graphs of average and maximum score versus generations for the tower scenario may be some distance away from the ideal curve shown in Figure 5.1. However, they are still showing a positive increase in average scores over time. There is some fluctuation, especially in the test runs that used the smaller populations. Due to the smaller sample afforded by these small populations, individual designs with extremely high or extremely low scores have a more pronounced effect on the average score. It is therefore expected that the graphs for test runs with larger populations would produce smoother curves.

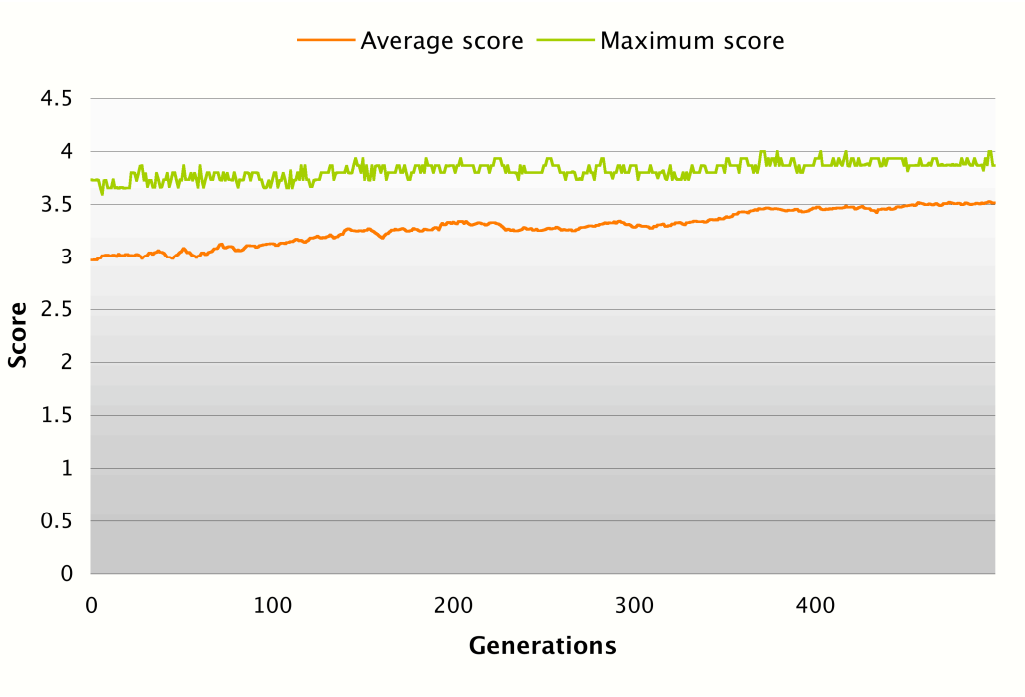


Figure 5.4: The results for the tower problem over 500 generations, with a population of 500, and a set mutation rate of 0.01

Three characteristic curve shapes are observed in the results. The first (Figure 5.4) shows a gradual but steady improvement and is exemplified by the test run using a population size of 500 and a set mutation rate of 0.01. These settings produced designs with the maximum score quite late, with the first ones occurring in generation 370. The increase in average score is also pushing up the maximum score curve. This slow increase in scores is welcome, but it does not entirely meet the efficiency expectations of an optimised genetic algorithm.

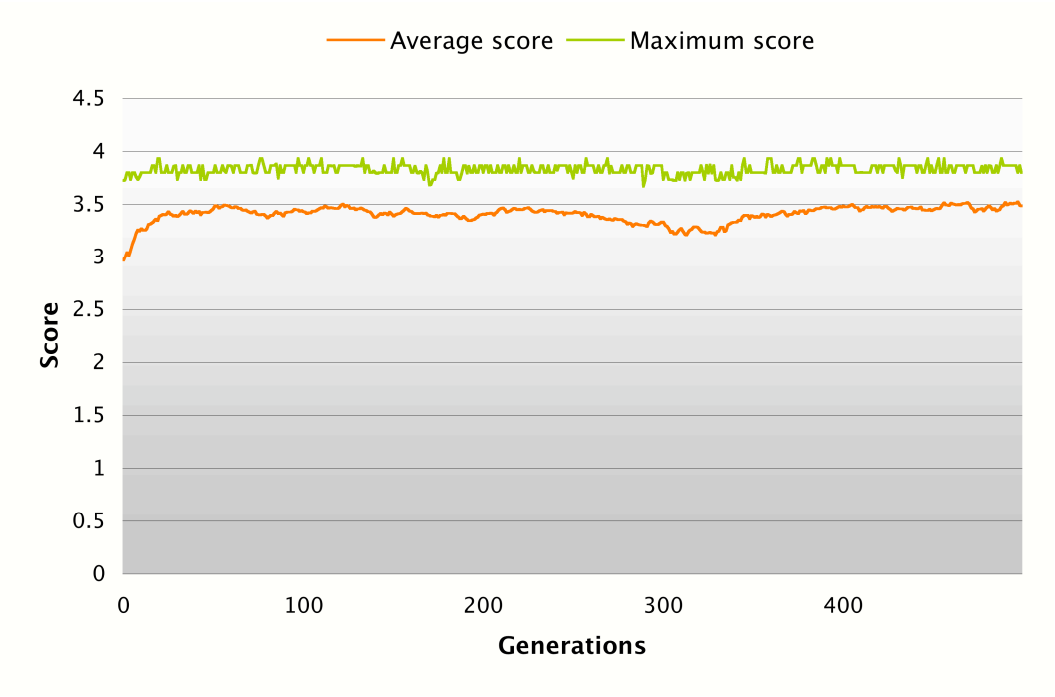


Figure 5.5: The results for the tower problem over 500 generations, with a population of 200, and a set mutation rate of 0.1

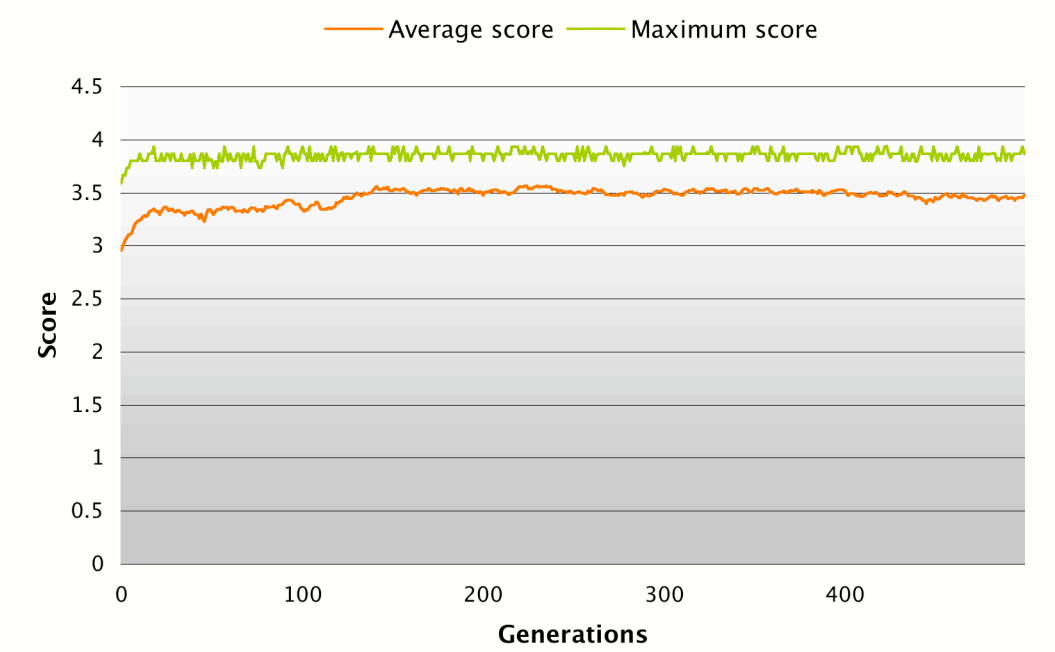


Figure 5.6: The results for the tower problem over 500 generations, with a population of 200, and a set mutation rate of 0.5

The second typical curve shape, observed in the four runs that used populations of 200 and 500 and set mutation rates of 0.1 and 0.5, are much more like the expected

results from a genetic algorithm, and within the first hundred generations approximates the typical curve shown in Figure 5.1. Interestingly, in Figure 5.5 the maximum score remains more or less flat during the dramatic increase in average score in the first few generations, signifying that the algorithm is very effective at weeding out low scoring individuals from the population, but is perhaps less capable at exploiting the genetic material of the top individuals in order to provide better designs. However, this interpretation is negated by the results shown in Figure 5.6, where the highest scoring individual of the original, random generation has a score of only 3.5905 approximately. In this case, the initial remarkable increase in average scores is paralleled by a similar increase in maximum scores. The maximum scores then fluctuate around the value 3.8667 approximately. This fluctuation, as well as the dip in the average score curve seen after generation 300 in Figure 5.5, is the result of the exploration of the solution space performed by the algorithm in search of potentially higher scores: the effect of the failed attempt to find better-performing designs in a different locus.

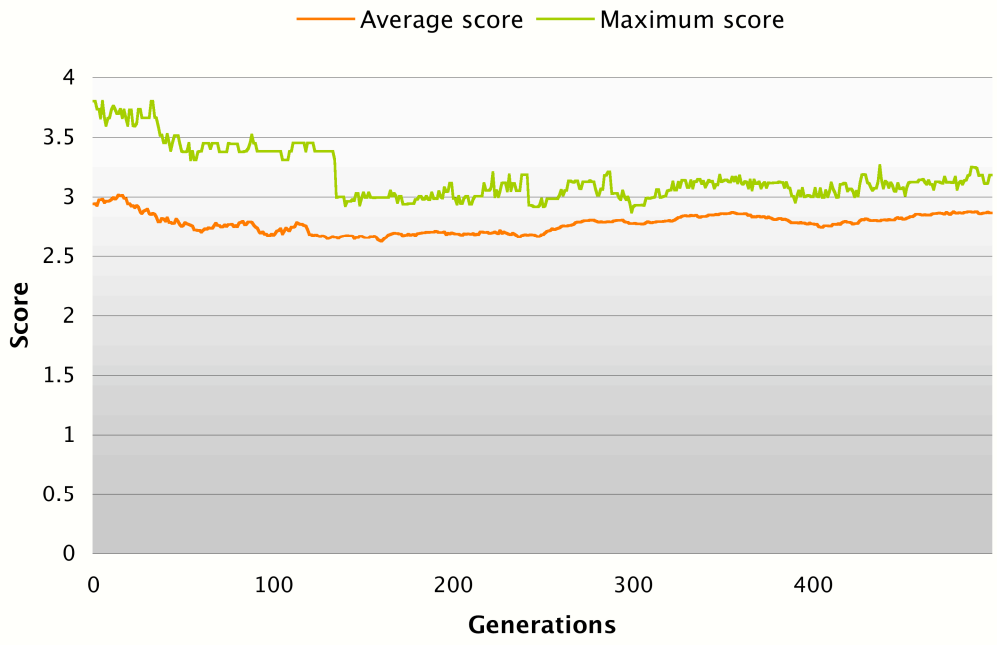


Figure 5.7: The results for the tower problem over 500 generations, with a population of 200, and a set mutation rate of 0.005

The third typical curve shape observed in the tower problem results involves precipitous drops in the maximum scores, as seen in Figure 5.7 for the run with a population of 200 and a set mutation rate of 0.005. In this case, the maximum overall score was 3.8, observed in generation 35. Two dramatic drops subsequently brought the maximum score down to values around 3. These extreme drops in the maximum scores *do* highlight a difficulty in exploiting some high scoring designs. The loss of the best performers in those cases can be caused when a genetic operator, mutation or crossover, produce designs with significantly lower scores. In terms of the solution space, this implies the existence of highly fit and highly unfit designs positioned in neighbouring locations, i.e. that share highly similar schemata in their genotypes. This can be understood intuitively by considering the effect of mutating allele **21** (that moves up one level) to allele **22** (that moves down one level). If that change produces a valid individual, it is likely to produce one with much reduced height, and in the case of the tower scenario, an appreciably lower score.

5.2.2 Low-Rise Block

The low-rise block problem is defined by using competing goals: the building is required to be only four storeys high, yet at the same time 24 rules must be applied, and with the maximum possible amount of apartments. To keep the height of the building down to four storeys, circulation must be placed in order to extend the building horizontally, thus limiting the possible number of apartments. The highest scoring designs produced within this brief featured 17 apartment units and 9 circulation units, giving a value of 65.38% for the percentage of apartment units versus all units placed (in order to maximise the number of flats, the goal value for the percentage of apartment units is set to 80%). To achieve this value, the highest scoring designs sacrifice compliance with the height constraint, adding two more storeys to the goal value of 16 metres. These designs scored 2.75188 out of a maximum score of less than 3.

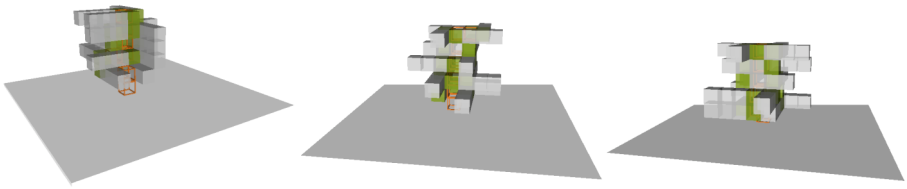


Figure 5.8: Three of the highest scoring designs for the low-rise block scenario, with scores of 2.75188, and featuring 17 apartments in 24 metre high buildings

However, depending on the way the problem is formulated, these designs, though highest-scoring, might not be the most desirable. If the issue is to find the designs that fit the most flats into a building that's only four storeys high (using 24 rule applications), then clearly, the designs that are actually four storeys high are more desirable. In that sense, the "best" design managed to fit 14 apartments versus 12 circulation blocks in a 16m high block, approximately 53.85% apartments. This best design had a score of 2.736842.

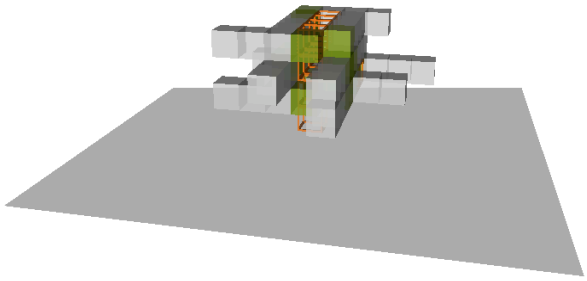


Figure 5.9: The "best" design produced for the low-rise block scenario, scoring 2.736842 with 14 apartments within a height of 16m

Several more of the designs produced scored 2.684211 by fitting 13 apartments within a height of 16 metres. Although these designs have a lower score than the 24 metre high designs, it is more likely that the solutions they provide are more relevant.

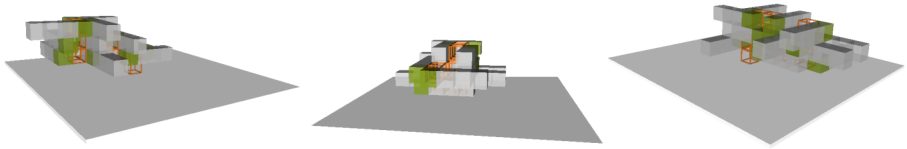


Figure 5.10: Some of the "second best" solutions for the low-rise block scenario, scoring 2.684211 with 13 apartments within a height of 16m

This discrepancy between high scoring and more appropriate designs highlights an important issue with the current prototype implementation, namely, the inadequacy of the scoring system when competing goals are used. With the current model of summing up weighted criteria, the user can attempt to rectify the situation by giving the height criterion a much higher weighting than the apartment percentage criterion. However, trial and error would be the only way to determine the weight differential that ensures the best results. Ideally, Shape Evolution should be capable of determining the designs that provide the best compromise between the two competing scoring components and not allow a design's score to increase by severely undermining compliance with one of the criteria. A solution to this is the use of Pareto optimisation, as discussed in section 5.3.2.

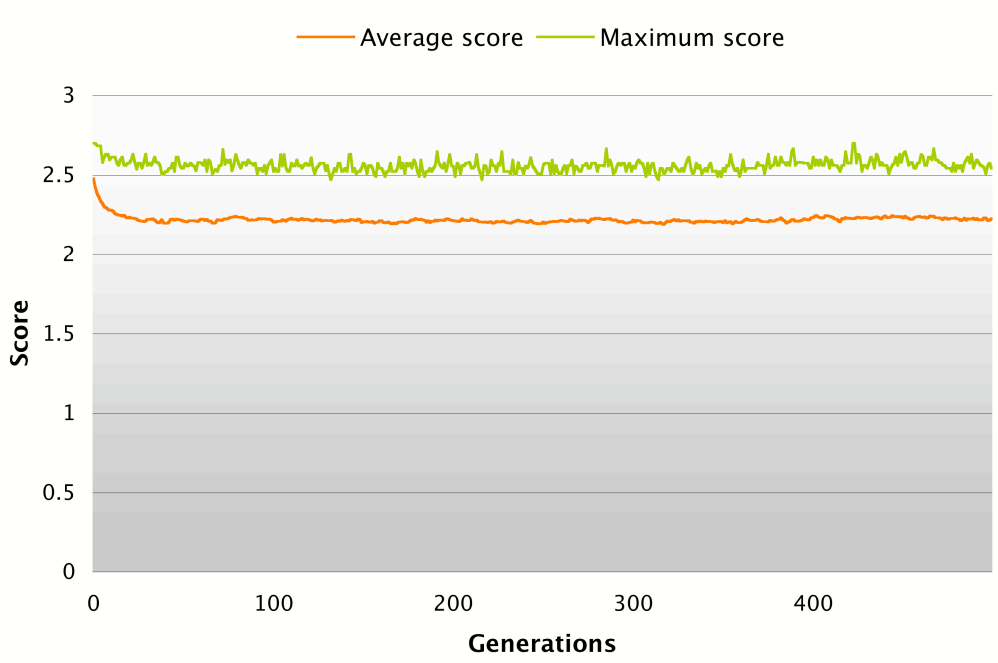


Figure 5.11: The results for the low-rise problem over 500 generations, with a population of 500, and a set mutation rate of 0.5

This scoring issue might also go some way towards explaining the rather unexpected phenomenon of scores *decreasing* over time in this scenario. The curves produced when plotting score against generations are reminiscent of the typical genetic algorithm progress curve seen in Figure 5.1 flipped along the horizontal axis, as seen in the run using a population size of 500 and a set mutation rate of 0.5 (Figure

5.11). The initial random populations have an average score of 2.5. Under the effect of the genetic algorithm the average score curve decreases sharply and slowly settles around the value 2.25. To say that this value represents the algorithm's chosen compromise between the competing goals of horizontal expansion and minimisation of circulation would be misguided, as the evaluation routine has obviously chosen other designs by giving them higher scores. The lower values are the result of the application of the crossover and mutation operators, which, as they are, seem to be letting go of the high-scoring schemata in favour of a wider exploration of the problem space. This suggests that the crossover operator is *disruptive*, i.e. the offspring produced through crossover is likely to perform very differently compared to its parents.

This has little effect on the usefulness of the results produced in this scenario. The maximum score curves in Appendix B do decrease in value. However, this decrease is highly non-monotonic: a look at the maximum score curve in Figure 5.11 reveals fluctuations around the value 2.6, with a constant supply of high-scoring solutions among them (the highest-scoring of all designs showing up as late as generation 425). This does mean, however, that the genetic algorithm is not performing as well as it should. Shape Evolution defaults to a random search that, though it may yield results, is not offering the efficiency promised by the use of the genetic algorithm.

The graphs of the results for the low-rise scenario also serve to show clearly the effect of the population size and the mutation rate on the rate at which the average score curve reaches its stable value. By looking at the graphs for runs using population sizes of 50, 200, and 500, and mutation rates of 0.005, 0.01, 0.05, 0.1, and 0.5 (as presented in Appendix B), some obvious trends can be discerned. Firstly, as noted previously, the increase in population (i.e. sample size) decreases the fluctuations in the average score curve, thus revealing its underlying, basic shape. Secondly, an increase in the mutation rate seems to compress the shape of the curve horizontally, i.e. the stable value is reached more quickly. The increased mutation rate allows the sampling of more new areas of the problem space at every

generation, thus shortening the time required to reach the stable score value. This suggests that mutation not only serves to maintain diversity and prevent the genetic stagnation of the population, but also complements crossover in exploring the solution landscape.

5.2.3 Views and Balconies

The third test scenario is an example of lenient goal requirements. The evaluation function awards score points to a design only on the basis of the percentage of apartments that, on the one hand, afford views in a particular direction (designated as the positive ***i*** direction in the programme) and, on the other hand, have balconies. 32 rule applications are used to generate each design, a rather large number that only slightly complicates things, and certainly does not impede the attainment of the views and balconies goals.

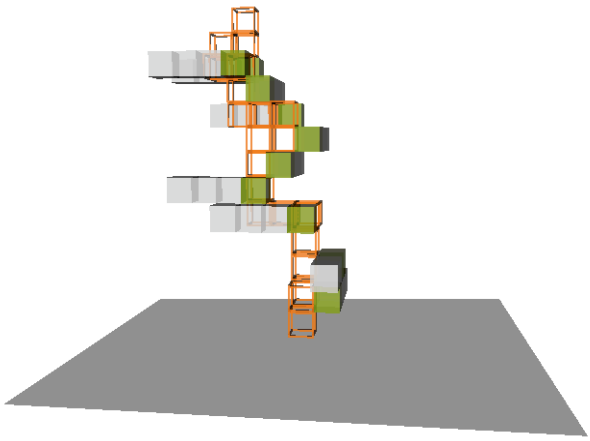


Figure 5.12: The highest scoring design for the views and balconies scenario with a score of 2.7 has 10 apartments, all with views in the desired direction, 70% of which have balconies

The highest scoring design for this scenario has a score of 2.7 and features 10 apartments, all of which have uninterrupted views towards the desired direction. 7 out of the 10 apartments have balconies. This champion design was produced using a population size of 200 and a mutation rate of 0.5. It should be noted, however, that it represents a prominent and solitary spike in the maximum score curve for this run, as seen in Figure 5.13.

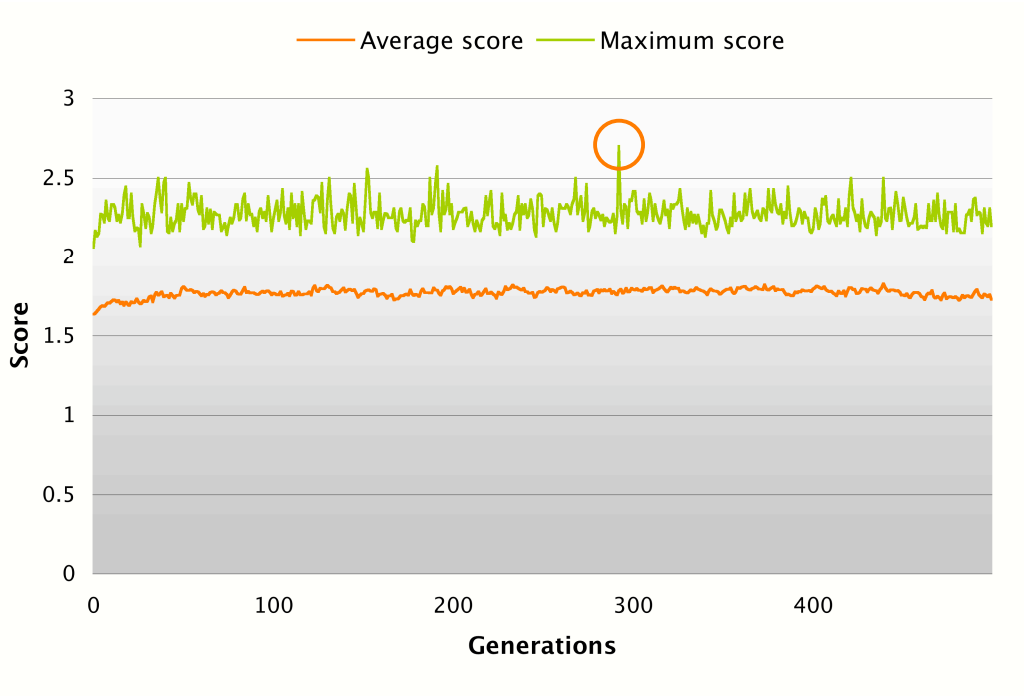


Figure 5.13: The results for the views and balconies problem over 500 generations, with a population of 200, and a set mutation rate of 0.5, showing the spike that produced the champion design

The second highest scoring design, shown in Figure 5.14, was also the result of a spike in the run using a population size of 500 and a mutation rate of 0.05. This design has a score of 2.666667, 8 out of its 9 apartments offer views in the desired direction, and 7 out of 9 apartments have balconies. An interesting feature of this design is an assembly of circulation modules at the top of the building that has no function other than to fill up the required 32 positions in the genotype. Since all three of the criteria employed in this design scenario are percentages of apartments with particular properties (views to *i+*, balconies, no views), the number of apartments and circulation modules used does not affect the score. Indeed, a design with a single apartment with a balcony and the right view would achieve the maximum score. (In fact, depending on how division by zero is handled in the evaluation routine, a building composed entirely of circulation could satisfy all goals and attain the maximum score.) The use of more circulation modules than apartments might actually help produce designs with higher scores, firstly by placing circulation blocks where it can be used to provide balconies for apartments in the

floor above, and secondly by limiting the amount of apartments for which views and balconies should be provided.

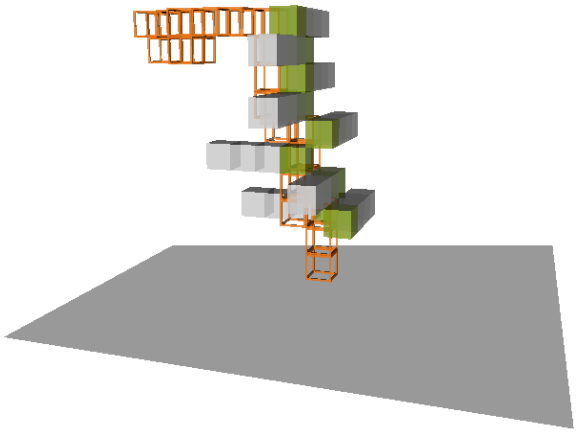


Figure 5.14: The second highest scoring solution for the views and balconies problem, with score 2.666667

The score graphs present a more positive image of the effect of the genetic algorithm. The average scores of the original, random populations is around 1.65. In the four runs where a trend is most clearly discernible (population size 200 with mutation rate 0.5, and population size 500 with mutation rate 0.05, 0.1, and 0.5) the average score rises within the first 50 generations to settle around the value 1.76. That quick initial increase in scores also seems to be affecting the maximum score curve, although that is obfuscated by the violent fluctuations. While in the low-rise scenario the algorithm has a detrimental effect to the average score of the random population, in this case there is a welcome overall increase in scores.

Despite this increase, these results are disappointing in that there were no champions achieving the maximum score, especially given the rather relaxed requirements. There are fairly obvious ways to design a building that completely fulfils the design goals; one example with 14 apartments is shown in Figure 5.15. The fact that high scoring designs are identified as singular spikes in the maximum score curves suggests that the capacity for higher scoring designs exists but the algorithm fails to exploit them. (This can also be evidenced by the fact that, apart from the test runs that used low values for the population size and the mutation

rate, there was no sign of population convergence: there are no obvious common sequences in the genotypes of the final population.) As a result of this failure to exploit good performers, the real-life use of the designs produced in this scenario is likely to be limited to the derivation of inspiration from their formal qualities. It is clear that the performance of the genetic algorithm leaves something to be desired in this case. However, in qualitative terms, the leaning tower archetype, one that supports views and balconies in a single direction, can be recognised in many of the best designs.

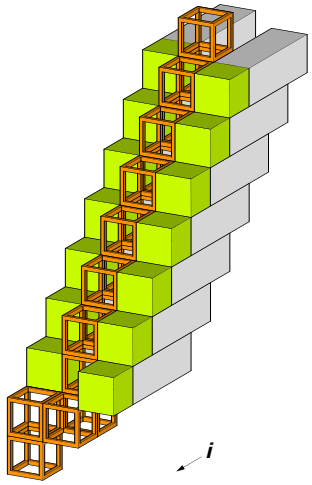


Figure 5.15: A human-designed solution for the views and balconies problem with a perfect score of 3

5.2.4 Multiple Criteria

In the final design scenario for which the Shape Evolution prototype was called to provide solutions there is a combination of the criteria used in the three previous cases. The complexity of the scoring function is matched by the 64-bit genotype. The two highest-scoring designs were both produced in a run using a population size of 500 and a mutation rate of 0.05. Out of all criteria, these top solutions only fully satisfied the height goal of 64 metres. The other criteria are all satisfied at varying degrees, with some of the lower scoring designs doing better in several areas. This is only to be expected when the scoring is subject to a large number of factors that are linearly summed to a single value.

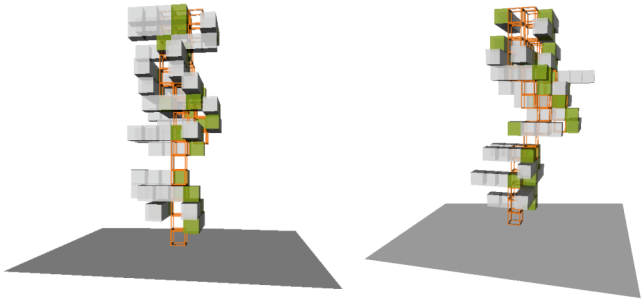


Figure 5.16: The two highest scoring designs produced for the multiple criteria scenario with scores of 5.048235 (left) and 5.036162 (right)

The general trend of the average curves for this scenario is similar to that observed in the views and balconies scenario, at least for the runs with higher values for the population size and mutation rate. There is a quick initial increase in the average score from around 3.8 for a random population to a plateau around 4.1. However, it is very difficult to discern that trend paralleled in the maximum score curves. Indeed, in the runs using smaller populations and lower mutation rates, precipitous drops can be observed in the maximum score curves, signifying the loss of high scoring schemata (similar drops were observed in the results for the tower problem). These drops happen within the first 250 generations and designs of similar scores do not reappear for the remainder of the 1000 generations. In the case of the run using a population of 50 designs and a mutation rate of 0.005 two substantial drops in the maximum score curve can be seen. Importantly, the champion designs in this run appeared at the very beginning, displayed a small increase until the third generation, and subsequently disappeared altogether. Once more, this is linked to disruptive crossover and a difficulty in exploiting high scoring designs.

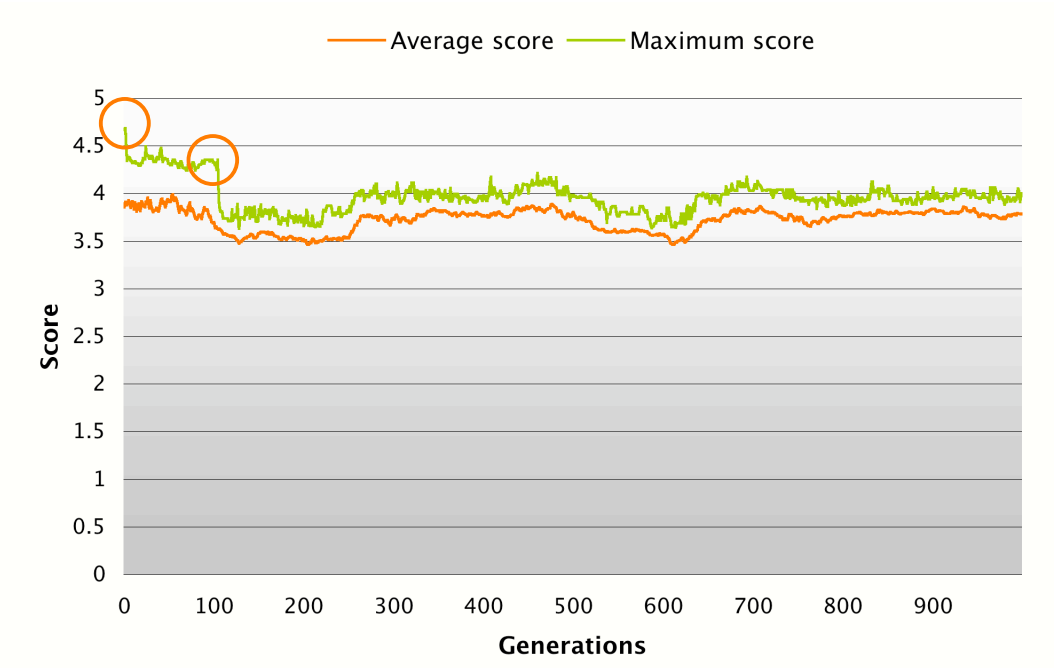


Figure 5.17: The results for the multiple criteria problem over 1000 generations, with a population of 50, and a set mutation rate of 0.005, showing two precipitous drops in the maximum score curve

A glance through the designs presented in Appendix D reveals great complexity and variety. Despite the fact that many of the best designs are far from displaying the desired functional characteristics, they offer designers choice based on their formal characteristics. The most exciting forms can be used by the human designer at the beginning of a design process that elaborates them and alters them to achieve functional compliance.

5.3 Analysis of Results

In summary, these results have shown that the genetic algorithm is not working as efficiently as expected. Two key problem areas have been identified. Firstly, the genetic algorithm seems unable to exploit high scoring individuals. Secondly, the algorithm seems to have difficulty in dealing with opposing criteria. In this section these problems will be explained in detail, their significance will be discussed, and potential ways to improve the behaviour of the Shape Evolution prototype regarding these issues will be suggested as needed.

5.3.1 Exploration Versus Exploitation

The trade-off between exploration and exploitation is a key issue in genetic algorithms. Exploration refers to the ability to sample new areas of the design space in search of better performers, while exploitation refers to the capacity to use the schemata in the genotypes of good performers already in the population to produce better offspring. Exploration and exploitation are thought to be antithetical in nature, representing opposing forces that need to be balanced in order for the genetic algorithm to effectively improve the population (Eiben and Schippers, 1998).

The application of the Shape Evolution prototype to the test scenarios revealed that the genetic algorithm implementation used was deficient in terms of exploiting high scoring individuals. This was made evident by the occurrence of spikes and precipitous drops in the maximum score curves, pointing out that high scoring designs were not retained. (Indeed, the incidence of spikes denotes successful exploratory forays.) This phenomenon can be attributed to the genetic operators, crossover and mutation, both of which can disrupt valuable schemata in the genotypes of good performers. With the apartment block shape grammar used in the prototype, even small changes in the rule application sequence can bring about substantial changes to the phenotype and its evaluated properties. The effect of this could be diminished by reducing the probability of crossover and/or the mutation rate sufficiently in order to find a good balance between exploration and exploitation.

It should be noted that the fact that crossover and mutation have the potential to create invalid individuals should not affect the potential for exploitation. The "controlled" crossover and mutation operators used in the prototype are designed to be conservative, i.e. to maintain the status quo of genotypes that would have otherwise produced invalid genotypes. Therefore, if an operator applied on a high performance design were to produce an invalid individual, that operator application would be cancelled, and the original high performer would be retained.

Another potential culprit for the algorithm's diminished capacity for exploitation would be the selection process. The simple tournament selection employed, as described in section 4.6, may be fair, but it is capable of leaving high scoring individuals behind, simply by not picking them to enter a tournament in the first place. A stronger selection method that ensures that the fittest individuals are represented in the next generation, such as stochastic universal sampling (Prügel-Bennett, 2000), would improve the results of the genetic algorithm by allowing better exploitation. Alternatively, or additionally, elitism (De Jong, 1975) could be employed. Elitism assures that some copies of the best individuals in the population are placed in the next generation, bypassing the normal selection process, as well as crossover and mutation. An instant effect of this would be that the maximum score curves would be monotonically increasing. More importantly, it would ensure that high quality schemata are always available to be used in crossover, allowing them to be better exploited. Additionally, even if the crossover operator is overwhelmingly disruptive, elitism will allow the algorithm to default to a hill-climbing strategy, relying on mutation as the source of variation. When behaving in this fashion, the genetic algorithm would work similarly to a simulated annealing algorithm, but with the added benefit of a massively parallel search.

In all of the test scenarios it was observed that the average score curve would quickly level off at a particular value, especially for large populations and high mutation rates. In most cases, this value was higher than the average score of the initial random population. In the low-rise scenario this value was actually lower, suggesting that the algorithm merely settles at a value it is capable of, instead of constantly optimising for higher scores. This "low potential" state for the average score seems to be the result of heavy exploration of the score landscape combined with minimal exploitation. The value at which the algorithm settles then represents the overall distribution of fitness values in the design space. Different sets of evaluation criteria are likely to produce specific average score values under such an extreme dominance of exploration over exploitation. The solutions suggested above to bring a balance

between exploration and exploitation would allow the genetic algorithm to break free of this low potential state and further improve results.

5.3.2 Multi-Objective Optimisation

The Shape Evolution prototype guides the genetic algorithm using a number of disparate quantifiable criteria, measured in different units. The goal numerical values for all the criteria are restricted to ranges. For example, the percentage of apartment modules in the total number of modules in the design can only lie in the range 0 % to 80%, the highest value occurring when there are four apartments attached to every single circulation block (and then the percentage can only approach 80 % asymptotically, as the initial circulation block, denoting the entrance, cannot have any apartments adjacent to it). Similarly, the height and footprint goals can take values between 4 m and 64 m in discrete 4 m steps because of the nature of the shape grammar used, and the limits in the size of the array. The goals for balconies and views are expressed in percentages of the total number of apartments in the design. In order for the prototype to convert all these criteria into a single score value they are normalised to the range 0 to 1, multiplied by their user-specified weights, then added together.

While this method might be able to produce results that appear to address the specified goals, it is in essence like “comparing apples to oranges” (Goldberg, 1989). As mentioned in section 4.5 above, the linear addition of qualitatively different goals, even after normalisation, buries the significance of each individual criterion. Moreover, as seen in the results it is possible for a design to achieve a high score by entirely failing to address one or more of the requirements and performing very well with others. Choosing appropriate weights for the scoring components is a task that requires a huge amount of trial and error for each set of functional requirements. Designers using Shape Evolution as a tool would be most certainly put off by the effort required.

The use of Pareto optimisation to tackle this problem has already been alluded to in section 5.2.2. Instead of combining non-commensurable quantities in a single scalar value, the fitness of a design can be thought of as a vector in *n*-dimensional space, where the number of dimensions *n* is equal to the number of separate components used. Thus the contribution of each component remains identifiable. Within this framework, Pareto-optimal designs are those for which no further increase in one vector component can be achieved without degradation in at least one of the remaining components (Goldberg, 1989). Obviously, this means that there is no single design representing the optimum; instead the best solution consists of a Pareto-optimal set of designs, representing a selection of acceptable trade-offs between opposing goals (as encountered in the low-rise block scenario). This fits in with the intention for Shape Evolution to be used interactively by a designer who will make the ultimate choice among a collection of good designs presented by the programme. This method makes this choice a much better informed one by encoding knowledge about the relationships of the optimised quantities, and allowing the designer to exploit this information towards the achievement of qualitative goals for the design (Radford et al., 1985). The mechanics of using Pareto optimisation in the context of a genetic algorithm have been suggested by Fonseca and Fleming (1993) in the form of Multiple Objective Genetic Algorithms (MOGAs). Similar frameworks that use Pareto optimisation in design problems have been suggested by Radford et al. (1985) in architectural design and by Matthews (2001) in spatial allocation problems, namely land-use planning.

5.4 Summary

The testing of the Shape Evolution prototype revealed deficiencies in the way the genetic algorithm searches through the design space. Solutions that would effectively eliminate these deficiencies were suggested in the previous section. Despite the less-than-perfect performance of the genetic algorithm (which still managed to generate high performance solutions in many cases), the final designs produced as solutions to the four example design scenarios had formal and

functional merits, and could be easily utilised by the designers as concept designs or merely as sources of inspiration. More importantly for this thesis, it has been shown that the combination of shape grammars and genetic algorithms through the use of a genotype encoding the rule application sequence allows the manipulation and evolution of grammar-generated designs without leaving the grammar-specified language. All of the champion designs produced were valid designs within the language specified by the apartment block shape grammar.

6
Conclusions and Further Directions

6.1 Evaluation of Shape Evolution Prototype

Having performed tests with the prototype, general comments can be made with respect to how Shape Evolution as a method, and its prototypical implementation, have addressed the requirements outlined in the introduction of this thesis. Namely, the goal was to produce a method and a tool that would be usable (i.e. fast and uncomplicated to its intended users), able to suggest functional solutions, and inspiring (by suggesting innovative design ideas). As argued, the use of shape grammars as the representation is theoretically capable of satisfying another stated goal, the ability to produce designs consistent with the designer's stylistic and aesthetic requirements.

6.1.1 Performance

The test runs of Shape Evolution used to provide data for the previous chapter took up to ten hours to complete on a 867 Mhz Apple PowerMac G4 (a low-end computer at the time of this writing). This should by no means be taken as an accurate measure of the time required for each of these runs. The vast majority of these were run in parallel with other applications and tens of other Shape Evolution instances, all sharing the computational resources of the single processor in the test machine. The measured times do, however, give some indication of the general speed of

execution of Shape Evolution. It seems that useful results would require hours of computation, rather than minutes or seconds, on a typical contemporary computer.

While the performance of the Shape Evolution prototype may not be ideal, and a far cry from the envisioned real-time system, the reasons for this lie in the inefficient genetic algorithm code. The coding process was result-oriented, only aiming to produce working code, which inadvertently sacrificed efficiency. For example, there exist several long loops that are largely redundant. Some of these are performed thousands of times per generation. It is expected that recoding the prototype with efficiency and optimised performance in mind by a proficient programmer would drastically accelerate execution of Shape Evolution on existing hardware. Of course, with the development of faster computer hardware (which, at this time, shows no signs of slowing down) Shape Evolution might be able to offer the facility to tweak parameters and monitor their results in real time in the not-so-distant future.

Beyond coding inefficiencies, the speed of the prototype is also affected in a more fundamental way by the inclusion of the controls on the genetic operators. These controls ensure that the changes they attempt on a rule sequence do not result in teratogenesis, i.e. the generation of invalid individuals. It can be observed by the discrepancy between the set mutation rate and the actual mutation rate that as low as 10% of all attempted mutations are valid. This implies that the mutation algorithm and its control mechanism (which involves going through embryogenesis to ensure validity of the mutated genotype) take ten times more processing resources than they would have needed if every mutation were possible. The controlled crossover operator introduces a similar performance penalty. Tests during the development of the prototype have shown that, depending on the design problem and settings, up to 40% of crossover attempts fail altogether, having exhausted all possible crossover points for a given pair of genotypes. That involves calling the embryogenesis routine for validation purposes up to 47 times per individual per generation, for designs using 48-rule-long genotypes (one time for each of the 47 possible crossover

points). It goes without question that using a shape grammar that allows all possible rule sequences would allow Shape Evolution to run at much faster speeds.

Overall, however, the speed of the current prototype is not worrying. With future faster hardware and optimised code it is expected that by the time Shape Evolution has been developed to an extent where its use by professional designers is feasible it might be able to also offer the real-time performance desired.

6.1.2 Usability

Currently, the Shape Evolution prototype is not a user-friendly programme, despite the use of a clearly legible data entry panel (see Figure 4.5) and result report (see Figure 4.6). One of the major issues limiting usability is the fact that it is not initially clear what values should be used for the operating parameters of the genetic algorithm, namely population size and mutation rate. It is also unclear what values should be used as weights for the optimisation criteria (an issue which can be eliminated if Pareto optimisation is used, as suggested above). The optimum values for these parameters are different for each set of optimisation criteria, and can only be derived after extensive experimentation through trial and error. Unfortunately, it is widely recognised that the use of genetic algorithms involves a lot of guesswork in an attempt to find the "sweet spot" of settings that will give the best results.

There have been several studies that aim to suggest the best settings for population size and mutation rate (Mitchell, 1996, Goldberg, 1999), but these are particular to specific kinds of genetic algorithms and are unable to deal with the variety of real-life problems that genetic algorithms are called to tackle. Furthermore, designers cannot be expected to be well-versed in genetic algorithm literature at this level. There is no easy solution to this problem. Of course, designers could employ computer scientists as consultants and to serve as Shape Evolution operators. Alternatively, interested designers might chose to spend some time and acquire some level of experience with the programme that would give them an intuition about which values are likely to work best. But these solutions are far from ideal. It

seems that the most user-friendly way to solve this problem is to transfer the burden of deciding these values from the user to the computer. Davis (1991) has suggested an approach in which the parameters pertaining to the genetic algorithm operators can be adapted throughout the duration of the run in order to achieve better results. This seems to be a sensible route to take when producing genetic algorithms that are meant to be user friendly and generically applicable.

6.1.3 Utility

Without the benefit of a highly optimised genetic algorithm at its core, Shape Evolution produces lower-scoring results, or takes significantly longer to find satisfactory solutions. However, even with a severely crippled genetic algorithm implementation, Shape Evolution is useful on the strength of its generation and evaluation components alone, which can perform a random search of the problem space. Used like that, before the genetic algorithm code for the prototype was completed, the programme still produced evocative, inspiring, and even functional designs. The added benefit of a guided search provided by a well-behaved genetic algorithm is that this search is performed much more efficiently. The designs produced for the example design scenarios during testing did have exploitable properties. Even for the low-rise scenario, where the genetic algorithm's performance was the poorest, champion designs with the required characteristics were produced. Therefore, from the point of view of usefulness to a designer, the tests have shown the Shape Evolution prototype to be successful.

6.2 Shape Evolution in Comparison

With the benefit of having produced and used the Shape Evolution prototype, it would be appropriate to attempt a comparison with other similar systems, as mentioned in sections 2.5 and 2.6. To be meaningful, the scope of the comparison will be limited to recently developed systems for building design with working computer implementations: the room layout tool by Elezkurtaj and Franck (2000), Caldas's building form optimisation programme (2002), GADES (Bentley, 1999),

Eifform (Shea, 2000), and the floor plan generator by Rosenman and Gero (Rosenman, 1997b, Rosenman and Gero, 1999). However, even within this limited selection, the comparison is problematic. The aforementioned systems are in various stages of development and firmly belong in the realm of research. It would therefore be impossible to compile a single "testing suite" of design problems that would be capable of effectively measuring the comparative strengths and weaknesses of each system. Furthermore, an attempt for such an objective test of the robustness of each algorithm against a diverse selection of design challenges is impeded by the specificity of some of the systems and/or their prototypical manifestations. Indeed, it seems that the only kind of comparison not entirely precluded by the diversity of this collection of systems is a qualitative one. However, along with a description of each system's method and the ways is which it diverges from the method followed by Shape Evolution, an assessment can be made of each system's position in the design process and capacity for usable and novel solutions.

6.2.1 Compared to the Programme by Elezkurtaj and Franck

Elezkurtaj and Franck (2000) have presented a software prototype for a system that supports the design of architectural floor plans. The system uses an evolutionary strategy and a genetic algorithm in combination, in order to fit a user-specified set of rooms of particular proportions into a user-specified floor outline. The configuration of the rooms is further constrained by a topological matrix that defines requirements of adjacency between rooms. As it was mentioned in section 2.5 this programme, along with the one by Caldas (2002), comes into the design process after several key elements of the design have been determined by hand. As a result, the input requirements of this system are high: the user must provide the geometrical properties of the rooms and the floor plate as well as the topological relationships between the rooms. The programme uses this input to find the configuration that provides the closest fit to the floor outline. Substantial care was taken so that the software is optimised for speed. This allows the interface to display the evolution of the fittest solution as a plan on screen. It also gives the user the

facility to interact with that evolution in real time by editing the geometry and arrangement of rooms using the mouse, and also to alter the weights of the adjacency constraints. The design process when using this system can then be summarised by the following diagram:

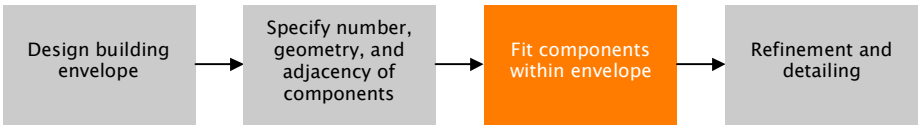


Figure 6.1: Design process when using the programme by Elezkurtaj and Franck; stages represented by an orange box are supported by the software

It should be noted that this system is designed to tackle a very specific, albeit central, sub-problem of architectural design. While many design problems can be reduced to a problem of component configuration, it is clear that this system is still unable to alter the geometry of the building envelope. This severely limits this system when it comes to the exploration of building form, a key consideration in initial design stages. By contrast, Shape Evolution can be introduced to the design process at an earlier stage, when the revelation not immediately obvious novel solutions might have the greatest impact. In addition, Shape Evolution can also be used to solve similar configuration problems by using a shape grammar that features the building envelope as the initial shape and subdivision rules such as those employed by the Palladian grammar (Stiny and Mitchell, 1978). The soft changes to the component geometry could also be incorporated in a Shape Evolution process by using a parametric shape grammar.

The most significant contribution of this work is the real-time interface elements of it. The on-screen graphical manipulation of the solution during the run is an intuitive and powerful way of guiding the evolutionary process. Of course, this is a feature particularly suited to *geometrical* manipulation; altering the product of grammatical generation could not happen in as fluent a fashion. However, the importance of a responsive interface cannot be stressed enough: a tool can only be useful if it's imminently usable. The interface of the Shape Evolution prototype has not been

developed as thoroughly, but it is recognised that real-time user interaction must be part of the finished tool.

6.2.2 Compared to the Programme by Caldas

Caldas (2002) proposed a system that can optimise a schematic building design for its performance with respect to daylighting and energy use. The schematic design to be optimised is represented by pre-arranged three-dimensional blocks whose dimensions can be modulated, in user-constrained ways, by a genetic algorithm. For example, in the case of a house schema that consists of eight blocks in a $2 \times 2 \times 2$ arrangement (four blocks on each of two floors), the point at which all blocks are tangent will remain fixed while the blocks' horizontal dimensions can change; the height of only the top floor blocks can change; the top floor blocks can also feature inclined roofs, with the lowest point of the incline always towards the interior of the building. This schema is fed into the system as a parametric geometric model. The genetic algorithm then generates populations of design variants by attaching different values to these parameters. The genetic algorithm assigns fitness scores to each variant based on calculations regarding energy consumption. After 200 generations, the whole final population of 30 design variants is presented as output. The design process using this system can be summarised as follows:

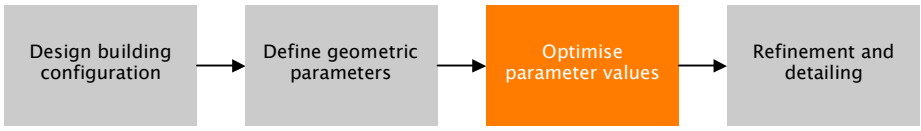


Figure 6.2: Design process when using the programme by Caldas; stages represented by an orange box are supported by the software

This process is very similar to that suggested by the system by Elezkurtaj and Franck. However, instead of deriving new configurations, Caldas limits the scope to the parametric manipulation of geometry, something that the programme by Elezkurtaj and Franck covers as well. In fact, despite Calda's assertion that her system is not an optimisation tool, its reliance on a schematic design as input makes it just that. The system's inability to add geometrical information to that input

precludes its characterisation as a generative system. As Bentley (1999) says, to move from optimisation to the generation of new designs, the system must be capable of modifying every part of the design. As a result, the observations made in the previous section regarding the capacity for the exploration of building form stand in this case as well. Similarly, Shape Evolution could emulate the functionality of this by using a parametric shape grammar. However, Shape Evolution as it stands has a much grater capacity for the production of unanticipated, novel solutions.

6.2.3 Compared to GADES

GADES (genetic algorithm designer) is a system that employs a genetic algorithm for the generation of new designs from scratch (Bentley, 1999). This system uses primitives called "clipped stretched cuboids." Essentially, these are parallelepipeds of variable dimensions and position, which have been sliced by a plane of variable orientation. Such a primitive can be defined by nine parameters. In GADES, designs are composed out of a number of attached clipped stretched cuboids. Similarly to the two systems mentioned so far, the genetic algorithm manipulates form by altering the values of the nine parameters for each primitive. However, GADES goes further by introducing a hierarchical genotype and a mutation operator that can add or delete a primitive. In this way, GADES can start from random blobs that are then shaped by the genetic algorithm using a selection of evaluation modules. Furthermore, the generation can be controlled by the introduction of fixed geometry that cannot be altered during the evolutionary process. GADES has been successfully employed to generate coffee tables, steps, heatsinks, optical prisms, boat hulls, and race cars. In architecture, it has been applied to the problem of the hospital layout, albeit with a few problem-specific additions to the code, in order to guarantee the separation of floors and a fixed building envelope. In effect, for the purpose of proving that GADES can tackle a classic architectural problem, the programme had to be "dumbed down" to attack the problem in the same way that a less capable two-dimensional algorithm would. The power of GADES is most evident when it is called

to generate complex three-dimensional designs such as coffee tables. Then, the design process can be summarised by this diagram:

Figure 6.3: Design process when using GADES; stages represented by an orange box are supported by the software

Indeed, this is an oversimplification; given that the results of the form generation will most probably feed back to the choice of evaluation criteria and goals, that aspect of the design process will be supported by GADES as well.

The key difference between GADES and Shape Evolution is the choice of system of representation for the designs: where GADES uses a completely generic system of geometrical primitives, Shape Evolution relies on a shape grammar. While this allows GADES to be used for any design problem with minimal initialisation, it means that there is little control over the nature of generated form. The choice of shape grammars for Shape Evolution relates to their use of rules and structure, which allow the limitation of the space of formal possibilities to those that adhere to an architectural style. They also allow the composition of designs out of modular components. This is not possible with GADES. On the contrary, it is conceivable that Shape Evolution could emulate this sort of form generation by using a parametric grammar that uses clipped stretched cuboids as shapes. At the same time, Shape Evolution requires a significant amount of time for the design of the shape grammar that will be used, a task that currently is left entirely to the human user. The design process using Shape Evolution can be summarised by the following diagram:

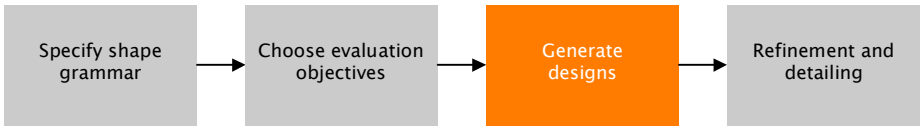


Figure 6.4: Design process when using Shape Evolution; stages represented by an orange box are supported by the software

This process lacks the immediacy afforded by GADES, but enables the designer to control both the stylistic and the functional validity of the generated designs.

A significant benefit of the GADES approach is that its flexible genetic algorithm is already capable of meaningful crossover and mutation using genotypes of variable length. This is of course important for Shape Evolution, because the number of rules employed to solve a particular problem should be an emergent feature of the solution, and not an a priori constraint.

6.2.4 Compared to eifForm

EifForm is a tool more akin to Shape Evolution in that it combines a shape grammar with an evolutionary algorithm (Shea 2000, Shea 2003). As mentioned in section 2.6, eifForm is based on shape annealing, a combination of a shape grammar and simulated annealing. In eifForm, that shape grammar is predefined and fixed and geared towards the generation of triangulated frame structures. The grammar itself is fairly simple: it consists of nine topology transformation rules, which can add and remove structural members, a parametric shape transformation rule, which can move any of the nodal points of the structure in space, and a parametric size transformation rule, which can alter the thickness of each structural member. This grammar allows the generation of single layer free form lattice structures in three dimensions. The user is required to input a three-dimensional surface to which the structural lattice will be projected. In other words, the user specifies the overall three-dimensional form of the structural surface. The simulated annealing portion of the system alters an initial random design slightly and picks the altered design if it performs better than the previous design (the worse design is sometimes picked according to a probability function in order to allow the system to escape local

optima). The designs are evaluated for mass, topological complexity, and structural performance. The design process while using eifForm can be summarised like so:

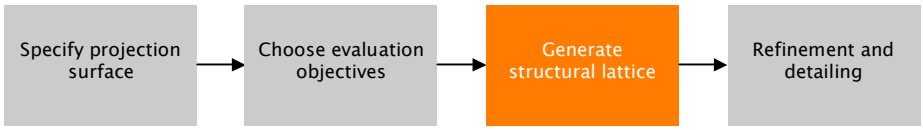


Figure 6.5: Design process when using eifForm; stages represented by an orange box are supported by the software

This system is different to Shape Evolution in two important ways. Firstly, in eifForm there are no provisions for modifying the shape grammar. Its triangulated lattice structure grammar allows the system to generate these kinds of structures alone. While there is no fundamental reason why the system could not use any other shape grammar, the development of eifForm has focused on this specific design problem. Shape Evolution was conceived from the start as a generic design system, and therefore the shape grammar is intended to be user-definable. However, choosing not to be bogged down by the problems relating to the creating of a capable shape grammar parser, this feature was not incorporated in the current Shape Evolution prototype.

The second major difference is that while Shape Evolution performs a massively parallel search by working with populations of designs, eifForm's hill-climbing search algorithm works on one design at the time. At the same time, the stochastic nature of the algorithm means that every run will produce a different solution, even with the same user input. As a result, in order to produce a number of alternative designs to choose from, it is necessary to run eifForm several times. It also means that eifForm is more likely to follow dead-end or low-fitness directions through the search space.

Finally, it should be mentioned that the specification of the projection surface by the user might impede the generation of truly novel structures. However, in its niche of

providing novel solutions for standard structures such as trusses, domes, and transmission towers, eifForm has been proved highly successful.

6.2.5 Compared to the Programme by Rosenman and Gero

The combination of a design grammar and a genetic algorithm is used by Rosenman and Gero for the generation of house floor plans (Rosenman, 1997b, Rosenman and Gero, 1999). In that, it is much like Shape Evolution. However, much like with eifForm, the grammar is used here as a generator of geometry. In this case, the grammar uses edge vectors to generate closed orthogonal polygons, which represent floor plan rooms. The genetic algorithm then conducts a parallel search for floor plans that minimise the perimeter to area ratio and conform to functional adjacency requirements between different rooms. Key to this system is the use of multi-level hierarchical genotypes, which reflect the component assembly nature of the design solutions. House plans are assemblies of rooms/polygons, which in turn are assemblies of edge vectors. This aims at the breaking down of the evolutionary process into discrete sections by hierarchical level. Different fitness functions come to play at the edge vector level and at the room assembly level. The shorter genotypes that result from this allow a more focused and efficient search at each level. The authors assert that this method will have significant advantages for problems where the solutions can be represented as hierarchical assemblies of components. The Shape Evolution prototype uses flat, linear genotypes, but the use of hierarchical genotypes would make sense for even the simple apartment building problem presented in this thesis, as seen in section 6.3.2 below. However, it seems that the added complexity of separate evaluation criteria at each hierarchical level might be problematic: the user would have to input the objectives for each level at the beginning of the process.

Most importantly, the use of a design grammar by Rosenman and Gero does make use of its full potential. There is no interest in preserving the grammatical nature of the generated designs during the evolutionary stage, nor are grammars employed as descriptors of style. It must be acknowledged, however, that the key focus of the

research by Rosenman and Gero is on allowing the genetic algorithm to search more efficiently by segmenting the task into more manageable portions.

6.2.6 Conclusions Drawn from the Comparison

In most cases, this comparison has shown that Shape Evolution is a more general method, even to the extent of being capable of emulating the functionality of more specific methods shown here. While other prototypical implementations of generative design systems have gone further by focusing on particular aspects of algorithms or interface, it seems that a final version of Shape Evolution would be a more useful tool.

6.3 Further Work on the Shape Evolution Prototype

Several improvements can be made to the prototype within its limited scope as a generator of apartment blocks. Some are detailed below.

6.3.1 Site Conditions

Shape Evolution has not been tested on a three-dimensional array that has been pre-initialised with values representing site conditions or existing buildings. However the likelihood of success in tackling site conditions is very high, as merely increasing the shape grammar iterations to an appropriately high number forces the apartment block geometry onto the boundaries of the cubic $16 \times 16 \times 16$ array, squeezing the building into its container, and causing it to take its form, as seen in Figure 6.6. Consequently, it should be trivial to “sculpt” the built form generated by Shape Evolution by forcing it to grow within the void areas in a pre-initialised three-dimensional array. This could be used to eliminate possible conflicts between the new design and existing features on the site, or to force the building to grow within a particular three-dimensional envelope for aesthetic reasons.

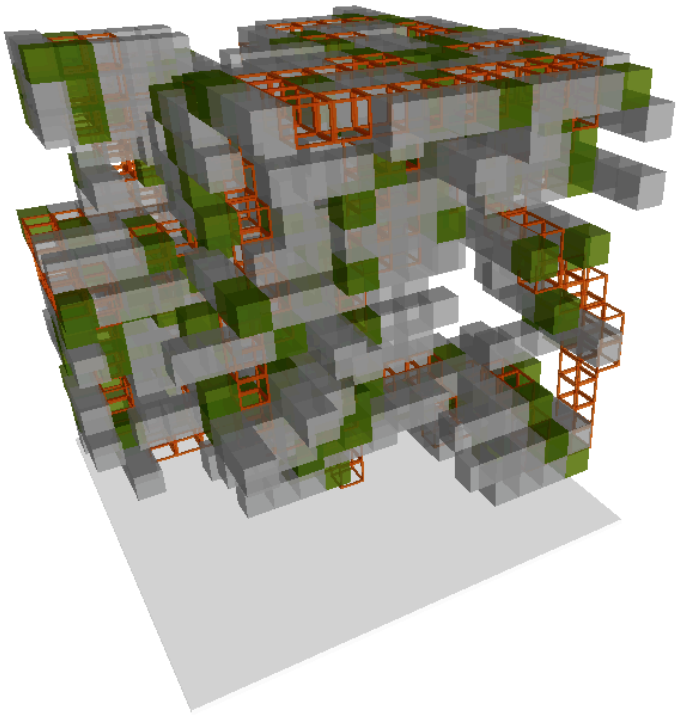


Figure 6.6: An apartment block generated using 512 rule applications taking the shape of a cube by being forced against the boundaries of allowable space

6.3.2 Genotype Elaboration

For the sake of simplicity and for ease of development, the genetic algorithm used in the prototype can only deal with fixed-length genotypes. The length of the genotype, and by extension the amount of modules used in the building, are preset before the beginning of the Shape Evolution run. This is another parameter whose value will largely be up to guesswork. Furthermore, most problems would benefit from the use of a variable-length genotype. This can be evidenced by the design shown in Figure 5.14, which features a large assembly of redundant circulation blocks. The low-rise block designs could benefit too, as the necessity to use a fixed number of rule applications forces the addition of circulation blocks which compromise the attainment of the maximum percentage of apartments requirement. For this reason, it is believed that the algorithm could gain a large amount of flexibility by being rewritten to allow the manipulation of variable length genotypes.

Currently, the Shape Evolution prototype produces buildings that are served by a single circulation route. This single route corresponds to the single string for the

sequence of rules, ensuring contiguity of circulation and accessibility for all flats, as explained in section 3.4. It is obvious that this is not always ideal. Especially for the larger buildings produced, i.e. those with the largest number of grammar rule applications, the circulation routes suggested by shape evolution are entirely impractical. It would be helpful if the largest and more complicated buildings could have two or more ways into them, not to mention the need for fire escape routes.

Still, with the current shape grammar, the rule sequence string ensures the contiguity of the circulation route. A simple way to add further circulation routes would be to use a genotype composed of two or more separate strings, manipulated by the genetic algorithm as a single entity, using a customised crossover operator. That system would produce buildings with as many independent circulation routes as strings, but it would not accommodate the crossing of circulation routes. To achieve that, the genotype should be reformulated as a lattice network, as shown in Figure 6.7. This method still relies on strings of circulation, but allows the sharing of nodes between these strings.

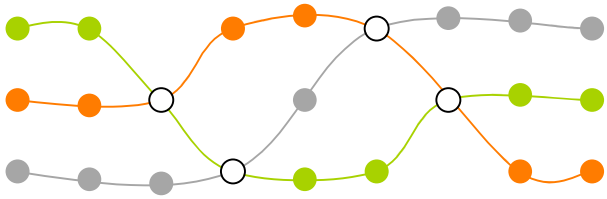


Figure 6.7: Diagram of three strings of circulation crossing and sharing nodes

A further elaboration would require doing away with the concept of strings altogether. The significance of the connections between the bits of the string can be seen when the string is reduced to a chain connecting the rules adding circulation blocks, with the apartment rules attached to the circulation node to which they relate. The rules adding apartments can then be seen as sub-branches of the original chain. To ensure contiguity of circulation it is sufficient for the circulation rules to be connected – it is not necessary for that connection to be linear. With that in mind, a genotype in the shape of a tree would produce a building with a tree of

circulation, i.e. a single entrance and branching routes. Furthermore, a more complicated system of interconnected routes with multiple start and end points could be represented by a genotype in the form of a general network of circulation rules. Again, in this framework, the apartment blocks can be seen as "hanging" off the circulation nodes, up to four per node (an upper limit dictated by the physical characteristics of this particular grammar). Representing them as linear branches would not be appropriate, as their order is of no consequence to the phenotype.

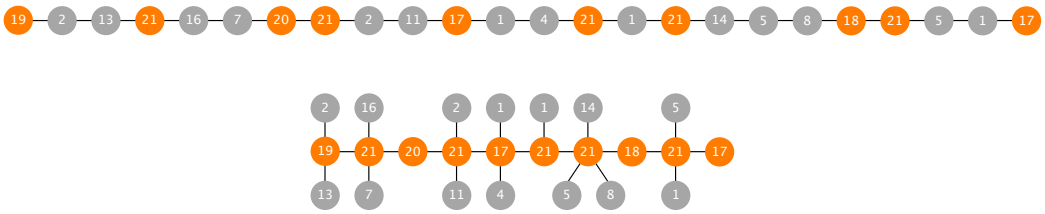


Figure 6.8: A 24-rule genotype from the Shape Evolution prototype and its reinterpretation using a more elaborate topology

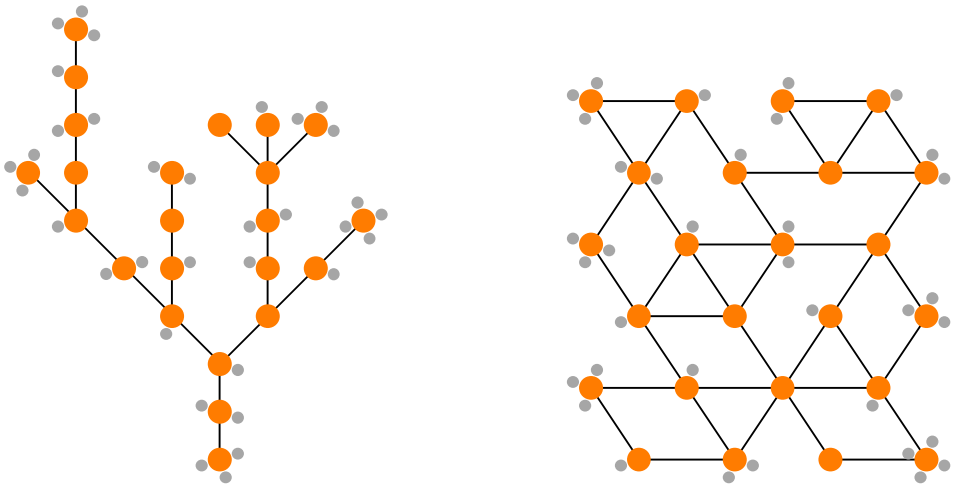


Figure 6.9: Examples of genotypes represented by trees or general networks of circulation units, with the apartments attached to circulation nodes

Obviously, the use of these elaborate genotypes would require new genetic operators, and increase the complexity of the algorithm. However, they have the potential of significantly improving the ability of the genetic algorithm to manipulate the building designs in a meaningful way.

6.3.3 Interface

The user interface of the prototype was put together only to facilitate the development process; no thought has been given to its use by an inexpert designer. The current implementation treats data entry, evolution, and the presentation of output as discrete steps. Indeed, the former is done using a custom control panel window, the second takes place in a UNIX terminal window, and the latter uses a web browser and a VRML plug-in. A more polished version of Shape Evolution intended for public consumption should combine these tasks in a single application environment, perhaps using OpenGL for the three-dimensional representation that is currently done using VRML. In combination with faster operation, this single environment would allow the designer to quickly test the use of different parameters and visually evaluate the results. Real-time operation would enable Shape Evolution to take its place in the designer's workflow as an effective source of inspiration.

When it comes to the specification of evaluation criteria, a more flexible and intelligent system would be desirable. For example, the interface should allow the specification of ranges of values for evaluated quantities as goals, instead of single values. Additionally, the user should be alerted when antagonistic goals have been selected. This can be achieved by encoding a limited amount of knowledge-based information about the likely interactions of the evaluated quantities.

Although the development of an interface of this sort is beyond the scope of this thesis, its availability would allow the testing of Shape Evolution by putting it in the hands of designers. Their feedback would be invaluable in evaluating and improving the interface, as well as the core algorithm itself.

6.4 An Application of Shape Evolution

In the autumn of 2003, the opportunity arose to employ Shape Evolution as a tool for the design of a system for low-cost housing in Chile, as an entry for the *Elemental* competition (see http://www.elementalchile.org/). The entry was produced as a collaboration between architectural teams *End Studio* and *Shaolin 76*. In brief, the

competition called for the design of a housing complex that would place 150 family homes of no more than 30 m^2 each on 1 hectare, at a cost of no more than USD 7,500 per home. In addition, the brief required the provision of quality neighbourhood spaces and the flexibility for future growth.

The *End Studio/Shaolin 76* proposal uses a system of prefabricated panels to form walls, floors, and roofs, by placing them between steel columns arranged in a grid. A rendering of an example housing block constructed using this component system is shown in Figure 6.10. A slightly modified version of the Shape Evolution prototype code was used to suggest economical and functional conglomerations of apartments. Limiting the amount of circulation space required to serve all apartments was one of the primary optimisation goals for Shape Evolution. The customisations to the code of the prototype were mostly changes to the shape grammar rules in order to allow the generation of 30 m^2 apartments. The programme was used to generate a large number of high-performance designs. The most appealing of these were selected and their VRML files as output from Shape Evolution were imported into a three-dimensional modelling software package. The designs were then modified by hand if required, before being further elaborated by the addition of more detail and the application of material textures. Beyond ensuring the production of economical designs, Shape Evolution was capable of creating formally evocative and innovative apartment blocks that went a long way towards satisfying the requirement for quality neighbourhood spaces.

Figure 6.10: An example of a low-cost housing block from the* End Studio/Shaolin 76 *entry to the* Elemental *competition

The teams' use of Shape Evolution did not stop there with this project. As part of the requirement for flexibility for future growth, and to further drop the cost, it was decided to eliminate the architects from the process of extension of the housing project. The architects' role in this proposal would be to come up with a system that would allow the users of the housing to generate further designs and construct them themselves. The simple column grid and component system reduces the requirements for construction expertise for the erection of one of these blocks. When it comes to design, a user-friendly version of the Shape Evolution programme with a limited set of adjustable parameters would be prepared for the use of the inhabitants of the housing project. The generation of new designs and their translation into buildings is thus reduced to a simple, efficient, and economical process.

This suggestion for the use of Shape Evolution was surprising, as it went beyond its intended use as a design tool for the studio. Having said that, this idea is in perfect compliance with DeLanda's and Leach's view of the architect as a controller of processes as opposed to a sculptor of form, as mentioned previously in section 1.4. In this case, this concept is taken up one level of generalisation, with the architect merely specifying a process and surrendering its control to the future users of the designed artifacts.

6.5 Further Work on Shape Evolution

Several improvements to the Shape Evolution prototype have been suggested so far in sections 5.3, 6.1, and 6.3. These include the use of Pareto optimisation to deal with multiple objectives more effectively, the use of more elaborate encoding topologies, and the enhancement of the user interface. However, the purpose of the prototype was merely to provide proof of concept for Shape Evolution as a method. A lot more needs to be achieved for Shape Evolution to become practical as a tool. This section will suggest directions for further research.

From the outset, Shape Evolution was described as a generic method. For the purposes of this thesis the prototype used a hard-coded shape grammar and selection method, as well as hard-coded evaluation functions, genetic operators, and representational frameworks. To produce a tool that can be truly generically applicable all these components must be interchangeable plug-ins. These plug-ins can either be written by advanced users, or prepared as presets that designers can choose from. Plug-in versions of complex analytical software dealing with structure, lighting, acoustics, or costing could thus be employed as part of the evaluation routine. Extensive recoding would be needed to create a core framework that could exploit all these plug-in components. In the meantime, hard-coded prototypes pertaining to a wide variety of examples would be desirable, in order to demonstrate the method's applicability to a wide range of problems.

Another promising route to explore would be the increase of interactivity between Shape Evolution and the designer. As it stands, the user provides input at the beginning of the programme and then waits for the end of the programme to get the results. The designer could be further engaged with the evolution process by being asked to provide feedback on a selection of solutions at various stages along the course of evolution. This could be done by selecting preferred designs (much like the way evolution is guided in Dawkins's (1986) biomorph programme), assigning scores, or ranking designs. In effect, this would be an additional scoring component, providing evolutionary pressure for unquantifiable or ill-defined criteria. This sort of user reinforcement of the evaluation process would also allow the algorithm to waste fewer processing cycles on directions that the designer deems uninspiring or inappropriate.

The introduction of such review points in the course of evolution could also be used as an opportunity to tweak parameters in response to the solutions produced so far. To give an example taken from the Shape Evolution prototype, when asked to generate tall buildings with a small footprint, Shape Evolution often produces apartment blocks with floors that only contain circulation and no apartment units. This is of course not a desirable trait, and it could be easily discouraged by adding an evaluation component that severely penalises designs whose shape code contains none of the rules 1 to 16 in between two instances of rule 21. The ability to perform these changes would rely on a successful development of the plug-in framework mentioned above, as well as a user interface for defining evaluation criteria. In a similar fashion, the primitives and rules of the shape grammar could be altered during the evolutionary process, thus altering the style and character of the produced designs and providing more opportunities for the generation of innovative forms (Knight, 1994).

In terms of usability, it would be very convenient if the specification of the shape grammar could be done by example. The first steps towards algorithms that would be able to automatically derive grammatical rules that would accommodate a given

body of existing designs have already been taken. (Rosenman and Gero, 1999, Rudolph and Alber, 2002). Such a system, if perfected, has the potential to vastly improve the creative capacity of a designer using Shape Evolution.

Finally, it would be extremely inspiring to test the concept of Shape Evolution disciplines other than architecture. Theoretically, as long as a generative grammar can be conceived for a particular kind of work or design, and as long as the desirable qualities of that work or design can be quantified, Shape Evolution should be able to contribute. Disciplines in which Shape Evolution could be applied with satisfactory results include graphic design, industrial design, painting, sculpture, civil engineering, mechanical engineering, automotive and aeronautical engineering, and musical composition.

6.6 Conclusions

By testing it in a variety of scenarios Shape Evolution as a method has been shown to have the capacity to generate functional, unusual, and inspiring designs. Deficiencies in the genetic algorithm that was used in the prototype, and possible ways of eliminating them were identified. However, these deficiencies are secondary. Crucially, it was demonstrated that the combination of a shape grammar with a genetic algorithm for optimisation is feasible and offers a number of important advantages.

Shape Evolution allows for the definition of a design space by using a shape grammar, and only searches for solutions inside this design space. This offers designers a significant amount of control over particular features of generated designs, namely the aspects of design that can be attributed to a particular style. Using computational power for what it can do best, Shape Evolution carries out a massively parallel serendipitous exploration of that design space that can yield high-performance solutions. The high degree of separation between the designer's input and the programme's output means that these high-performance designs are also

unanticipated, a quality that can inspire the designer towards innovative formal solutions.

With the improvements suggested in this chapter, an easy to use Shape Evolution programme would be a valuable addition to many designers' studios. It would allow different design languages, search algorithms, and evaluation routines to be plugged-in. In conjunction with faster, or even real-time operation, these components could be interchanged as needed to generate better results. Incorporating user feedback along the course of the evolutionary process would further allow the optimisation of designs using criteria that cannot be easily formulated or are entirely unquantifiable.

It should be restated that while Shape Evolution is intended to be generically applicable it is not universally recommended. Shape Evolution is merely one tool, one design method among many, and its use will be more appropriate to particular kinds of projects. The low-cost housing project in Chile that was mentioned in section 6.4 was an especially good match for this method, even introducing the use of Shape Evolution as a way to involve communities in the design of their own environment.

Algorithmic design methods are gaining momentum with architects. It is hoped that interest and research on Shape Evolution and other similar design tools will continue in the future, enabling the production of spaces that are not merely economical, but are also functional, innovative, and beautiful, and can delight their users.

References

Alexander, C. (1965). The question of computers in design.*Landscape*, (Spring), pp. 6-8.

Asimow, M. (1962). *Introduction to Design*. Englewood Cliffs, New Jersey: Prentice-Hall.

Auger, B. (1972). *The architect and the computer*. London: Pall Mall Press.

Barnes, M., Dickson, M. and Happold, E. (2000). *Widespan roof structures*. London: T. Telford.

Bentley, P. J. (1996). Generic evolutionary design of solid objects using a genetic algorithm. PhD thesis, University of Huddersfield, Huddersfield.

Bentley, P. J. (1999). From Coffee Tables to Hospitals: Generic Evolutionary Design. **In:** *Evolutionary design by computers* (Bentley, P. J. ed.), pp. 405-422. San Fransisco: Morgan Kaufmann Publishers, Inc.

Cagan, J. and Mitchell, W. J. (1993). Optimally Directed Shape Generation by Shape Annealing. *Environment and Planning B*, **20**, pp. 5-12.

Caldas, L. (2002). Evolving three-dimensional architecture form. **In:** *Artificial Intelligence in Design '02 Proceedings* (Gero, J. S., ed.), pp. 351-370. Cambridge: Kluwer Academic Publishers.

Cawthorne, D. A. and Sparreboom, M. E. (1995). Evolve - A generative, environmental optimisation tool for architects. **In:** *Preprints of the IFIP WG5.2 Formal Design Methods for CAD Conference* (Gero, J. S. and Sudweeks, F., eds.), pp. 51-65. Mexico City: Key Centre of Design Computing, University of Sydney.

Chalmers, J. (1972). The development of CEDAR. **In:** *Proceedings of the international conference on computers in architecture*, pp. 126-140. London: British Computer Society.

Chan, K. H., Frazer, J. H. and Tang, M.-X. (2002). An evolutionary framework for enhancing design. **In:** *Artificial Intelligence in Design '02 Proceedings* (Gero, J. S., ed.), pp. 383-403. Cambridge: Kluwer Academic Publishers.

Chase, S. C. (2000). User interaction in grammar based design systems: from interface analysis to formal models. **In:** *Digital Creativity Symposium Proceedings*, pp. 61-70. London: The University of Greenwich.

Chomsky, N. (1957). *Syntactic structures*. The Hague: Mouton.

Chomsky, N. (1965). *Aspects of the theory of syntax*. Cambridge,: M.I.T. Press.

Christiansen, F. B. and Feldman, M. W. (1998). Algortihms, genetics, and populations: The schemata theorem revisited. *Complexity*, **3** (3), pp. 57-64.

Constantinou, C. (2001). Can computers design? MArch thesis, University of Bath, Bath.

Cross, N. (1977). *The automated architect*. London: Pion.

Davis, L. D. (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

Dawkins, R. (1986). *The blind watchmaker*. 1st American. New York: Norton.

De Jong, K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. PhD thesis, University of Michigan, Ann Arbor.

De Landa, M. (2001). Philosophies of Design: the Case of Modeling Software. **In:** *Verb processing : architecture boogazine* (Salazar, J. ed., pp. 131-143. Barcelona: ACTAR.

De Landa, M. (2002). Deleuze and the Use of the Genetic Algorithm in Architecture. **In:** *Designing for a digital world* (Leach, N. ed., p. 141 p. Chichester: Wiley-Academy.

Dreyfus, H. L. (1972). *What computers can't do; a critique of artificial reason*. New York: Harper & Row.

Duarte, J. (2001). Customising mass housing: A discursive grammar for Siza's Malagueira houses. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA.

Eiben, A. E. and Schippers, C. A. (1998). On evolutionary exploration and exploitation. *Fundamenta Informaticae*, **35**, pp. 35-50.

Elezkurtaj, T. and Franck, G. (2000). Interactive floor plan design supported by an evolutionary strategy and a genetic algorithm. **In:** *Artificial Intelligence in Design '00 Poster Abstracts* (Gero, J. S., ed.), pp. 11-14. Worcester, MA: Key Centre of Design Computing and Cognition.

Fogel, D. B. and Council, I. N. N. (1995). *Evolutionary computation : toward a new philosophy of machine intelligence*. New York: IEEE Press.

Fonseca, C. M. and Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. **In:** *Genetic Algorithms: Proceedings of the Fifth International Conference* (Forrest, S., ed.), pp. 416-423. San Mateo, CA: Morgan Kaufmann.

Frazer, J. (1995). *An evolutionary architecture*. London: Architectural Association.

Funes, P., Lapat, L. and Pollack, J. B. (2000). EvoCAD: evolution-assisted design. **In:** *Artificial Intelligence in Design '00 Poster Abstracts* (Gero, J. S., ed.), pp. 21-24. Worcester, MA: Key Centre of Design Computing and Cognition.

Funes, P. and Pollack, J. (1999). Computer Evolution of Buildable Objects. **In:** *Evolutionary design by computers* (Bentley, P. J. ed., pp. 387-403. San Fransisco: Morgan Kaufmann Publishers, Inc.

Gips, J. (1979). Artificial Intelligence. *Environment and Planning B*, **6**, pp. 353-364.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, Mass.: Addison-Wesley Pub. Co.

Goldberg, D. E. (1999). The Race, the Hurdle, and the Sweet Spot. **In:** *Evolutionary design by computers* (Bentley, P. J. ed., pp. 105-118. San Fransisco: Morgan Kaufmann Publishers, Inc.

Heisserman, J., Callahan, S. and Mattikalli, R. (2000). A design representation to support automated design generation. **In:** *Artificial Intelligence in Design 2000* (Gero, J. S., ed.), pp. 545-566. Worcester, MA: Kluwer Academic Publishers.

Hersey, G. L. and Freedman, R. (1992). *Possible Palladian villas : plus a few instructively impossible ones*. Cambridge, Mass.: MIT Press.

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

Johnson, S. (2001). *Emergence : the connected lives of ants, brains, cities, and software*. New York: Scribner.

Kahn, L. ed. (1973). *Shelter*. Bolinas, California: Shelter Publications.

Kanal, L. and Cumar, V. eds. (1988). *Search in artificial intelligence*. Spring-Verlag.

Knight, T. W. (1981). Languages of designs: from known to new. *Environment and Planning B*, **8**, pp. 213-238.

Knight, T. W. (1994). *Transformations in design : a formal approach to stylistic change and innovation in the visual arts*. Cambridge ; New York: Cambridge University Press.

Knight, T. W. (1999). Shape Grammars in Education and Practice: History and Prospects. *International Journal of Design Computing*, **2**.

Knight, T. W. (2003). Computing with emergence. *Environment and Planning B*, **30**, pp. 125-155.

Koutamanis, A. (2000). Representations from generative systems. **In:** *Artificial Intelligence in Design '00* (Gero, J. S., ed.), pp. 225-245. Dordrecht: Kluwer.

Lawson, B. (1990). *How Designers Think: The Design Process Demystified*. 2nd. London: Butterworth Arthitecture.

Leach, N. (2002). *Designing for a digital world*. Chichester: Wiley-Academy.

Leach, N., Williams, C. and Turnbull, D. (2003). *Digital tectonics*. Chichester: Wiley.

Matthews, K. B. (2001). Applying Genetic Algorithms to Multi-objective Land-Use Planning. PhD thesis, The Robert Gordon University, Aberdeen.

McGill, M. C. (2001). A Visual Approach for Exploring Computational Design. SMArchS thesis, Massachusetts Institute of Technology, Cambridge, MA.

Mitchell, M. (1996). *An introduction to genetic algorithms*. Cambridge, Mass.: MIT Press.

Mitchell, W. J. (1990). *The logic of architecture : design, computation, and cognition*. Cambridge, Mass.: MIT Press.

Mitchell, W. J. (1999). CAD/CAM and the web at the frontiers of architectural practice. **In***: Design Computing on the Net '99 Proceedings*. http://www.arch.usyd.edu.au/kcdc/conferences/dcnet99/index.html.

Moseley, L. (1963). A rational design theory for the planning of buildings based on the analysis and solution of circulation problems. *The Architects' Journal*, 11 November 1963, pp. 525-537.

Negroponte, N. (1967). URBAN 5: An on-line urban design partner. *Ekistics*, **24**, pp. 289-291.

Negroponte, N. and Groisser, L. (1970). URBAN 5: a Machine that Discusses Urban Design. **In:** *Emerging Methods in Environmental Design and Planning* (Moore, G. T. ed., pp. 105-114. Cambridge, MA: MIT Press.

Newell, A., Shaw, J. C. and Simon, H. A. (1967). The Process of Creative Thinking. **In:** *Contemporary approaches to creative thinking* (Gruber, H., Terrell, G. and Wertheimer, M. eds.), pp. 63-119. New York: Atherton Press.

O'Reilly, U.-M. and Ramachandran, G. (1998). A preliminary investigation of evolution as a form design strategy. **In:** *Artificial Life VI* (Adami, C., Belew, R., Kitano, H. and Taylor, C., eds.). Los Angeles.

Parmee, I. C. and Denham, M. J. (1994). The integration of adaptive search techniques with current engineering design practice. **In:** *Proceedings of the Second International Conference on Adaptive Computing in Engineering Design and Control '94*, pp. 1-13. Plymouth: PEDC.

Perrella, S. ed. (1998). *Hypersurface Architecture*. London: John Wiley & Sons.

Pinker, S. (1997). *How the mind works*. 1st. New York: Norton.

Prügel-Bennett, A. (2000). Modelling Evolving Populations. [WWW] http://www.isis.ecs.soton.ac.uk/isystems/evolutionary/notes/evol/evol.html (11 October 2002).

Radford, A. D., Gero, J. S., Rosenman, M. A. and Muthucumaru, B. (1985). Pareto optimization as a computer-aided design tool. **In:** *Optimization in Computer-Aided Design* (Gero, J. S., ed.). Elsevier Science Publishers B.V.

Rahim, A. (2000). *Contemporary processes in architecture*. London ; New York: Wiley-Academy.

Rahim, A. (2002). *Contemporary techniques in architecture*. London ; New York: Wiley-Academy.

Rittel, H. W. J. and Webber, M. M. (1973). Dilemmas in a general theory of planning. *Policy Sciences*, **4**, pp. 155-169.

Rosenman, M. A. (1997). The Generation of Form Using an Evolutionary Approach. **In:** *Evolutionary algorithms in engineering applications* (Dasgupta, D. and Michalewicz, Z. eds.), pp. 69-85. Southampton and Berlin: Springer Verlag.

Rosenman, M. A. (1997). An exploration into evolutionary models for non-routine design. *Artificial Intelligence in Engineering*, **11** (3), pp. 287-293.

Rosenman, M. A. and Gero, J. S. (1999). Evolving Designs by Generating Useful Complex Gene Structures. **In:** *Evolutionary design by computers* (Bentley, P. J. ed., pp. 345-364. San Fransisco: Morgan Kaufmann Publishers, Inc.

Rowbottom, A. (1999). Evolutionary Art and Form. **In:** *Evolutionary design by computers* (Bentley, P. J. ed., pp. 261-277. San Fransisco: Morgan Kaufmann Publishers, Inc.

Rowe, P. G. (1987). *Design thinking*. Cambridge, Mass.: MIT Press.

Rudolph, S. and Alber, R. (2002). An evolutionary approach to the inverse problem in rule-based design representations. **In:** *Artificial Intelligence in Design '02 Proceedings* (Gero, J. S., ed.), pp. 329-350. Cambridge: Kluwer Academic Publishers.

Shea, K. (2000). Generating rational free-form structures. **In:** *Digital Creativity Symposium Proceedings*, pp. 119-128. London: The University of Greenwich.

Shea, K. (2003). Directed Randomness. **In:** *Digital tectonics* (Leach, N., Williams, C. and Turnbull, D. eds.), Chichester: Wiley. Forthcoming.

Shea, K. and Cagan, J. (1997). Innovative dome design: Applying geodesic patterns with shape annealing. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **11**, pp. 379-394.

Shea, K. and Cagan, J. (1999). The design of novel roof trusses with shape annealing: assessing the ability of a computational method in aiding structural designers with varuing design intent. *Design Studies*, **20**, pp. 3-23.

Shea, K. and Cagan, J. (1999). Languages and semantics of grammatical discrete structures. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, **13**, pp. 241-251.

Simon, H. A. (1971). Style in design. **In:** *Proceedings of the 2nd Annual Conference of the Environmental Design Research Association*, pp. 1-10. Pittsburgh, PA: Carnegie Mellon University.

Simon, H. A. (1973). The Structure of Ill-structured Problems. *Artificial Intelligence*, **4**, pp. 181-200.

Simon, H. A. (1996). *The sciences of the artificial*. 3rd. Cambridge, Mass.: MIT Press.

Sims, K. (1994). Evolving 3D Morphology and Behavior by Competition. **In:** *Artificial Life IV* (Brooks, R. and Maes, P., eds.), pp. 28-39. MIT Press.

Sims, K. (1994). *Evolving Virtual Creatures*. **In:** *SIGGRAPH '94 Proceedings*, pp. 15-22.

Stiny, G. (1975). *Pictorial and formal aspects of shape and shape grammars*. Basel ; Stuttgart: Birkhauser.

Stiny, G. (1994). Shape rules: closure, continuity, and emergence. *Environment and Planning B*, **21**, pp. S49-S78.

Stiny, G. (1998). New ways to look at things. *Environment and Planning B*, (Anniversary Issue), pp. 68-75.

Stiny, G. and Gips, J. (1972). Shape Grammars and the Generative Specification of Painting and Sculpture. **In:** *Proceedings of IFIP Congress 71* (Freiman, C. V., ed.), pp. 1460-1465. Amsterdam: North-Holland.

Stiny, G. and March, L. (1981). Design machines. *Environment and Planning B*, **8**, pp. 245-255.

Stiny, G. and Mitchell, W. J. (1978). The Palladian grammar. *Environment and Planning B*, **5**, pp. 5-18.

Tanaka, H. and Kiriyama, T. (2000). An interactive evolutionary system for generating spatial structures. **In:** *Artificial Intelligence in Design '00 Poster Abstracts* (Gero, J. S., ed.), pp. 59-62. Worcester, MA: Key Centre of Design Computing and Cognition.

Tapia, M. (1999). A visual implementation of a shape grammar system. *Environment and Planning B*, **26**, pp. 59-73.

Tapia, M. (2000). The shape docking problem. **In:** *Digital Creativity Symposium Proceedings*, pp. 17-26. London: The University of Greenwich.

Th'ng, R. and Davies, M. (1975). SPACES. *Computer Aided Design*, **7** (2), pp. 112-118.

Todd, S. and Latham, W. (1992). *Evolutionary art and computers*. London ; San Diego, CA: Academic Press.

Todd, S. and Latham, W. (1999). The Mutation and Growth of Art by Computers. **In:** *Evolutionary design by computers* (Bentley, P. J. ed., pp. 221-250. San Fransisco: Morgan Kaufmann Publishers, Inc.

Wade, J. W. (1977). *Architecture, Problems, and Purposes*. New York: John Wiley & Sons.

Wang, X., Tang, M.-X. and Frazer, J. (2000). Creative stimulator: an interface to enhance creativity in design. **In:** *Artificial Intelligence in Design '00 Poster Abstracts* (Gero, J. S., ed.), pp. 69-72. Worcester, MA: Key Centre of Design Computing and Cognition.

Watanabe, M. S. (2002). *Induction design: A method for evolutionary design*. Basel: Birkhäuser.

Whitehead, B. and Eldars, M. Z. (1964). An approach to the optimum layout of single-storey buildings. *The Architects' Journal*, 17 June 1964, pp. 1373-1380.

Willoughby, T. (1970). Computer aided design of a university campus. *The Architects' Journal*, 25 March, pp. 753-758.

Willoughby, T., Paterson, W. and Drummond, G. (1970). Computer-aided architectural planning. *Operational Research Quarterly*, **21** (1), pp. 91-98.

Wolfram, S. (2002). *A new kind of science*. Champaign, IL: Wolfram Media.

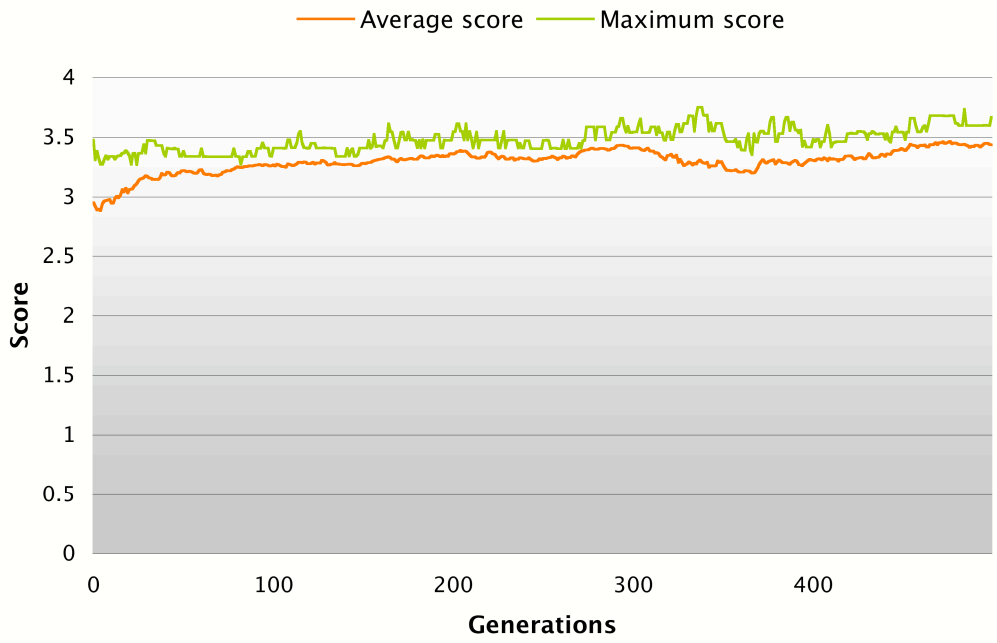
Woodbury, R. F. (1993). A Genetic Approach to Creative Design. **In:** *Modeling creativity and knowledge based creative design* (Gero, J. S. and Maher, M. L. eds.), pp. 211-232. New Jersey: Lawrence Erlbaum Associates.

Appendix A
Results for Tower Block Scenario

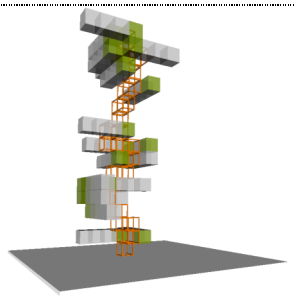
This appendix, and the following three appendices (B, C, and D) collect the detailed results of the Shape Evolution test runs using the tower, low-rise, views and balconies, and multiple criteria problems, as described in chapter 5. The results for each test instance are presented on a single page and include a graph of maximum and average scores versus generations as well as images and attributes of the three top-scoring designs produced during each run.

For the tower problem the goal value for the building's height is set to the maximum value, 64 metres. Both dimensions for the footprint are set to 24 metres, equivalent to six cubic modules. The weights for the height and the two footprint criteria are set to 1. The genotype length is set to 48, meaning that 48 shape rules will be applied to the initial shape. The maximum score for designs under these conditions is 4.

Tower Population: **50** Set mutation rate: **0.005**



Actual mutation rate: **0.0012** Final average score: **3.4356** Time elapsed: **322 s**



Champ №1

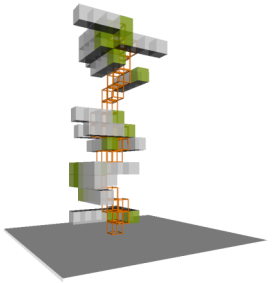
Shape code: **16 7 20 21 19 21 19 6 21 18 6 9 21 5 9 6 17 21 17 21 12 10 20 21 7 16 19 19 21 11 17 21 21 20 21 20 16 21 10 21 2 20 19 5 2 21 3 2**

Score: **3.75**
Appeared in generation: **340**

№ of apartments: **20**
№ of circulation blocks: **30**
Volume: **7040 m³**
Total area: **1760 m²**
Apartment area: **1280 m²**
Circulation area: **480 m²**
Height: **64 m**
Footprint: **28 m × 32 m**

№ of apartments with balconies: **1**
Percentage of apartments with balconies: **5 %**

Apartments with views to +i: **7**
Apartments with views to −i: **12**
Apartments with views to +j: **10**
Apartments with views to −j: **6**
Apartments with no views: **1**



Champ №2

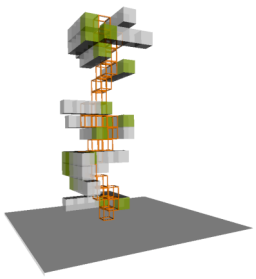
Shape code: **16 7 20 21 19 21 19 6 21 18 6 9 21 5 9 6 17 21 17 21 12 10 20 21 7 3 19 19 21 11 17 21 21 20 21 20 16 21 10 21 2 20 19 5 2 21 3 2**

Score: **3.75**
Appeared in generation: **340**

№ of apartments: **20**
№ of circulation blocks: **30**
Volume: **7040 m³**
Total area: **1760 m²**
Apartment area: **1280 m²**
Circulation area: **480 m²**
Height: **64 m**
Footprint: **28 m × 32 m**

№ of apartments with balconies: **4**
Percentage of apartments with balconies: **20 %**

Apartments with views to +i: **8**
Apartments with views to −i: **11**
Apartments with views to +j: **10**
Apartments with views to −j: **6**
Apartments with no views: **1**



Champ №3

Shape code: **16 7 20 21 19 21 19 6 21 18 6 9 21 5 9 6 17 21 17 21 12 10 20 21 7 16 19 19 21 11 17 21 21 20 21 20 16 21 10 21 2 20 19 5 2 21 12 2**

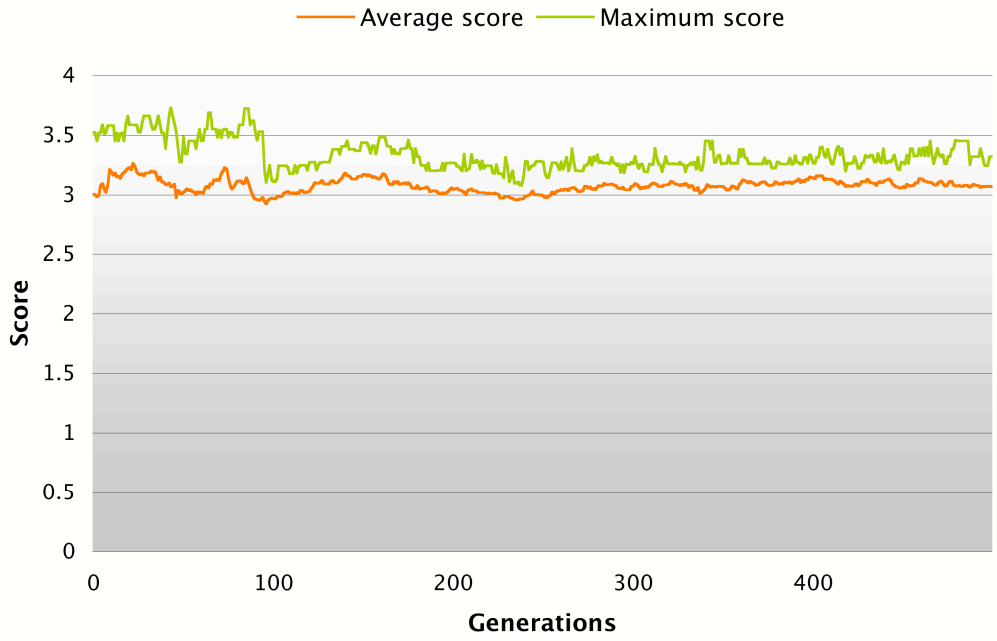
Score: **3.75**
Appeared in generation: **339**

№ of apartments: **20**
№ of circulation blocks: **30**
Volume: **7040 m³**
Total area: **1760 m²**
Apartment area: **1280 m²**
Circulation area: **480 m²**
Height: **64 m**
Footprint: **24 m × 36 m**

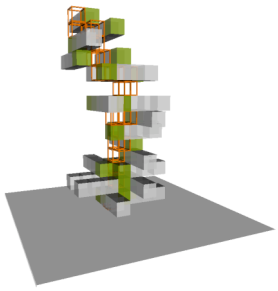
№ of apartments with balconies: **1**
Percentage of apartments with balconies: **5 %**

Apartments with views to +i: **7**
Apartments with views to −i: **12**
Apartments with views to +j: **10**
Apartments with views to −j: **7**
Apartments with no views: **1**

Tower Population: **50** Set mutation rate: **0.01**



Actual mutation rate: **0.0025** Final average score: **3.0657** Time elapsed: **473 s**



Champ №1

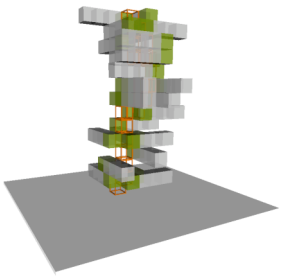
Shape code: **20 6 5 8 21 3 7 4 21 1 2 5 21 4 21 16 20 21 13 1 21 20 12 19 21 14 5 18 17 21 20 21 15 19 19 5 21 16 18 2 21 14 21 16 18 19 21 8**

Score: **3.728571**
Appeared in generation: **45**

№ of apartments: **23**
№ of circulation blocks: **27**
Volume: **7616 m³**
Total area: **1904 m²**
Apartment area: **1472 m²**
Circulation area: **432 m²**
Height: **60 m**
Footprint: **28 m × 32 m**

№ of apartments with balconies: **10**
Percentage of apartments with balconies: **43.47826 %**

Apartments with views to +i: **11**
Apartments with views to −i: **11**
Apartments with views to +j: **12**
Apartments with views to −j: **11**
Apartments with no views: **0**



Champ №2

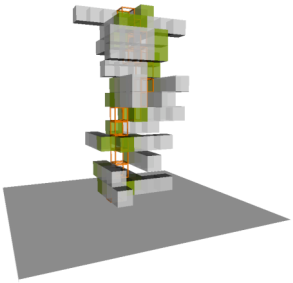
Shape code: **20 2 5 8 21 14 21 4 21 8 2 5 21 1 21 17 1 5 21 6 9 20 12 21 14 18 1 17 4 21 3 21 15 19 20 11 21 11 18 2 21 8 10 5 18 21 16 7**

Score: **3.723809**
Appeared in generation: **88**

№ of apartments: **27**
№ of circulation blocks: **23**
Volume: **8384 m³**
Total area: **2096 m²**
Apartment area: **1728 m²**
Circulation area: **368 m²**
Height: **56 m**
Footprint: **28 m × 28 m**

№ of apartments with balconies: **10**
Percentage of apartments with balconies: **37.037037 %**

Apartments with views to +i: **12**
Apartments with views to −i: **15**
Apartments with views to +j: **12**
Apartments with views to −j: **10**
Apartments with no views: **0**



Champ №3

Shape code: **20 6 5 8 21 14 21 4 21 8 2 5 21 1 21 17 1 5 21 6 9 20 12 21 14 18 1 17 4 21 3 21 15 19 20 11 21 11 18 2 21 8 10 5 18 21 16 7**

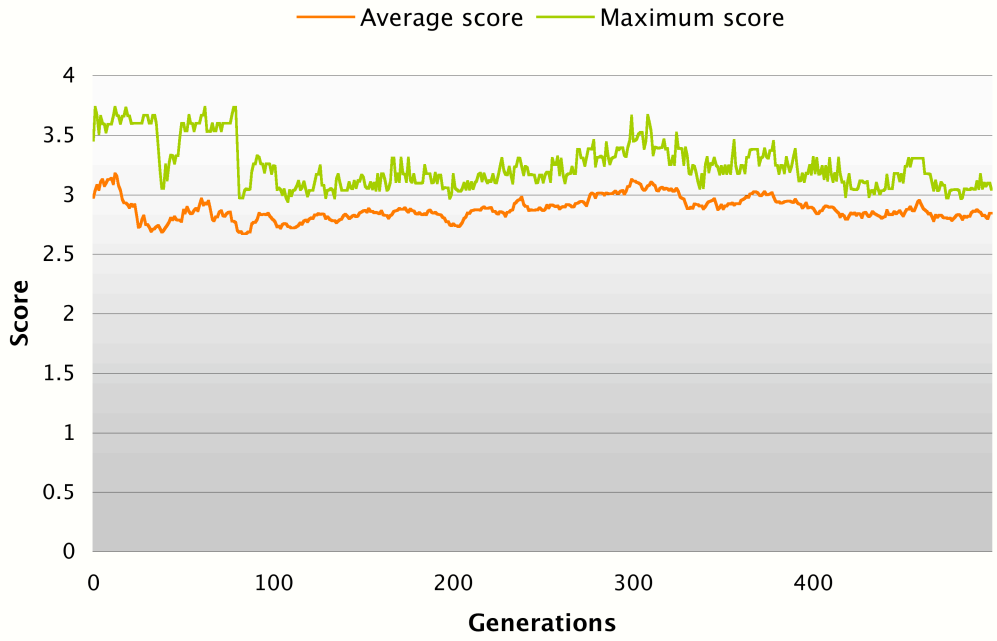
Score: **3.723809**
Appeared in generation: **87**

№ of apartments: **27**
№ of circulation blocks: **23**
Volume: **8384 m³**
Total area: **2096 m²**
Apartment area: **1728 m²**
Circulation area: **368 m²**
Height: **56 m**
Footprint: **28 m × 28 m**

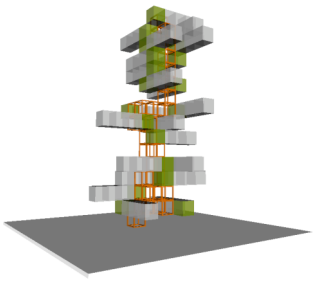
№ of apartments with balconies: **10**
Percentage of apartments with balconies: **37.037037 %**

Apartments with views to +i: **12**
Apartments with views to −i: **14**
Apartments with views to +j: **11**
Apartments with views to −j: **11**
Apartments with no views: **0**

Tower Population: **50** Set mutation rate: **0.05**



Actual mutation rate: **0.0126** Final average score: **2.8472** Time elapsed: **873 s**



Champ Nº1

Shape code: **20 14 17 20 16 21 19 18 7 21 17 15 5 21 4 3 21 18 21 5 21 12 19 3 21 1 20 17 17 4 21 21 14 8 21 2 21 2 8 21 4 14 21 18 2 4 21 13**

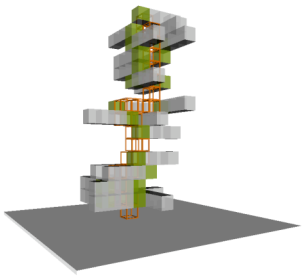
Score: **3.733333**

Appeared in generation: **81**

Nº of apartments: **22**
Nº of circulation blocks: **28**
Volume: **7424 m³**
Total area: **1856 m²**
Apartment area: **1408 m²**
Circulation area: **448 m²**
Height: **64 m**
Footprint: **32 m x 32 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **36.363636 %**

Apartments with views to +i: **11**
Apartments with views to −i: **11**
Apartments with views to +j: **11**
Apartments with views to −j: **11**
Apartments with no views: **0**



Champ Nº2

Shape code: **1 4 19 11 2 21 11 8 2 21 17 15 11 21 17 13 21 18 21 5 21 12 19 3 21 1 20 17 17 4 21 21 14 8 21 2 21 2 8 21 4 14 21 18 2 4 21 13**

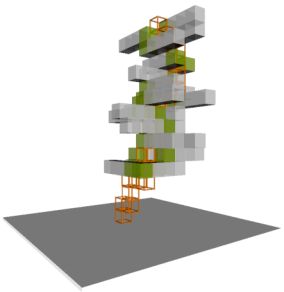
Score: **3.733333**

Appeared in generation: **80**

Nº of apartments: **25**
Nº of circulation blocks: **25**
Volume: **8000 m³**
Total area: **2000 m²**
Apartment area: **1600 m²**
Circulation area: **400 m²**
Height: **64 m**
Footprint: **32 m x 32 m**

Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **36 %**

Apartments with views to +i: **12**
Apartments with views to −i: **12**
Apartments with views to +j: **12**
Apartments with views to −j: **9**
Apartments with no views: **0**



Champ Nº3

Shape code: **20 21 21 20 20 21 13 8 2 21 17 9 5 21 4 6 21 3 18 12 21 12 2 3 21 1 20 6 17 4 21 1 14 17 21 7 21 10 8 21 19 14 21 18 2 4 21 4**

Score: **3.733333**

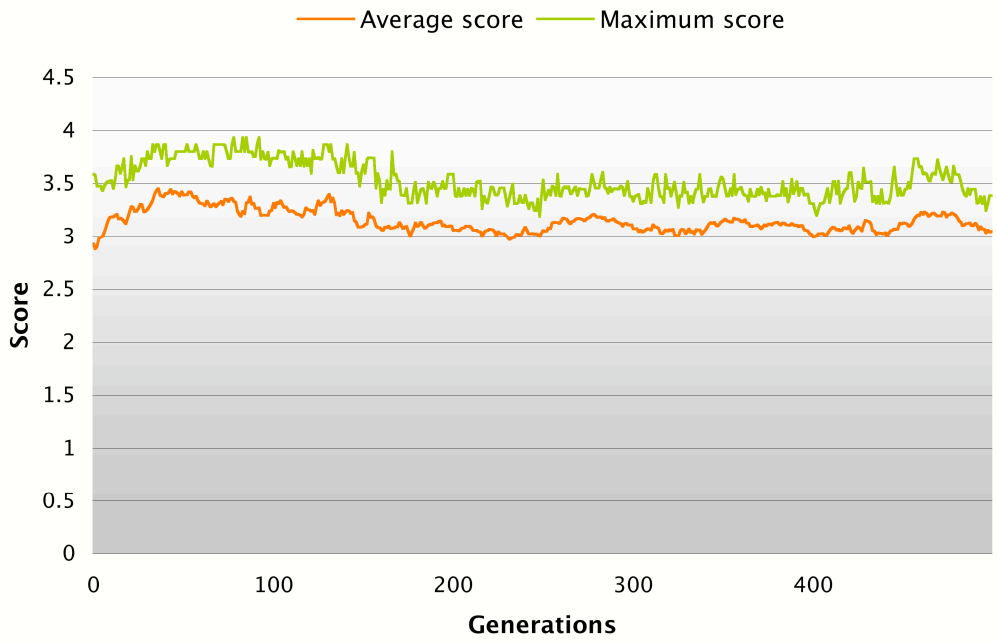
Appeared in generation: **64**

Nº of apartments: **24**
Nº of circulation blocks: **26**
Volume: **7808 m³**
Total area: **1952 m²**
Apartment area: **1536 m²**
Circulation area: **416 m²**
Height: **64 m**
Footprint: **32 m x 32 m**

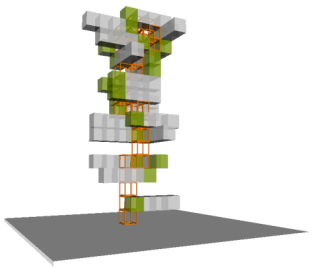
Nº of apartments with balconies: **12**
Percentage of apartments with balconies: **50 %**

Apartments with views to +i: **11**
Apartments with views to −i: **13**
Apartments with views to +j: **7**
Apartments with views to −j: **13**
Apartments with no views: **0**

Tower Population: **50** Set mutation rate: **0.1**



Actual mutation rate: **0.0269** Final average score: **3.0464** Time elapsed: **1267 s**



Champ №1

Shape code: **5 21 21 11 21 17 16 7 21 21 12 7 21 13 7 21 18 19 1 21 13 1 21 13 9 21 17 16 21 14 16 21 19 11 9 21 1 3 17 21 13 17 22 22 16 20 21 12**

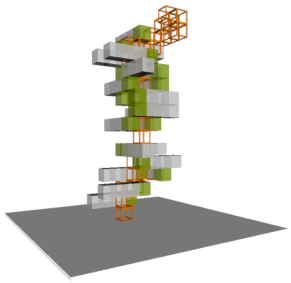
Score: **3.933333**

Appeared in generation: **94**

№ of apartments: **23**
№ of circulation blocks: **27**
Volume: **7616 m³**
Total area: **1904 m²**
Apartment area: **1472 m²**
Circulation area: **432 m²**
Height: **64 m**
Footprint: **28 m × 24 m**

№ of apartments with balconies: **6**
Percentage of apartments with balconies: **26.086956 %**

Apartments with views to +i: **8**
Apartments with views to −i: **13**
Apartments with views to +j: **12**
Apartments with views to −j: **13**
Apartments with no views: **0**



Champ №2

Shape code: **15 21 11 17 21 17 16 7 21 13 19 21 6 1 20 21 21 8 2 21 13 8 21 13 9 21 2 16 21 14 16 21 19 11 15 21 1 17 17 21 2 17 21 20 22 20 21 19**

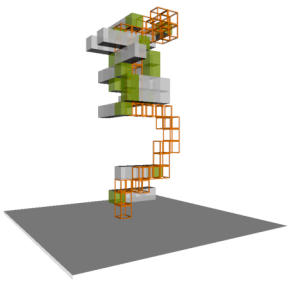
Score: **3.933333**

Appeared in generation: **87**

№ of apartments: **21**
№ of circulation blocks: **29**
Volume: **7232 m³**
Total area: **1808 m²**
Apartment area: **1344 m²**
Circulation area: **464 m²**
Height: **64 m**
Footprint: **28 m × 24 m**

№ of apartments with balconies: **8**
Percentage of apartments with balconies: **38.095238 %**

Apartments with views to +i: **9**
Apartments with views to −i: **13**
Apartments with views to +j: **13**
Apartments with views to −j: **9**
Apartments with no views: **0**



Champ №3

Shape code: **1 21 17 20 21 17 17 7 21 21 17 21 21 18 21 19 19 19 2 21 13 8 21 13 9 21 2 16 21 14 16 21 19 11 15 21 1 17 17 21 2 17 8 20 22 20 21 19**

Score: **3.933333**

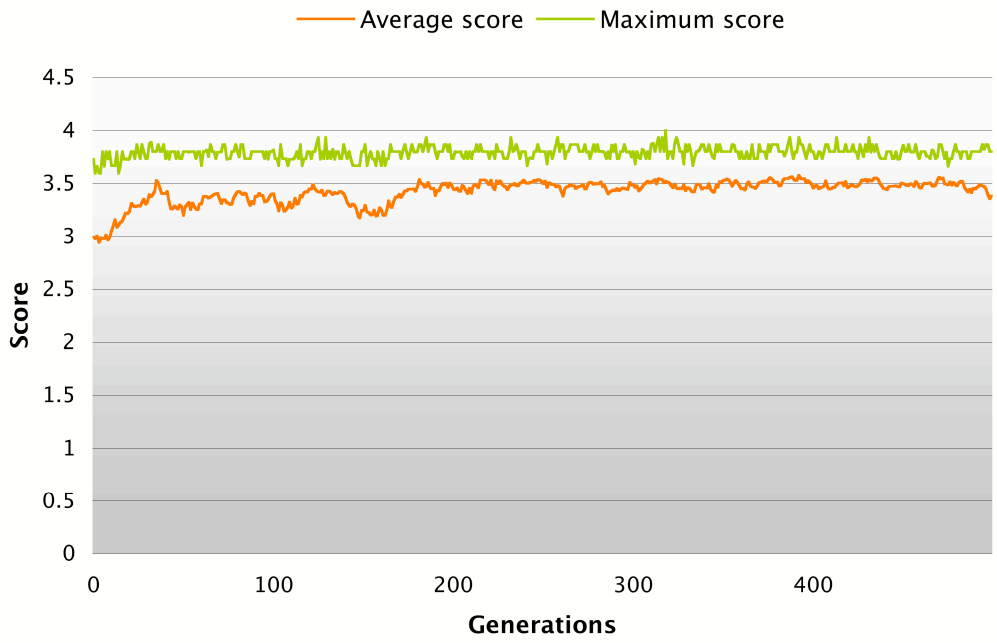
Appeared in generation: **85**

№ of apartments: **16**
№ of circulation blocks: **34**
Volume: **6272 m³**
Total area: **1568 m²**
Apartment area: **1024 m²**
Circulation area: **544 m²**
Height: **64 m**
Footprint: **28 m × 24 m**

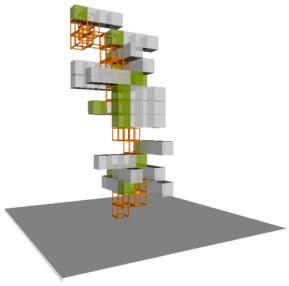
№ of apartments with balconies: **9**
Percentage of apartments with balconies: **56.25 %**

Apartments with views to +i: **7**
Apartments with views to −i: **10**
Apartments with views to +j: **9**
Apartments with views to −j: **7**
Apartments with no views: **0**

Tower Population: **50** Set mutation rate: **0.5**



Actual mutation rate: **0.1439** Final average score: **3.3819** Time elapsed: **2839 s**



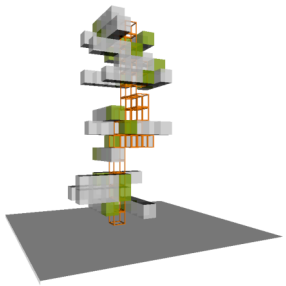
Champ Nº1

Shape code: **20 17 21 12 18 10 21 12 14 21 3 21 4 21 19 20 21 13 21 19 5 9 21 5 21 11 17 8 18 7 21 12 19 15 20 21 18 12 21 5 19 19 18 21 17 18 21 13**

Score: **4**
Appeared in generation: **320**

Nº of apartments: **18**
Nº of circulation blocks: **32**
Volume: **6656 m³**
Total area: **1664 m²**
Apartment area: **1152 m²**
Circulation area: **512 m²**
Height: **64 m**
Footprint: **24 m × 24 m**

Nº of apartments with balconies: **6**
Percentage of apartments with balconies: **33.333336 %**
Apartments with views to +i: 8
Apartments with views to −i: **9**
Apartments with views to +j: **10**
Apartments with views to −j: **11**
Apartments with no views: **0**



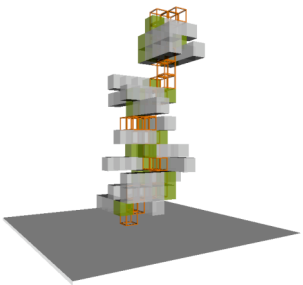
Champ Nº2

Shape code: **10 4 21 7 3 21 3 9 21 21 14 21 1 17 17 20 19 21 19 15 5 21 10 17 20 21 18 18 21 21 7 17 12 3 21 11 19 21 3 9 21 16 21 19 14 22 15 6**

Score: **3.933333**
Appeared in generation: **433**

Nº of apartments: **21**
Nº of circulation blocks: **29**
Volume: **7232 m³**
Total area: **1808 m²**
Apartment area: **1344 m²**
Circulation area: **464 m²**
Height: **64 m**
Footprint: **28 m × 24 m**

Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **42.857143 %**
Apartments with views to +i: **11**
Apartments with views to −i: **9**
Apartments with views to +j: **9**
Apartments with views to −j: **12**
Apartments with no views: **0**



Champ Nº3

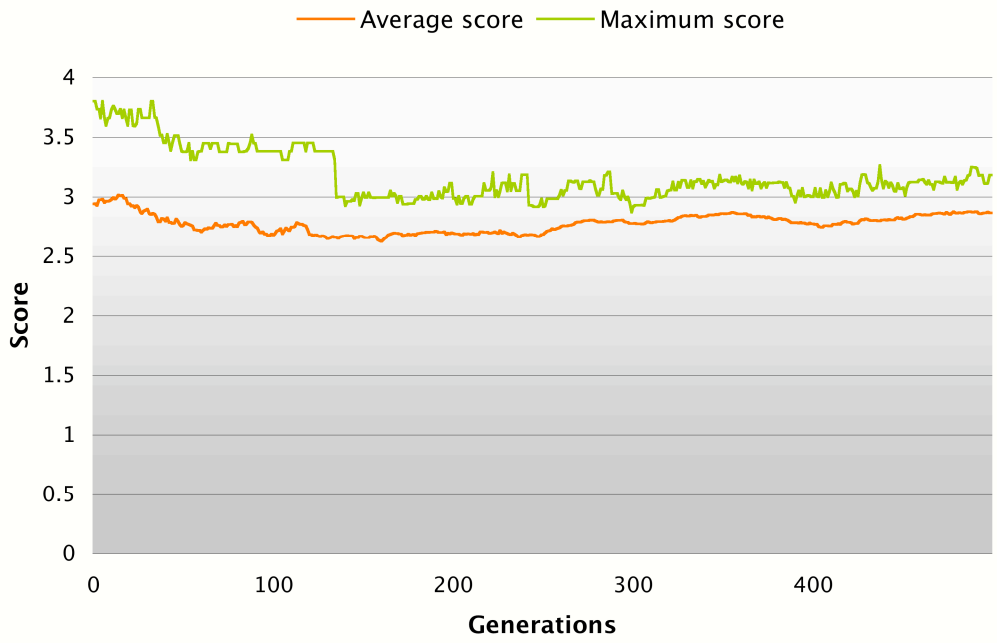
Shape code: **12 20 2 21 17 17 7 3 21 3 9 21 18 7 4 21 9 13 21 3 19 21 12 19 20 17 21 6 21 2 17 3 21 7 20 21 21 14 18 4 21 6 16 21 13 21 17 20**

Score: **3.933333**
Appeared in generation: **415**

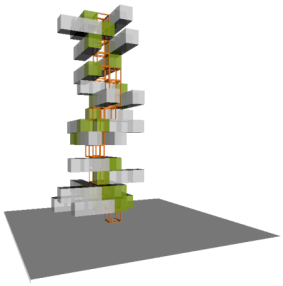
Nº of apartments: **21**
Nº of circulation blocks: **29**
Volume: **7232 m³**
Total area: **1808 m²**
Apartment area: **1344 m²**
Circulation area: **464 m²**
Height: **64 m**
Footprint: **24 m × 28 m**

Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **33.333336 %**
Apartments with views to +i: **12**
Apartments with views to −i: **10**
Apartments with views to +j: **8**
Apartments with views to −j: **13**
Apartments with no views: **0**

Tower Population: **200** Set mutation rate: **0.005**



Actual mutation rate: **0.0009** Final average score: **2.8636** Time elapsed: **4821 s**



Champ №1

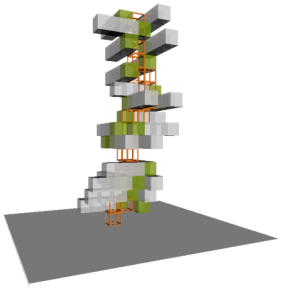
Shape code: **17 16 7 21 12 3 21 20 2 3 12 21 11 8 21 18 21 6 4 9 20 21 5 2 21 1 17 21 4 6 20 21 15 21 19 10 21 8 21 10 4 21 3 18 21 10 4 20**

Score: **3.8**
Appeared in generation: **35**

№ of apartments: **25**
№ of circulation blocks: **25**
Volume: **8000 m³**
Total area: **2000 m²**
Apartment area: **1600 m²**
Circulation area: **400 m²**
Height: **64 m**
Footprint: **28 m × 32 m**

№ of apartments with balconies: **11**
Percentage of apartments with balconies: **44 %**

Apartments with views to +i: **13**
Apartments with views to −i: **11**
Apartments with views to +j: **14**
Apartments with views to −j: **8**
Apartments with no views: **0**



Champ №2

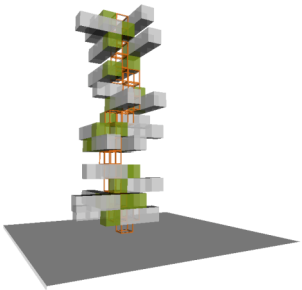
Shape code: **17 16 7 21 12 3 21 20 2 3 12 21 11 8 21 18 21 6 4 9 20 21 5 2 21 1 17 21 4 6 20 21 15 21 19 10 21 8 21 10 4 21 3 18 21 10 4 20**

Score: **3.8**
Appeared in generation: **35**

№ of apartments: **25**
№ of circulation blocks: **25**
Volume: **8000 m³**
Total area: **1760 m²**
Apartment area: **2000 m²**
Circulation area: **400 m²**
Height: **64 m**
Footprint: **28 m × 32 m**

№ of apartments with balconies: **13**
Percentage of apartments with balconies: **52 %**

Apartments with views to +i: **13**
Apartments with views to −i: **12**
Apartments with views to +j: **12**
Apartments with views to −j: **10**
Apartments with no views: **0**



Champ №3

Shape code: **7 20 19 13 21 2 16 3 18 21 5 21 11 8 21 18 21 6 4 9 20 21 5 2 21 1 17 21 4 6 20 21 15 21 19 10 21 8 21 10 4 21 3 18 21 10 4 20**

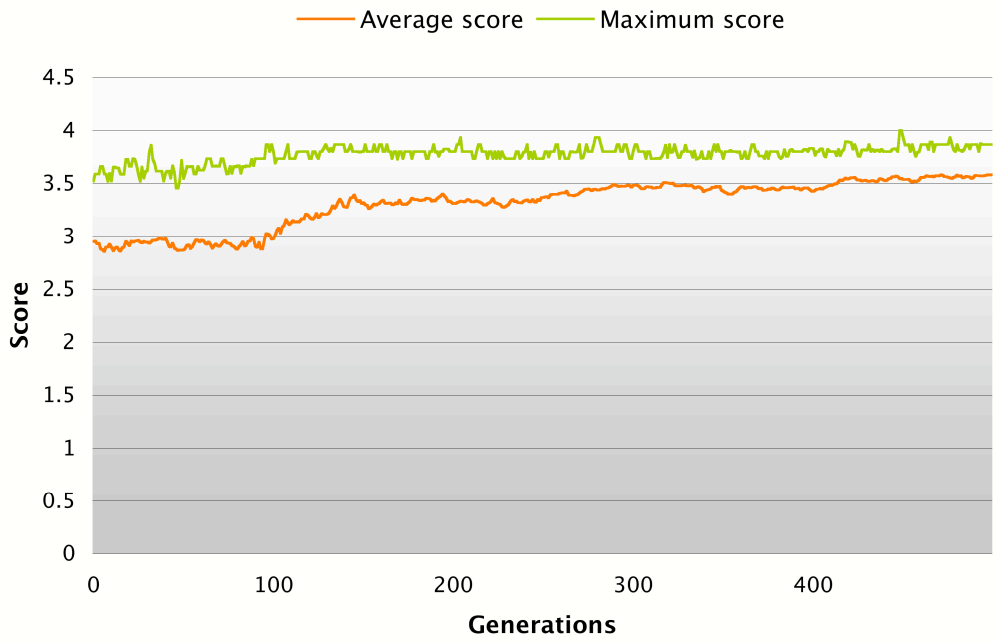
Score: **3.8**
Appeared in generation: **34**

№ of apartments: **24**
№ of circulation blocks: **26**
Volume: **7808 m³**
Total area: **1952 m²**
Apartment area: **1536 m²**
Circulation area: **416 m²**
Height: **64 m**
Footprint: **28 m × 32 m**

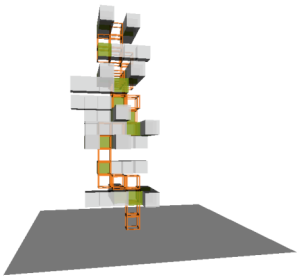
№ of apartments with balconies: **12**
Percentage of apartments with balconies: **50 %**

Apartments with views to +i: **12**
Apartments with views to −i: **12**
Apartments with views to +j: **13**
Apartments with views to −j: **8**
Apartments with no views: **0**

Tower Population: **200** Set mutation rate: **0.01**



Actual mutation rate: **0.0025** Final average score: **3.5829** Time elapsed: **4434 s**

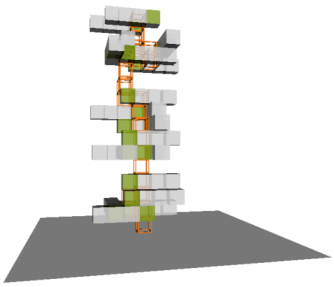


Champ №1

Shape code: **21 2 1 11 21 18 17 18 21 4 21 21 15 4 3 21 20 10 20 3 16 21 8 7 21 19 18 8 21 2 17 21 3 17 7 17 20 21 21 18 19 14 18 21 20 21 7 11**

Score: **4** Appeared in generation: **450**

№ of apartments: **19** № of apartments with balconies: **3**
№ of circulation blocks: **31** Percentage of apartments with balconies: **15.789473 %**
Volume: **6848 m³**
Total area: **1712 m²** Apartments with views to +i: **10**
Apartment area: **1216 m²** Apartments with views to −i: **4**
Circulation area: **496 m²** Apartments with views to +j: **8**
Height: **64 m** Apartments with views to −j: **8**
Footprint: **24 m × 24 m** Apartments with no views: **0**

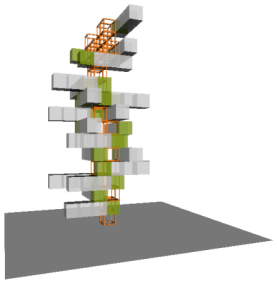


Champ №2

Shape code: **19 2 1 11 21 5 17 18 21 13 21 21 2 12 3 21 13 21 18 9 4 21 12 7 21 13 18 19 21 20 21 21 6 17 13 17 18 9 21 4 19 14 18 21 20 21 7 11**

Score: **3.933333** Appeared in generation: **478**

№ of apartments: **21** № of apartments with balconies: **5**
№ of circulation blocks: **29** Percentage of apartments with balconies: **23.809525 %**
Volume: **7232 m³**
Total area: **1808 m²** Apartments with views to +i: **7**
Apartment area: **1344 m²** Apartments with views to −i: **10**
Circulation area: **464 m²** Apartments with views to +j: **9**
Height: **64 m** Apartments with views to −j: **13**
Footprint: **28 m × 24 m** Apartments with no views: **0**



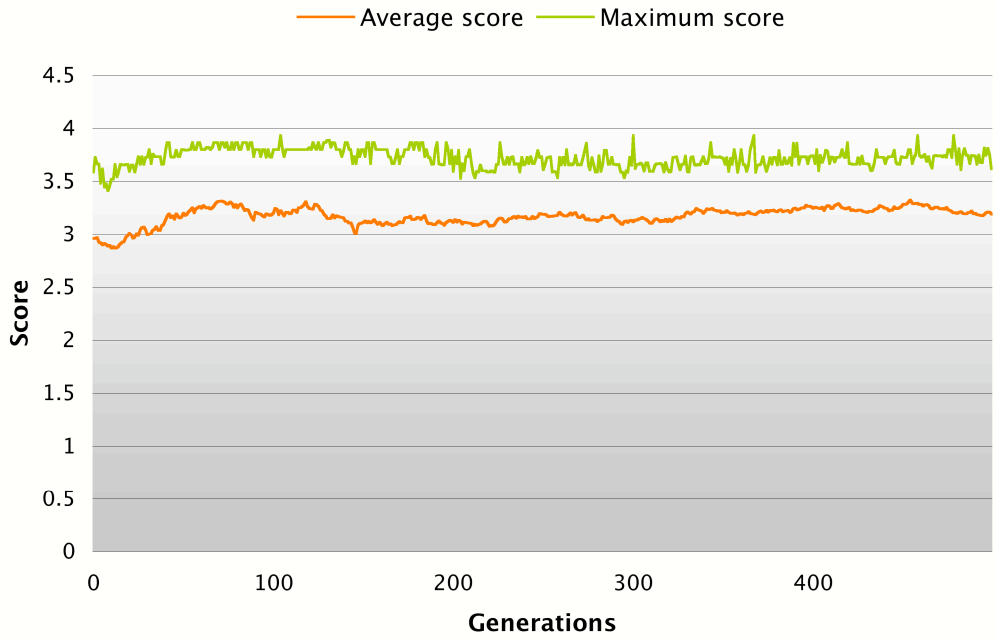
Champ №3

Shape code: **3 21 20 2 21 3 19 7 21 5 17 21 2 12 21 3 15 21 8 19 14 21 12 7 21 14 5 8 21 2 3 21 18 21 13 21 18 9 21 17 21 20 4 22 22 12 20 19**

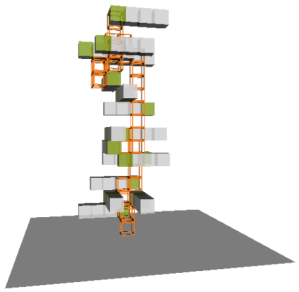
Score: **3.933333** Appeared in generation: **283**

№ of apartments: **22** № of apartments with balconies: **10**
№ of circulation blocks: **28** Percentage of apartments with balconies: **45.454548 %**
Volume: **7424 m³**
Total area: **1856 m²** Apartments with views to +i: **12**
Apartment area: **1408 m²** Apartments with views to −i: **8**
Circulation area: **448 m²** Apartments with views to +j: **12**
Height: **64 m** Apartments with views to −j: **10**
Footprint: **28 m × 24 m** Apartments with no views: **0**

Tower Population: **200** Set mutation rate: **0.05**



Actual mutation rate: **0.0013** Final average score: **3.1935** Time elapsed: **4962 s**



Champ Nº1

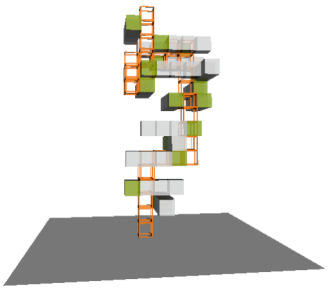
Shape code: **16 7 20 21 19 21 19 6 21 18 6 9 21 5 9 6 17 21 17 21 12 10 20 21 7 16 19 19 21 11 17 21 21 20 21 20 16 21 10 21 2 20 19 5 2 21 3 2**

Score: **3.933333**
Appeared in generation: **480**

Nº of apartments: **16**
Nº of circulation blocks: **34**
Volume: **6272 m³**
Total area: **1568 m²**
Apartment area: **1024 m²**
Circulation area: **544 m²**
Height: **64 m**
Footprint: **28 m x 24 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **31.25 %**

Apartments with views to +i: **8**
Apartments with views to −i: **10**
Apartments with views to +j: **7**
Apartments with views to −j: **11**
Apartments with no views: **0**



Champ Nº2

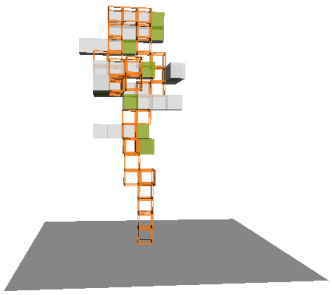
Shape code: **21 4 21 13 21 20 21 17 17 3 17 21 18 14 21 20 3 21 21 2 18 16 21 8 21 4 21 21 7 19 13 22 19 1 19 19 22 22 16 20 21 17 5 21 21 21 10 21**

Score: **3.933333**
Appeared in generation: **460**

Nº of apartments: **15**
Nº of circulation blocks: **35**
Volume: **6080m³**
Total area: **1520 m²**
Apartment area: **960 m²**
Circulation area: **560 m²**
Height: **64 m**
Footprint: **24 m x 28 m**

Nº of apartments with balconies: **4**
Percentage of apartments with balconies: **26.666668 %**

Apartments with views to +i: **10**
Apartments with views to −i: **6**
Apartments with views to +j: **10**
Apartments with views to −j: **6**
Apartments with no views: **0**



Champ Nº3

Shape code: **21 21 21 20 19 21 21 18 8 18 21 18 15 21 17 21 13 21 17 21 12 21 19 21 21 15 21 15 19 19 22 22 15 22 19 22 17 22 6 9 20 20 20 21 17 15 20 22**

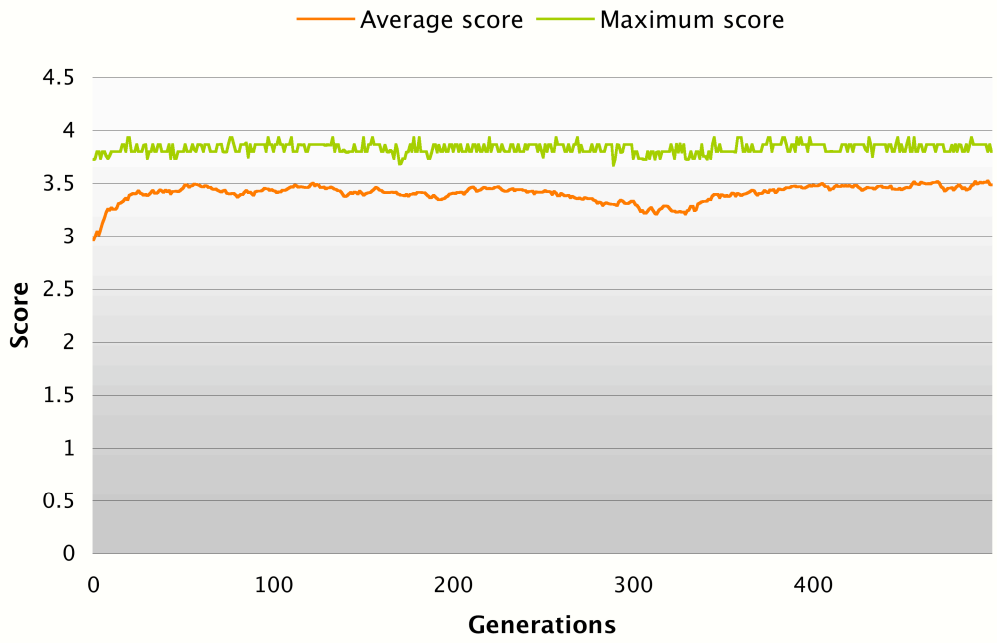
Score: **3.933333**
Appeared in generation: **369**

Nº of apartments: **10**
Nº of circulation blocks: **40**
Volume: **5120 m³**
Total area: **1280 m²**
Apartment area: **640 m²**
Circulation area: **640 m²**
Height: **64 m**
Footprint: **28 m x 24 m**

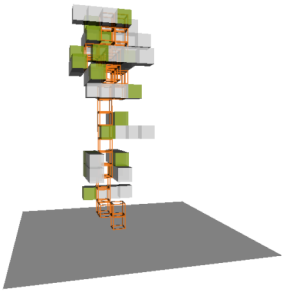
Nº of apartments with balconies: **2**
Percentage of apartments with balconies: **20 %**

Apartments with views to +i: **6**
Apartments with views to −i: **4**
Apartments with views to +j: **7**
Apartments with views to −j: **8**
Apartments with no views: **0**

Tower Population: **200** Set mutation rate: **0.1**



Actual mutation rate: **0.0291** Final average score: **3.4873** Time elapsed: **5455 s**

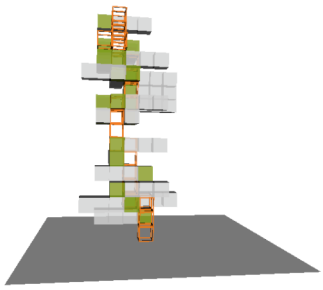


Champ Nº1

Shape code: **19 18 21 9 21 21 6 20 12 21 16 14 21 21 5 21 16 21 21 17 11 21 20 19 20 10 20 21 13 18 16 18 19 18 10 21 9 17 17 20 21 21 3 19 9 22 2 1**

Score: **3.933333** Appeared in generation: **490**

Nº of apartments: **17** Nº of apartments with balconies: **7**
Nº of circulation blocks: **33** Percentage of apartments with balconies: **41.176472 %**
Volume: **6464 m³**
Total area: **1616 m²** Apartments with views to +i: **6**
Apartment area: **1088 m²** Apartments with views to −i: **11**
Circulation area: **528 m²** Apartments with views to +j: **11**
Height: **64 m** Apartments with views to −j: **8**
Footprint: **24 m × 28 m** Apartments with no views: **0**

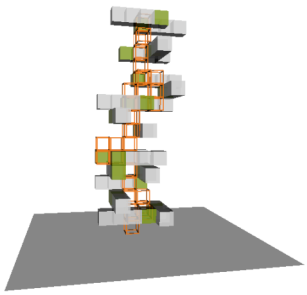


Champ Nº2

Shape code: **8 21 17 18 2 4 21 14 17 21 10 21 9 21 18 6 21 21 17 15 11 21 9 19 20 14 4 21 4 21 1 2 4 21 14 18 9 21 5 1 21 21 12 19 22 22 19 18**

Score: **3.933333** Appeared in generation: **458**

Nº of apartments: **21** Nº of apartments with balconies: **10**
Nº of circulation blocks: **29** Percentage of apartments with balconies: **47.619049 %**
Volume: **7232 m³**
Total area: **1808 m²** Apartments with views to +i: **10**
Apartment area: **1344 m²** Apartments with views to −i: **12**
Circulation area: **464 m²** Apartments with views to +j: **6**
Height: **64 m** Apartments with views to −j: **12**
Footprint: **24 m × 28 m** Apartments with no views: **0**



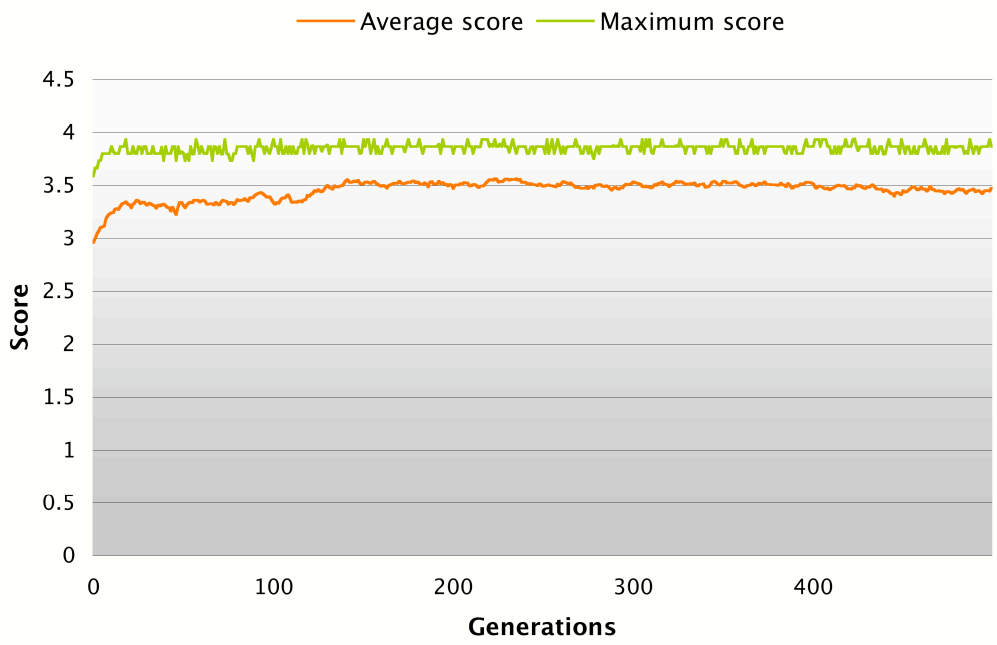
Champ Nº3

Shape code: **17 3 18 4 21 19 6 21 15 21 4 10 21 19 5 19 1 21 17 17 21 12 20 21 14 18 21 11 21 17 22 9 17 21 12 21 19 18 1 21 21 9 4 21 20 21 1 11**

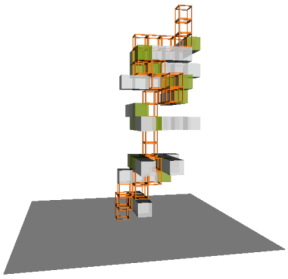
Score: **3.933333** Appeared in generation: **455**

Nº of apartments: **18** Nº of apartments with balconies: **3**
Nº of circulation blocks: **32** Percentage of apartments with balconies: **16.666668 %**
Volume: **6656 m³**
Total area: **1664 m²** Apartments with views to +i: **8**
Apartment area: **1152 m²** Apartments with views to −i: **11**
Circulation area: **512 m²** Apartments with views to +j: **7**
Height: **64 m** Apartments with views to −j: **12**
Footprint: **28 m × 24 m** Apartments with no views: **0**

Tower Population: **200** Set mutation rate: **0.5**



Actual mutation rate: **0.1452** Final average score: **3.4731** Time elapsed: **9134 s**



Champ Nº1

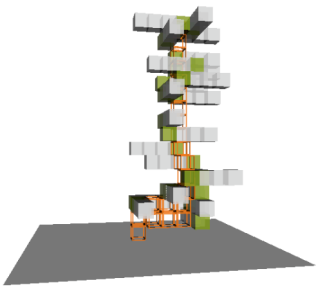
Shape code: **4 20 21 17 20 21 17 15 21 18 4 6 9 21 21 21 6 1 5 21 10 21 17 16 20 20 20 21 18 8 7 19 21 1 19 18 1 21 1 5 21 17 17 17 8 20 21 21**

Score: **3.941176** Appeared in generation: **425**

Nº of apartments: **17**
Nº of circulation blocks: **33**
Volume: **6464 m³**
Total area: **1616 m²**
Apartment area: **1088 m²**
Circulation area: **528 m²**
Height: **64 m**
Footprint: **24 m x 24 m**

Nº of apartments with balconies: **3**
Percentage of apartments with balconies: **17.647058 %**

Apartments with views to +i: **7**
Apartments with views to −i: **8**
Apartments with views to +j: **7**
Apartments with views to −j: **8**
Apartments with no views: **1**



Champ Nº2

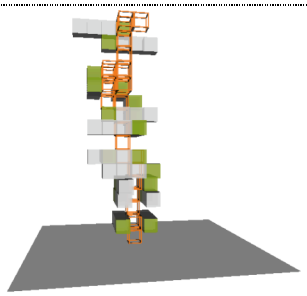
Shape code: **17 21 6 9 17 20 19 22 17 17 16 21 12 21 20 8 14 21 5 8 21 18 15 21 18 7 21 13 21 6 21 21 1 6 21 9 5 21 1 11 21 1 21 6 21 1 4 3**

Score: **3.933333** Appeared in generation: **500**

Nº of apartments: **23**
Nº of circulation blocks: **27**
Volume: **7616 m³**
Total area: **1904 m²**
Apartment area: **1472 m²**
Circulation area: **432 m²**
Height: **64 m**
Footprint: **28 m x 24 m**

Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **39.130436 %**

Apartments with views to +i: **8**
Apartments with views to −i: **15**
Apartments with views to +j: **9**
Apartments with views to −j: **14**
Apartments with no views: **0**



Champ Nº3

Shape code: **10 16 18 21 21 4 6 21 20 6 16 21 20 11 21 19 11 2 21 21 11 8 18 21 13 8 21 8 19 21 8 10 21 2 20 20 22 17 18 21 20 21 21 9 21 7 20 17**

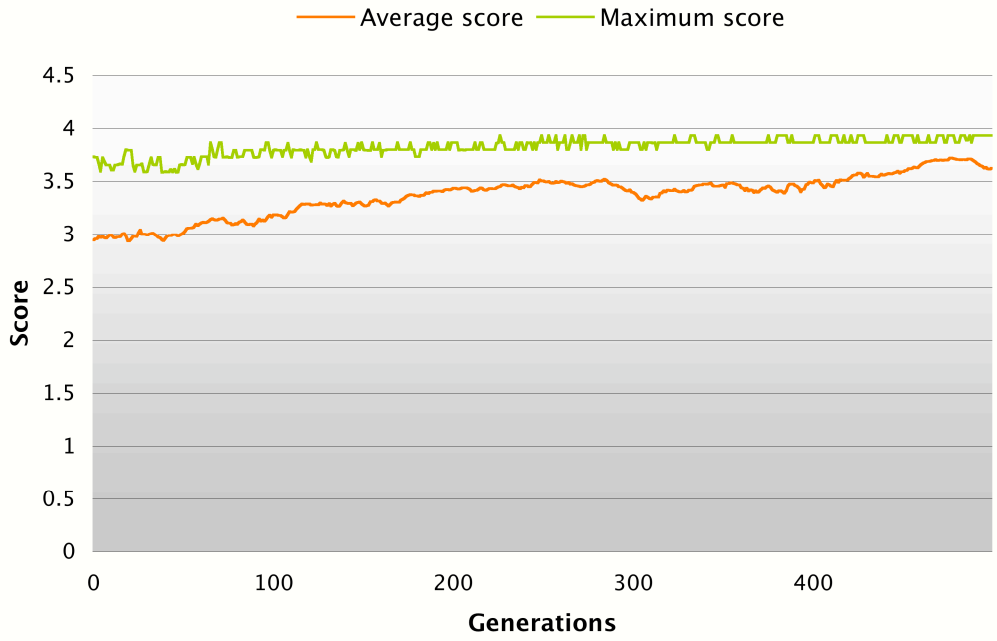
Score: **3.933333** Appeared in generation: **489**

Nº of apartments: **19**
Nº of circulation blocks: **31**
Volume: **6848 m³**
Total area: **1712 m²**
Apartment area: **1216 m²**
Circulation area: **496 m²**
Height: **64 m**
Footprint: **24 m x 28 m**

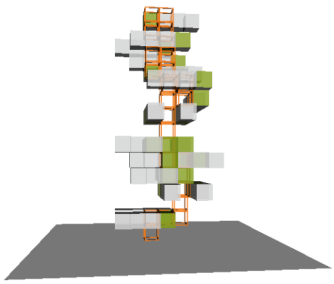
Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **47.368423 %**

Apartments with views to +i: **11**
Apartments with views to −i: **8**
Apartments with views to +j: **11**
Apartments with views to −j: **7**
Apartments with no views: **0**

Tower Population: **500** Set mutation rate: **0.005**



Actual mutation rate: **0.0012** Final average score: **3.6265** Time elapsed: **10928 s**



Champ №1

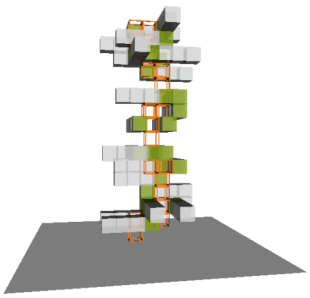
Shape code: **11 15 17 17 21 21 4 6 20 21 3 2 21 7 13 8 19 21 3 16 2 21 21 12 14 21 17 20 8 21 11 19 18 21 9 19 21 17 18 15 21 11 7 21 21 9 19 22**

Score: **3.933333**
Appeared in generation: **500**

№ of apartments: **21**
№ of circulation blocks: **29**
Volume: **7232 m³**
Total area: **1808 m²**
Apartment area: **1344 m²**
Circulation area: **464 m²**
Height: **64 m**
Footprint: **28 m x 24 m**

№ of apartments with balconies: **6**
Percentage of apartments with balconies: **28.57143 %**

Apartments with views to +i: **12**
Apartments with views to −i: **7**
Apartments with views to +j: **9**
Apartments with views to −j: **12**
Apartments with no views: **0**



Champ №2

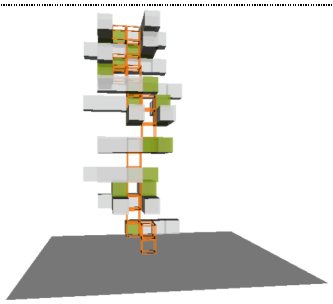
Shape code: **11 15 17 17 21 9 4 6 21 9 21 7 21 7 20 8 19 21 3 8 2 21 21 16 10 21 18 17 16 21 16 3 21 21 9 19 21 5 6 15 21 11 7 21 4 9 19 22**

Score: **3.933333**
Appeared in generation: **500**

№ of apartments: **25**
№ of circulation blocks: **25**
Volume: **8000 m³**
Total area: **2000 m²**
Apartment area: **1600 m²**
Circulation area: **400 m²**
Height: **64 m**
Footprint: **28 m x 24 m**

№ of apartments with balconies: **10**
Percentage of apartments with balconies: **40 %**

Apartments with views to +i: **14**
Apartments with views to −i: **8**
Apartments with views to +j: **12**
Apartments with views to −j: **10**
Apartments with no views: **0**



Champ №3

Shape code: **18 9 19 21 21 15 4 6 20 21 2 8 21 7 11 21 21 17 3 16 7 21 21 12 14 21 8 19 7 21 11 1 21 19 2 18 21 14 17 9 21 12 7 21 4 9 19 14**

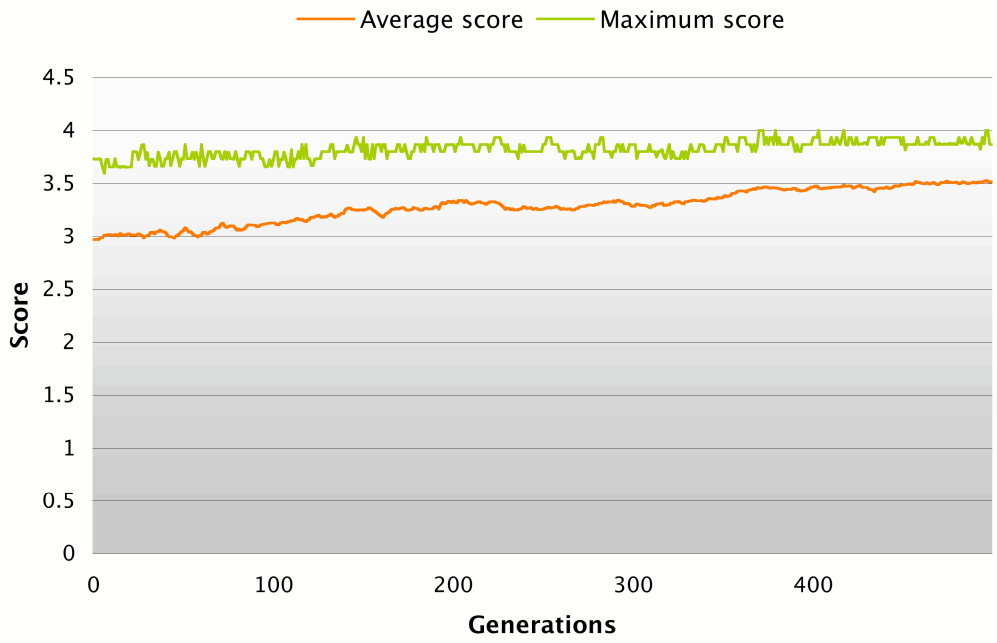
Score: **3.933333**
Appeared in generation: **500**

№ of apartments: **25**
№ of circulation blocks: **25**
Volume: **8000 m³**
Total area: **2000 m²**
Apartment area: **1600 m²**
Circulation area: **400 m²**
Height: **64 m**
Footprint: **28 m x 24 m**

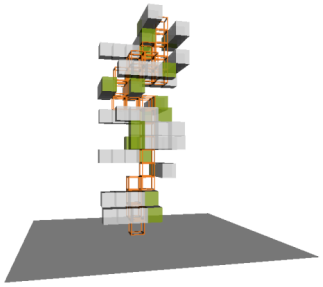
№ of apartments with balconies: **11**
Percentage of apartments with balconies: **44 %**

Apartments with views to +i: **11**
Apartments with views to −i: **8**
Apartments with views to +j: **8**
Apartments with views to −j: **15**
Apartments with no views: **0**

Tower Population: **500** Set mutation rate: **0.01**



Actual mutation rate: **0.0025** Final average score: **3.5156** Time elapsed: **10787 s**



Champ Nº1

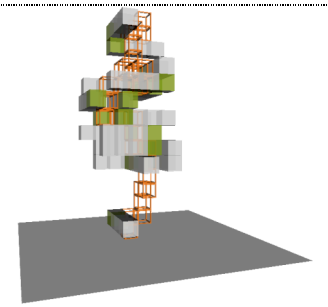
Shape code: **11 18 21 11 21 17 21 12 21 7 21 13 9 21 13 21 7 11 21 8 18 19 19 21 20 16 10 21 17 1 21 20 17 22 11 18 17 21 12 19 21 7 17 21 2 12 21 14**

Score: **4**
Appeared in generation: **498**

Nº of apartments: **19**
Nº of circulation blocks: **31**
Volume: **6848 m³**
Total area: **1712 m²**
Apartment area: **1216 m²**
Circulation area: **496 m²**
Height: **64 m**
Footprint: **24 m x 24 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **42.105263 %**

Apartments with views to +i: **13**
Apartments with views to −i: **9**
Apartments with views to +j: **10**
Apartments with views to −j: **13**
Apartments with no views: **0**



Champ Nº2

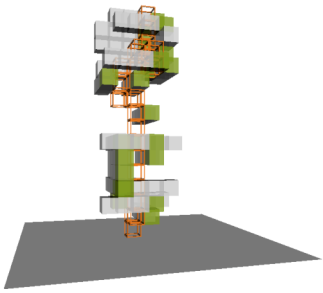
Shape code: **6 17 21 21 20 21 21 6 12 7 21 12 7 3 19 21 7 11 21 13 19 19 1 21 20 16 10 21 17 15 21 20 17 22 11 18 8 21 12 18 21 21 9 21 14 18 17 21**

Score: **4**
Appeared in generation: **419**

Nº of apartments: **19**
Nº of circulation blocks: **31**
Volume: **6848 m³**
Total area: **1712 m²**
Apartment area: **1216 m²**
Circulation area: **496 m²**
Height: **64 m**
Footprint: **24 m x 24 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **26.31579 %**

Apartments with views to +i: **8**
Apartments with views to −i: **7**
Apartments with views to +j: **7**
Apartments with views to −j: **12**
Apartments with no views: **0**



Champ Nº3

Shape code: **16 20 21 11 9 21 13 2 21 10 21 10 16 21 11 1 2 21 21 8 21 18 19 21 20 16 10 21 17 15 21 20 17 22 3 18 16 21 8 18 21 7 12 21 11 7 21 15**

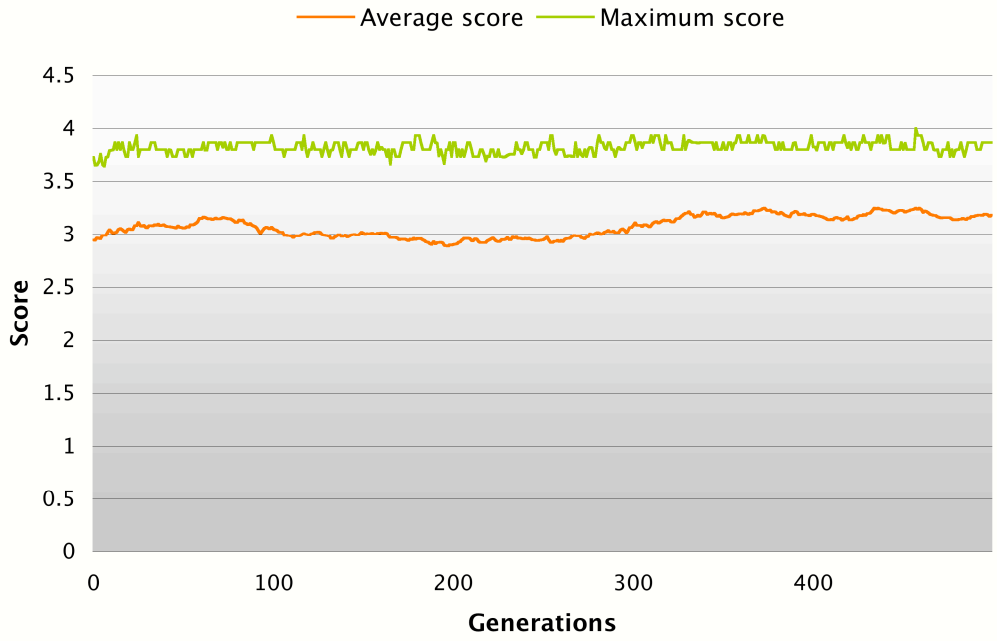
Score: **4**
Appeared in generation: **339**

Nº of apartments: **23**
Nº of circulation blocks: **27**
Volume: **7616 m³**
Total area: **1904 m²**
Apartment area: **1472 m²**
Circulation area: **432 m²**
Height: **64 m**
Footprint: **24 m x 24 m**

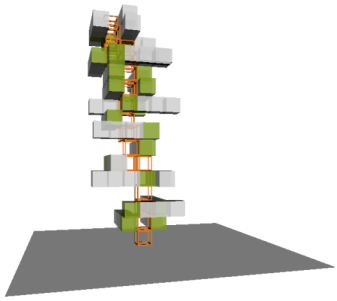
Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **34.782608 %**

Apartments with views to +i: **14**
Apartments with views to −i: **7**
Apartments with views to +j: **15**
Apartments with views to −j: **8**
Apartments with no views: **0**

Tower Population: **500** Set mutation rate: **0.05**



Actual mutation rate: **0.0123** Final average score: **3.1846** Time elapsed: **12120 s**



Champ Nº1

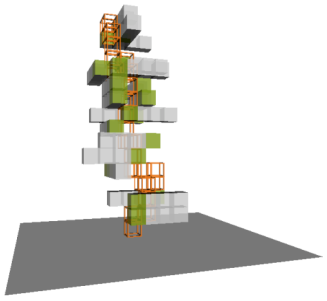
Shape code: **8 2 21 13 8 21 21 3 12 21 19 14 20 21 8 21 11 18 18 21 10 21 15 5 21 20 16 10 21 10 8 21 19 5 18 10 17 21 1 19 21 14 12 18 1 21 18 4**

Score: **4**
Appeared in generation: **459**

Nº of apartments: **23**
Nº of circulation blocks: **27**
Volume: **7616 m³**
Total area: **1904 m²**
Apartment area: **1472 m²**
Circulation area: **432 m²**
Height: **64 m**
Footprint: **24 m × 24 m**

Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **30.434782 %**

Apartments with views to +i: **13**
Apartments with views to −i: **13**
Apartments with views to +j: **14**
Apartments with views to −j: **9**
Apartments with no views: **0**



Champ Nº2

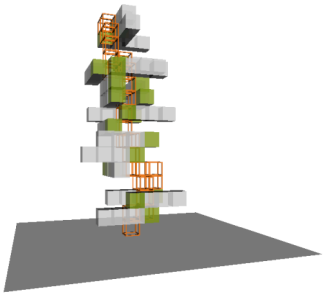
Shape code: **5 17 21 13 15 21 18 17 21 19 19 14 20 21 3 21 11 18 18 21 10 21 15 5 21 20 16 10 21 10 8 21 19 5 18 10 17 21 1 19 21 21 12 18 1 21 18 4**

Score: **3.933333**
Appeared in generation: **462**

Nº of apartments: **19**
Nº of circulation blocks: **31**
Volume: **6848 m³**
Total area: **1712 m²**
Apartment area: **1216 m²**
Circulation area: **496 m²**
Height: **64 m**
Footprint: **28 m × 24 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **26.31579 %**

Apartments with views to +i: **11**
Apartments with views to −i: **11**
Apartments with views to +j: **12**
Apartments with views to −j: **7**
Apartments with no views: **0**



Champ Nº3

Shape code: **11 17 21 13 15 21 18 17 21 19 19 14 20 21 3 21 11 18 18 21 10 21 15 5 21 20 16 10 21 10 8 21 19 5 18 10 17 21 1 19 21 21 12 18 1 21 18 4**

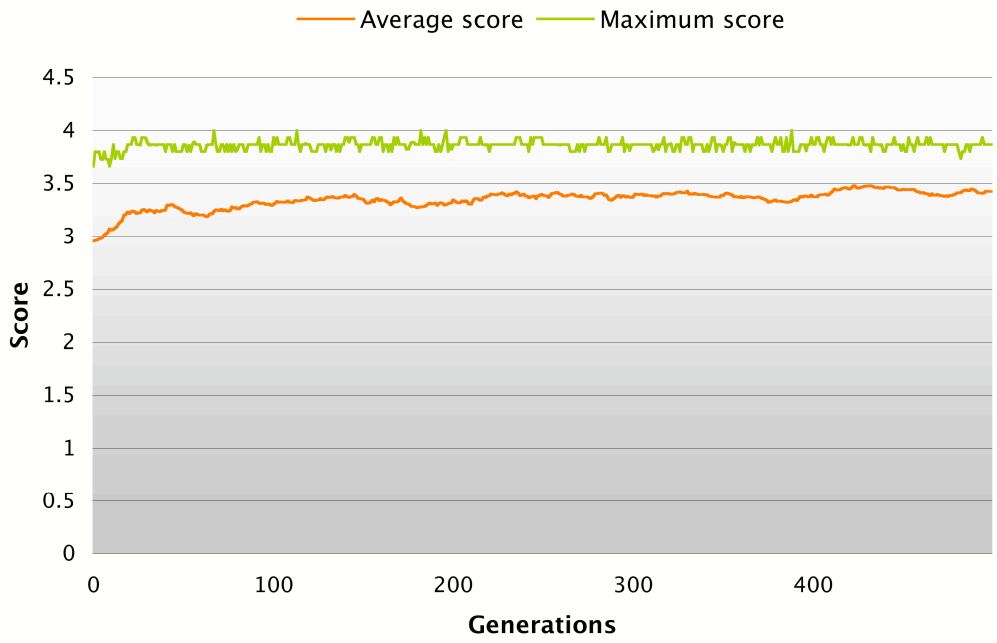
Score: **3.933333**
Appeared in generation: **461**

Nº of apartments: **19**
Nº of circulation blocks: **31**
Volume: **6848 m³**
Total area: **1712 m²**
Apartment area: **1216 m²**
Circulation area: **496 m²**
Height: **64 m**
Footprint: **28 m × 24 m**

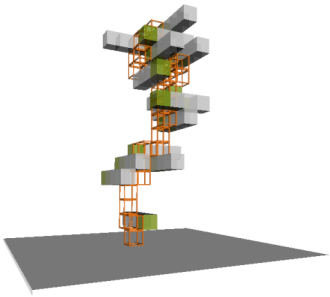
Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **26.31579 %**

Apartments with views to +i: **12**
Apartments with views to −i: **10**
Apartments with views to +j: **12**
Apartments with views to −j: **7**
Apartments with no views: **0**

Tower Population: **500** Set mutation rate: **0.1**



Actual mutation rate: **0.0287** Final average score: **3.4256** Time elapsed: **12113 s**



Champ Nº1

Shape code: **18 16 21 21 21 14 17 21 4 14 21 1 20 17 21 21 12 20 21 13 8 18 21 8 20 21 10 17 21 18 18 21 4 6 18 22 19 19 21 20 21 15 17 13 21 4 18 18**

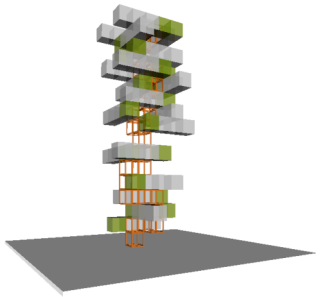
Score: **4**

Appeared in generation: **390**

Nº of apartments: **15**
Nº of circulation blocks: **35**
Volume: **6080 m³**
Total area: **1760 m²**
Apartment area: **1520 m²**
Circulation area: **560 m²**
Height: **64 m**
Footprint: **24 m x 24 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **33.333336 %**

Apartments with views to +i: **11**
Apartments with views to −i: **7**
Apartments with views to +j: **5**
Apartments with views to −j: **10**
Apartments with no views: **0**



Champ Nº2

Shape code: **9 17 20 21 14 8 21 15 19 19 21 5 21 17 21 2 11 21 21 5 18 21 10 17 12 21 11 21 20 2 12 21 7 3 8 19 21 9 13 21 14 21 17 3 16 18 19 15**

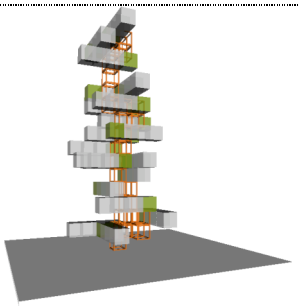
Score: **4**

Appeared in generation: **198**

Nº of apartments: **22**
Nº of circulation blocks: **28**
Volume: **7424m³**
Total area: **1856 m²**
Apartment area: **1408 m²**
Circulation area: **448 m²**
Height: **64 m**
Footprint: **24 m x 24 m**

Nº of apartments with balconies: **10**
Percentage of apartments with balconies: **45.454548 %**

Apartments with views to +i: **10**
Apartments with views to −i: **10**
Apartments with views to +j: **14**
Apartments with views to −j:11
Apartments with no views: **0**



Champ Nº3

Shape code: **6 7 20 21 17 4 22 20 21 19 21 11 21 18 18 14 21 12 21 3 4 15 21 7 20 21 15 13 21 17 15 21 7 3 19 19 21 9 4 21 21 11 21 14 17 21 6 18**

Score: **4**

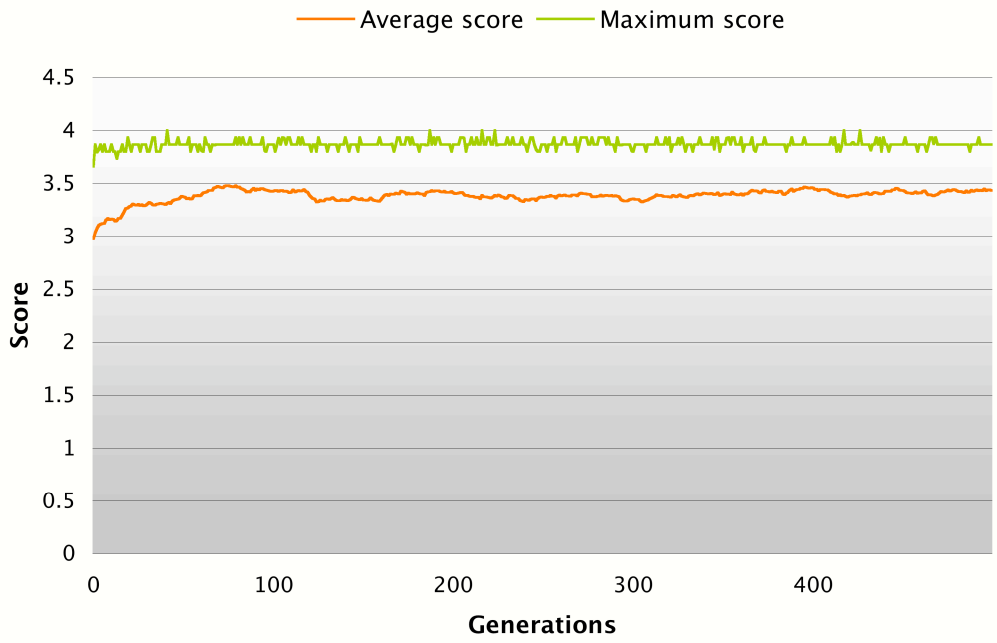
Appeared in generation: **184**

Nº of apartments: **20**
Nº of circulation blocks: **30**
Volume: **7040 m³**
Total area: **1760 m²**
Apartment area: **1280 m²**
Circulation area: **480 m²**
Height: **64 m**
Footprint: **24 m x 24 m**

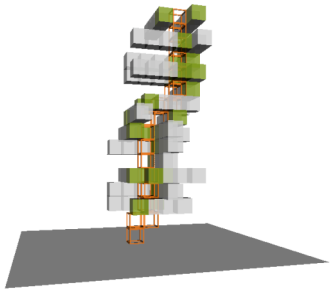
Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **25 %**

Apartments with views to +i: **15**
Apartments with views to −i: **7**
Apartments with views to +j: **7**
Apartments with views to −j: **14**
Apartments with no views: **0**

Tower Population: **500** Set mutation rate: **0.5**



Actual mutation rate: **0.1432** Final average score: **3.4344** Time elapsed: **14043 s**



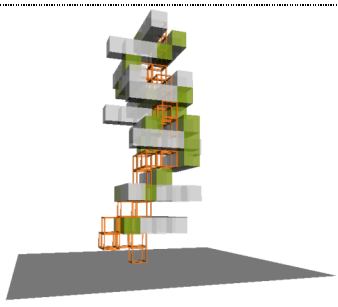
Champ Nº1

Shape code: **17 21 13 18 15 21 4 15 20 21 2 8 5 18 21 12 21 4 15 21 1 4 10 21 6 17 16 21 13 9 17 21 20 8 21 7 3 16 21 8 7 21 3 12 21 21 16 2**

Score: **4**
Appeared in generation: **428**

Nº of apartments: **27**
Nº of circulation blocks: **23**
Volume: **8384 m³**
Total area: **2096 m²**
Apartment area: **1728 m²**
Circulation area: **368 m²**
Height: **64 m**
Footprint: **24 m × 24 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **29.629629 %**
Apartments with views to +i: **14**
Apartments with views to −i: **11**
Apartments with views to +j: **13**
Apartments with views to −j: **10**
Apartments with no views: **0**



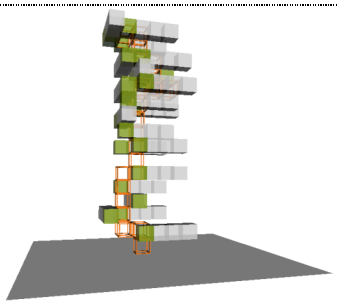
Champ Nº2

Shape code: **19 19 20 21 17 1 5 21 17 21 14 5 1 21 16 21 20 20 17 16 21 11 9 21 16 2 21 18 8 7 21 2 8 11 21 18 12 19 14 21 10 17 21 3 21 16 6 21**

Score: **4**
Appeared in generation: **419**

Nº of apartments: **22**
Nº of circulation blocks: **28**
Volume: **7424 m³**
Total area: **1856 m²**
Apartment area: **1408 m²**
Circulation area: **448 m²**
Height: **64 m**
Footprint: **24 m × 24 m**

Nº of apartments with balconies: **11**
Percentage of apartments with balconies: **50 %**
Apartments with views to +i: **11**
Apartments with views to −i: **12**
Apartments with views to +j: **12**
Apartments with views to −j: **8**
Apartments with no views: **0**



Champ Nº3

Shape code: **5 18 19 18 21 1 10 21 16 21 1 21 17 5 21 21 5 9 21 13 21 14 21 1 20 13 21 10 20 17 13 17 18 19 21 14 8 21 19 10 5 21 21 19 5 18 1 6**

Score: **4**
Appeared in generation: **225**

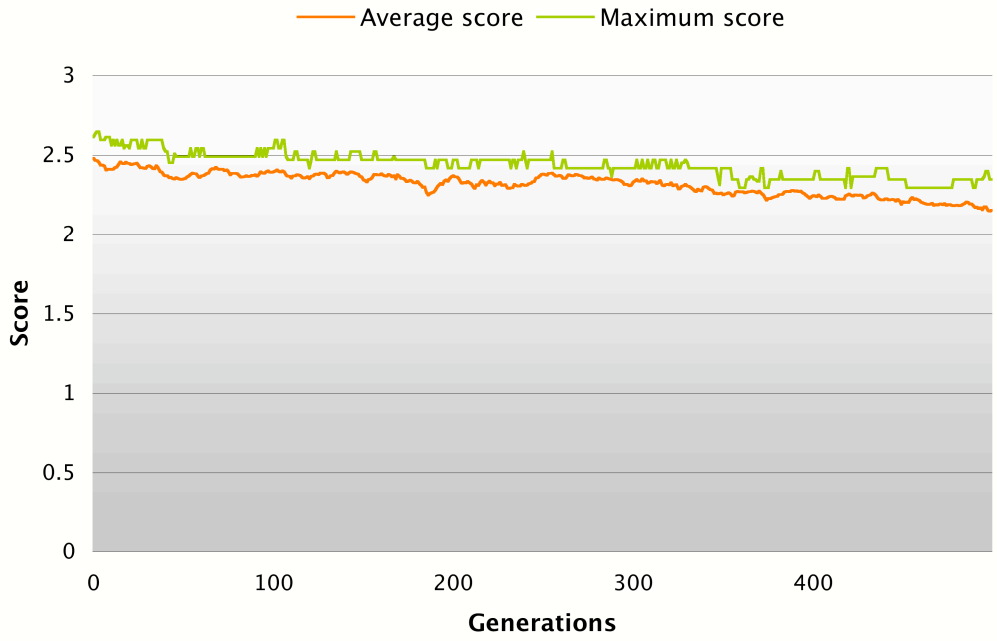
Nº of apartments: **21**
Nº of circulation blocks: **29**
Volume: **7232 m³**
Total area: **1808 m²**
Apartment area: **1344 m²**
Circulation area: **464 m²**
Height: **64 m**
Footprint: **24 m × 24 m**

Nº of apartments with balconies: **6**
Percentage of apartments with balconies: **28.57143 %**
Apartments with views to +i: **4**
Apartments with views to −i: **17**
Apartments with views to +j: **14**
Apartments with views to −j: **10**
Apartments with no views: **0**

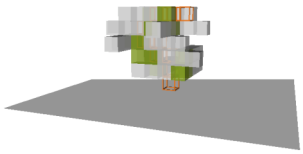
Appendix B
Results for Low-Rise Block Scenario

For the low-rise block the goal height is set to 16 metres, equivalent to four storeys. The footprint of the building is left unconstrained. Instead, to avoid unnecessary spreading of the building horizontally, the goal relating to the percentage of apartment modules in the design was set to the maximum value, 80%. The effect of this is that designs with a more efficient use of circulation will be preferred. The genotype length is 24. The weights for height and percentage of apartments are set to 1, making the highest possible score less than 3 (since the maximum value for the percentage of apartments can only be approached asymptotically, as explained in section 4.4.1).

Low-Rise Population: **50** Set mutation rate: **0.005**



Actual mutation rate: **0.0015** Final average score: **2.1529** Time elapsed: **651 s**



Champ Nº1

Shape code: **11 8 2 21 13 8 19 10 21 19 9 6 5 21 7 17 4 1 3 21 11 9 17 17**

Score: **2.646616**

Appeared in generation: **5**

Nº of apartments: **15**
Nº of circulation blocks: **11**
Volume: **4544 m³**
Total area: **1136 m²**
Apartment area: **960 m²**
Circulation area: **176 m²**
Height: **24 m**
Footprint: **32 m × 28 m**

Nº of apartments with balconies: **2**
Percentage of apartments with balconies: **13.333334 %**

Apartments with views to +i: **5**
Apartments with views to −i: **5**
Apartments with views to +j: **6**
Apartments with views to −j: **4**
Apartments with no views: **0**

Champ Nº2

Shape code: **5 2 1 21 13 8 19 10 21 19 9 6 5 21 7 17 4 1 3 21 11 9 17 17**

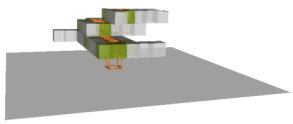
Score: **2.646616**

Appeared in generation: **4**

Nº of apartments: **15**
Nº of circulation blocks: **11**
Volume: **4544 m³**
Total area: **1136 m²**
Apartment area: **960m²**
Circulation area: **176 m²**
Height: **24 m**
Footprint: **36 m × 28 m**

Nº of apartments with balconies: **2**
Percentage of apartments with balconies: **13.333334 %**

Apartments with views to +i: **3**
Apartments with views to −i: **6**
Apartments with views to +j: **7**
Apartments with views to −j: **5**
Apartments with no views: **0**



Champ Nº3

Shape code: **10 8 5 18 18 18 2 18 8 18 18 7 21 6 1 20 17 20 13 17 21 14 5 1**

Score: **2.631579**

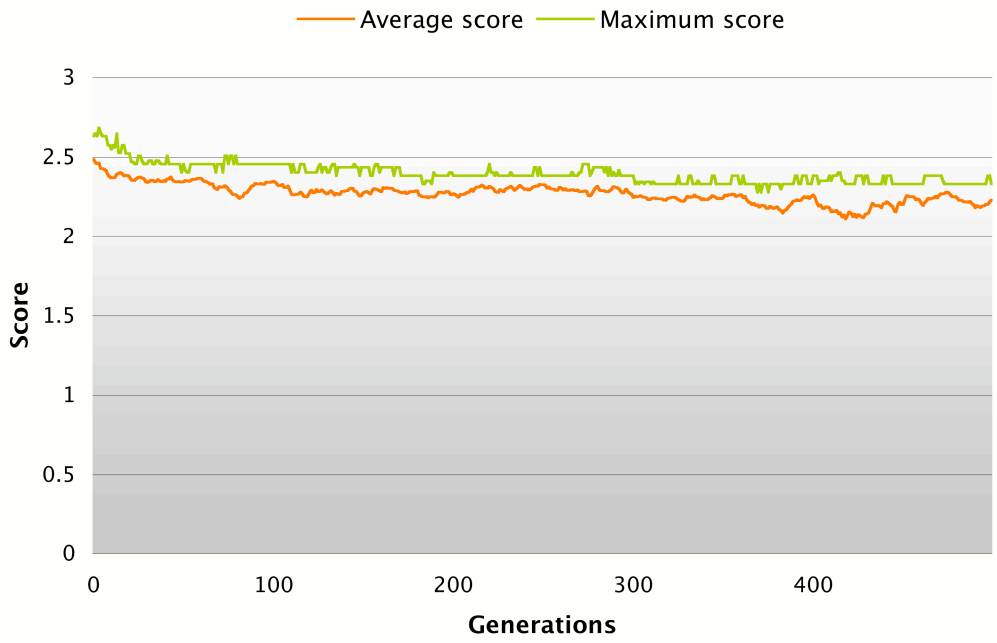
Appeared in generation: **339**

Nº of apartments: **12**
Nº of circulation blocks: **14**
Volume: **3968 m³**
Total area: **992 m²**
Apartment area: **768 m²**
Circulation area: **224 m²**
Height: **24 m**
Footprint: **36 m × 36 m**

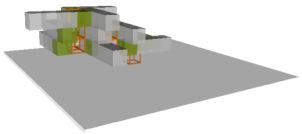
Nº of apartments with balconies: **4**
Percentage of apartments with balconies: **33.333336 %**

Apartments with views to +i: **2**
Apartments with views to −i: **5**
Apartments with views to +j: **4**
Apartments with views to −j: **5**
Apartments with no views: **0**

Low-Rise Population: **50** Set mutation rate: **0.01**



Actual mutation rate: **0.0028** Final average score: **2.2292** Time elapsed: **781 s**



Champ Nº1

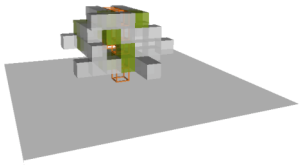
Shape code: **8 7 13 19 19 6 21 12 18 18 8 6 22 15 19 19 10 20 21 10 21 1 3 4**

Score: **2.684211**
Appeared in generation: **5**

Nº of apartments: **13**
Nº of circulation blocks: **13**
Volume: **4160 m³**
Total area: **1040 m²**
Apartment area: **832 m²**
Circulation area: **208 m²**
Height: **16 m**
Footprint: **40 m × 36 m**

Nº of apartments with balconies: **4**
Percentage of apartments with balconies: **30.769232 %**

Apartments with views to +i: **4**
Apartments with views to −i: **4**
Apartments with views to +j: **4**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ Nº2

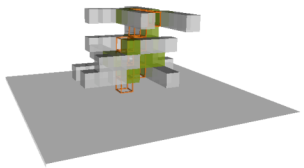
Shape code: **4 2 3 1 21 5 21 13 9 21 11 2 8 18 18 22 22 8 19 6 15 21 14 1**

Score: **2.651378**
Appeared in generation: **6**

Nº of apartments: **15**
Nº of circulation blocks: **11**
Volume: **4544 m³**
Total area: **1136 m²**
Apartment area: **960 m²**
Circulation area: **176 m²**
Height: **20 m**
Footprint: **28 m × 36 m**

Nº of apartments with balconies: **6**
Percentage of apartments with balconies: **40 %**

Apartments with views to +i: **4**
Apartments with views to −i: **5**
Apartments with views to +j: **4**
Apartments with views to −j: **4**
Apartments with no views: **1**



Champ Nº3

Shape code: **7 3 17 4 1 21 16 3 19 2 21 5 15 17 8 21 8 21 8 7 6 20 20 4**

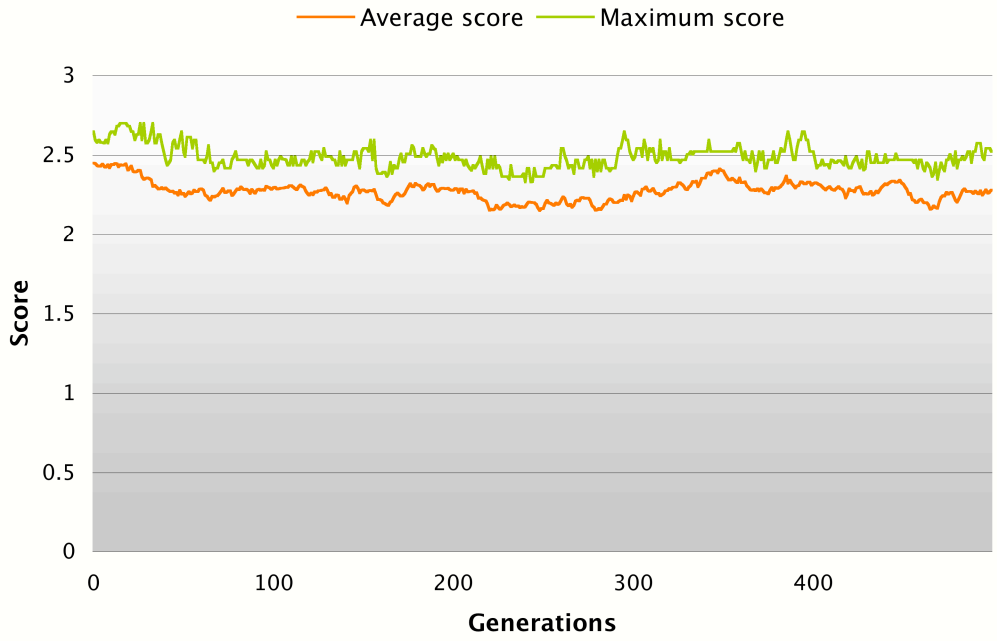
Score: **2.646616**
Appeared in generation: **15**

Nº of apartments: **15**
Nº of circulation blocks: **11**
Volume: **4544 m³**
Total area: **1136 m²**
Apartment area: **960 m²**
Circulation area: **176 m²**
Height: **24 m**
Footprint: **32 m × 36 m**

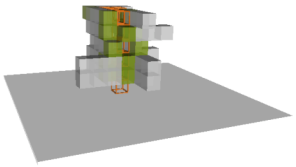
Nº of apartments with balconies: **3**
Percentage of apartments with balconies: **20 %**

Apartments with views to +i: **6**
Apartments with views to −i: **3**
Apartments with views to +j: **5**
Apartments with views to −j: **4**
Apartments with no views: **0**

Low-Rise Population: **50** Set mutation rate: **0.05**



Actual mutation rate: **0.015** Final average score: **2.2798** Time elapsed: **935 s**



Champ №1

Shape code: **3 4 2 1 21 3 1 2 4 21 16 10 21 2 5 8 21 18 10 4 18 18 9 17**

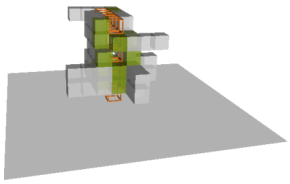
Score: **2.699248**

Appeared in generation: **35**

№ of apartments: **16**
№ of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **28 m × 32 m**

№ of apartments with balconies: **5**
Percentage of apartments with balconies: **31.25 %**

Apartments with views to +i: **5**
Apartments with views to −i: **7**
Apartments with views to +j: **7**
Apartments with views to −j: **3**
Apartments with no views: **0**



Champ №2

Shape code: **3 4 2 1 21 3 1 2 4 21 16 10 21 2 5 8 21 18 10 18 18 18 4 15**

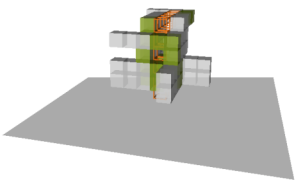
Score: **2.699248**

Appeared in generation: **30**

№ of apartments: **16**
№ of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **28 m × 36 m**

№ of apartments with balconies: **4**
Percentage of apartments with balconies: **25 %**

Apartments with views to +i: **6**
Apartments with views to −i: **6**
Apartments with views to +j: **7**
Apartments with views to −j: **3**
Apartments with no views: **0**



Champ №3

Shape code: **3 4 2 1 21 3 1 2 4 21 16 10 21 2 3 16 21 18 10 18 18 18 4 15**

Score: **2.699248**

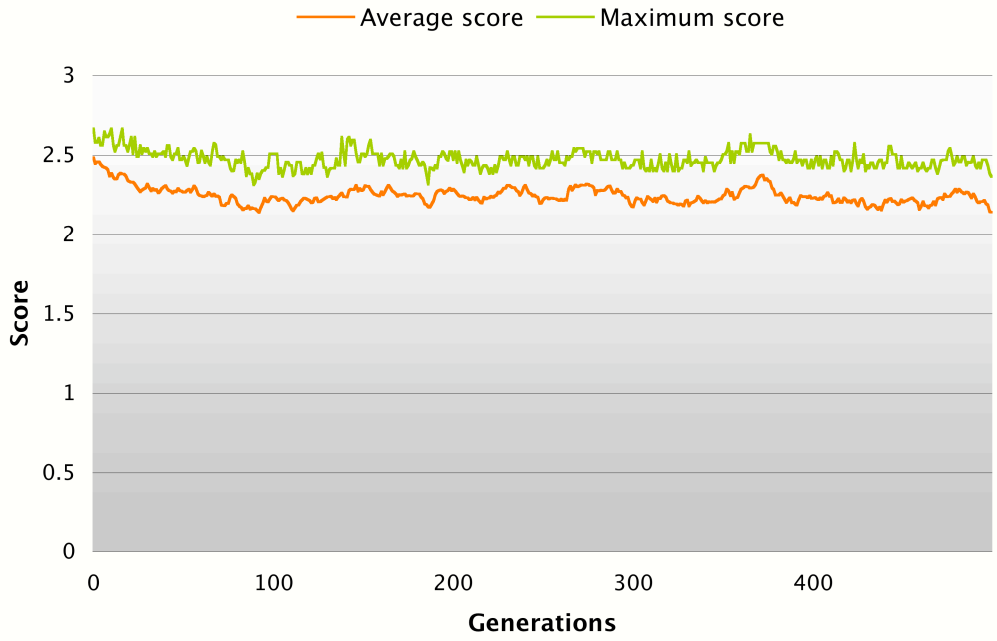
Appeared in generation: **28**

№ of apartments: **16**
№ of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **28 m × 36 m**

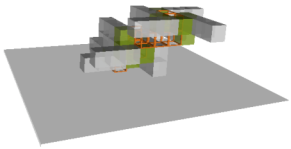
№ of apartments with balconies: **3**
Percentage of apartments with balconies: **18.75 %**

Apartments with views to +i: **6**
Apartments with views to −i: **5**
Apartments with views to +j: **7**
Apartments with views to −j: **3**
Apartments with no views: **0**

Low-Rise Population: **50** Set mutation rate: **0.1**



Actual mutation rate: **0.032** Final average score: **2.1411** Time elapsed: **1089 s**



Champ №1

Shape code: **10 8 20 4 3 21 1 4 20 10 20 3 21 2 20 17 17 8 21 3 4 18 10 1**

Score: **2.665414**

Appeared in generation: **18**

№ of apartments: **14**
№ of circulation blocks: **12**
Volume: **4352 m³**
Total area: **1088 m²**
Apartment area: **896 m²**
Circulation area: **192 m²**
Height: **20 m**
Footprint: **36 m × 44 m**

№ of apartments with balconies: **3**
Percentage of apartments with balconies: **21.428572 %**

Apartments with views to +i: **5**
Apartments with views to −i: **4**
Apartments with views to +j: **5**
Apartments with views to −j: **2**
Apartments with no views: **0**

Champ №2

Shape code: **10 8 20 4 3 21 1 4 20 10 20 3 21 2 17 17 8 20 6 21 13 18 2 1**

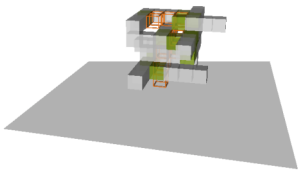
Score: **2.665414**

Appeared in generation: **12**

№ of apartments: **14**
№ of circulation blocks: **12**
Volume: **4352 m³**
Total area: **1088 m²**
Apartment area: **896 m²**
Circulation area: **192 m²**
Height: **20 m**
Footprint: **36 m × 44 m**

№ of apartments with balconies: **4**
Percentage of apartments with balconies: **28.57143 %**

Apartments with views to +i: **4**
Apartments with views to −i: **6**
Apartments with views to +j: **5**
Apartments with views to −j: **2**
Apartments with no views: **0**



Champ №3

Shape code: **5 6 18 9 21 17 8 11 7 21 3 4 18 10 8 18 21 5 7 8 19 19 6 20**

Score: **2.665414**

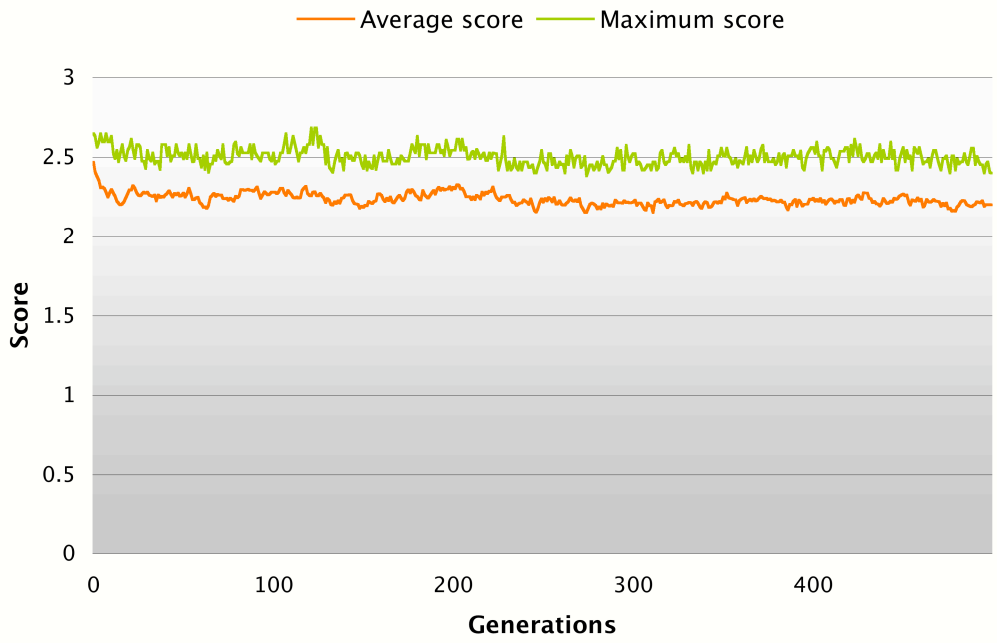
Appeared in generation: **2**

№ of apartments: **14**
№ of circulation blocks: **12**
Volume: **4352 m³**
Total area: **1088 m²**
Apartment area: **896 m²**
Circulation area: **192 m²**
Height: **20 m**
Footprint: **28 m × 40 m**

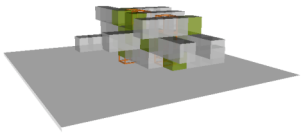
№ of apartments with balconies: **2**
Percentage of apartments with balconies: **14.285715 %**

Apartments with views to +i: **5**
Apartments with views to −i: **4**
Apartments with views to +j: **4**
Apartments with views to −j: **3**
Apartments with no views: **0**

Low-Rise Population: **50** Set mutation rate: **0.5**



Actual mutation rate: **0.1549** Final average score: **2.1999** Time elapsed: **1865 s**



Champ Nº1

Shape code: **7 20 21 21 1 2 20 13 22 2 13 22 20 12 14 20 20 5 21 6 4 7 21 11**

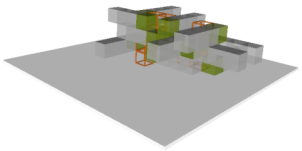
Score: **2.684211**

Appeared in generation: **126**

Nº of apartments: **13**
Nº of circulation blocks: **13**
Volume: **4160 m³**
Total area: **1040 m²**
Apartment area: **832 m²**
Circulation area: **208 m²**
Height: **16 m**
Footprint: **28 m × 44 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **38.46154 %**

Apartments with views to +i: 5
Apartments with views to −i: **7**
Apartments with views to +j: **2**
Apartments with views to −j: **3**
Apartments with no views: **0**



Champ Nº2

Shape code: **7 20 21 21 1 2 20 13 22 2 13 22 20 12 14 20 20 5 21 6 4 7 21 3**

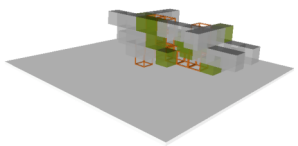
Score: **2.684211**

Appeared in generation: **125**

Nº of apartments: **13**
Nº of circulation blocks: **13**
Volume: **4160 m³**
Total area: **1040 m²**
Apartment area: **832 m²**
Circulation area: **208 m²**
Height: **16 m**
Footprint: **28 m × 44 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **38.46154 %**

Apartments with views to +i: **5**
Apartments with views to −i: **7**
Apartments with views to +j: **2**
Apartments with views to −j: **3**
Apartments with no views: **0**



Champ Nº3

Shape code: **8 20 21 21 1 2 20 6 22 2 16 22 20 12 2 20 20 5 21 6 4 1 21 11**

Score: **2.684211**

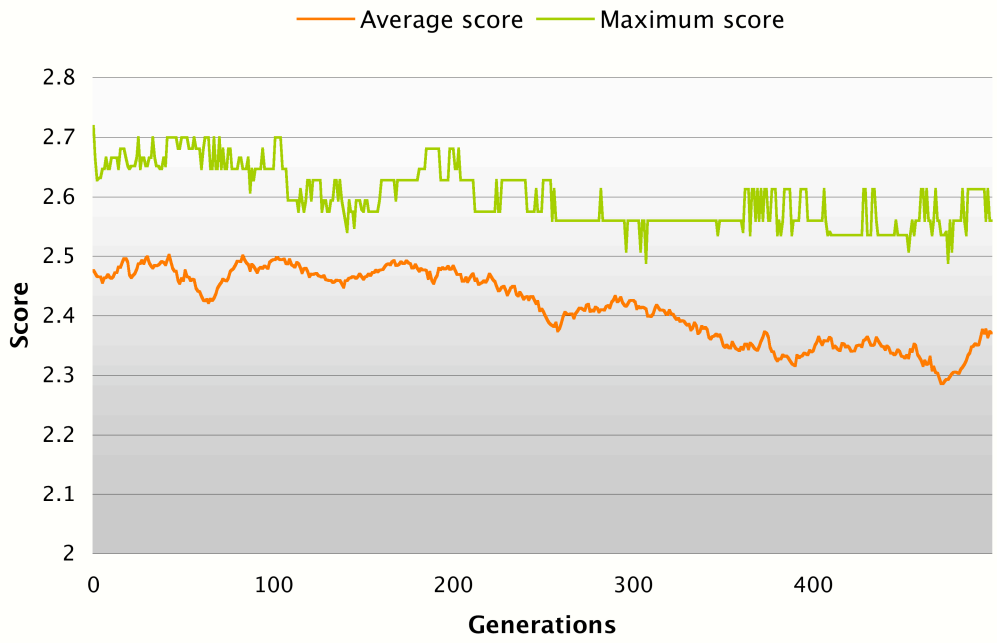
Appeared in generation: **123**

Nº of apartments: **13**
Nº of circulation blocks: **13**
Volume: **4160 m³**
Total area: **1040 m²**
Apartment area: **832 m²**
Circulation area: **208 m²**
Height: **16 m**
Footprint: **24 m × 48 m**

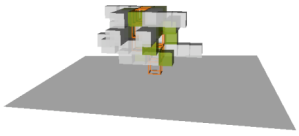
Nº of apartments with balconies: **4**
Percentage of apartments with balconies: **30.769232 %**

Apartments with views to +i: **5**
Apartments with views to −i: **8**
Apartments with views to +j: **3**
Apartments with views to −j: **1**
Apartments with no views: **0**

Low-Rise Population: **200** Set mutation rate: **0.005**



Actual mutation rate: **0.001** Final average score: **2.3708** Time elapsed: **3871 s**

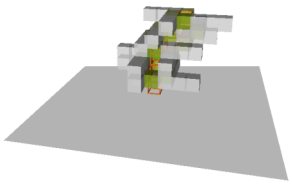


Champ Nº1

Shape code: **11 18 1 10 21 15 21 3 7 8 19 21 7 12 20 20 14 5 22 6 4 22 11 7**

Score: **2.718045** Appeared in generation: **2**

Nº of apartments: **15** Nº of apartments with balconies: **5**
Nº of circulation blocks: **11** Percentage of apartments with balconies: **33.333336 %**
Volume: **4544 m³**
Total area: **1136 m²** Apartments with views to +i: **8**
Apartment area: **960 m²** Apartments with views to −i: **3**
Circulation area: **176 m²** Apartments with views to +j: **5**
Height: **20 m** Apartments with views to −j: **5**
Footprint: **32 m × 36 m** Apartments with no views: **0**

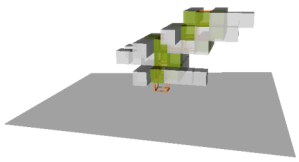


Champ Nº2

Shape code: **14 5 8 18 7 21 12 6 7 21 5 18 14 17 8 21 5 6 17 8 21 7 6 5**

Score: **2.699248** Appeared in generation: **103**

Nº of apartments: **16** Nº of apartments with balconies: **5**
Nº of circulation blocks: **10** Percentage of apartments with balconies: **31.25 %**
Volume: **4736 m³**
Total area: **1184 m²** Apartments with views to +i: **5**
Apartment area: **1024 m²** Apartments with views to −i: **5**
Circulation area: **160 m²** Apartments with views to +j: **4**
Height: **24 m** Apartments with views to −j: **6**
Footprint: **36 m × 32 m** Apartments with no views: **0**



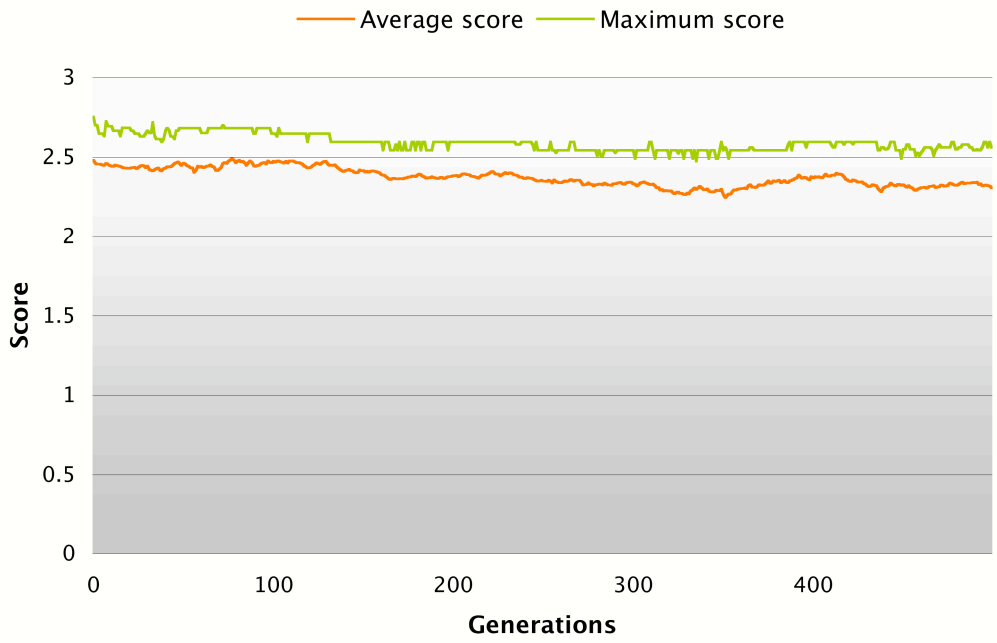
Champ Nº3

Shape code: **14 5 8 18 21 16 7 3 21 9 5 6 17 17 8 21 5 6 17 8 21 2 8 5**

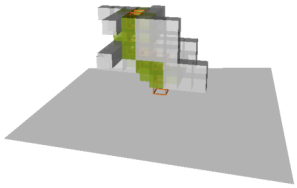
Score: **2.699248** Appeared in generation: **72**

Nº of apartments: **16** Nº of apartments with balconies: **6**
Nº of circulation blocks: **10** Percentage of apartments with balconies: **37.5 %**
Volume: **4736 m³**
Total area: **1184 m²** Apartments with views to +i: **5**
Apartment area: **1024 m²** Apartments with views to −i: **6**
Circulation area: **160 m²** Apartments with views to +j: **7**
Height: **24 m** Apartments with views to −j: **4**
Footprint: **40 m × 28 m** Apartments with no views: **0**

Low-Rise Population: **200** Set mutation rate: **0.01**



Actual mutation rate: **0.0023** Final average score: **2.3054** Time elapsed: **3802 s**



Champ №1

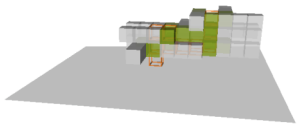
Shape code: **5 2 8 21 5 19 1 2 3 21 13 1 19 6 21 5 2 8 21 14 8 5 18 7**

Score: **2.751880**
Appeared in generation: **2**

№ of apartments: **17**
№ of circulation blocks: **9**
Volume: **4928 m³**
Total area: **1232 m²**
Apartment area: **1088 m²**
Circulation area: **144 m²**
Height: **24 m**
Footprint: **36 m × 28 m**

№ of apartments with balconies: **4**
Percentage of apartments with balconies: **23.529411 %**

Apartments with views to +i: **3**
Apartments with views to −i: **8**
Apartments with views to +j: **8**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ №2

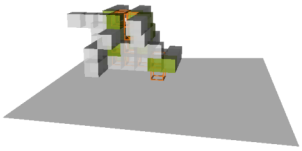
Shape code: **15 5 6 21 17 7 3 8 22 17 17 18 15 17 5 8 21 10 1 5 21 8 14 5**

Score: **2.722807**
Appeared in generation: **9**

№ of apartments: **15**
№ of circulation blocks: **11**
Volume: **4544 m³**
Total area: **1136 m²**
Apartment area: **960 m²**
Circulation area: **176 m²**
Height: **16 m**
Footprint: **40 m × 32 m**

№ of apartments with balconies: **5**
Percentage of apartments with balconies: **33.333336 %**

Apartments with views to +i: **3**
Apartments with views to −i: **2**
Apartments with views to +j: **7**
Apartments with views to −j: **4**
Apartments with no views: **1**



Champ №3

Shape code: **16 19 11 19 19 1 6 21 6 5 15 17 8 21 12 6 7 21 4 19 3 18 2 8**

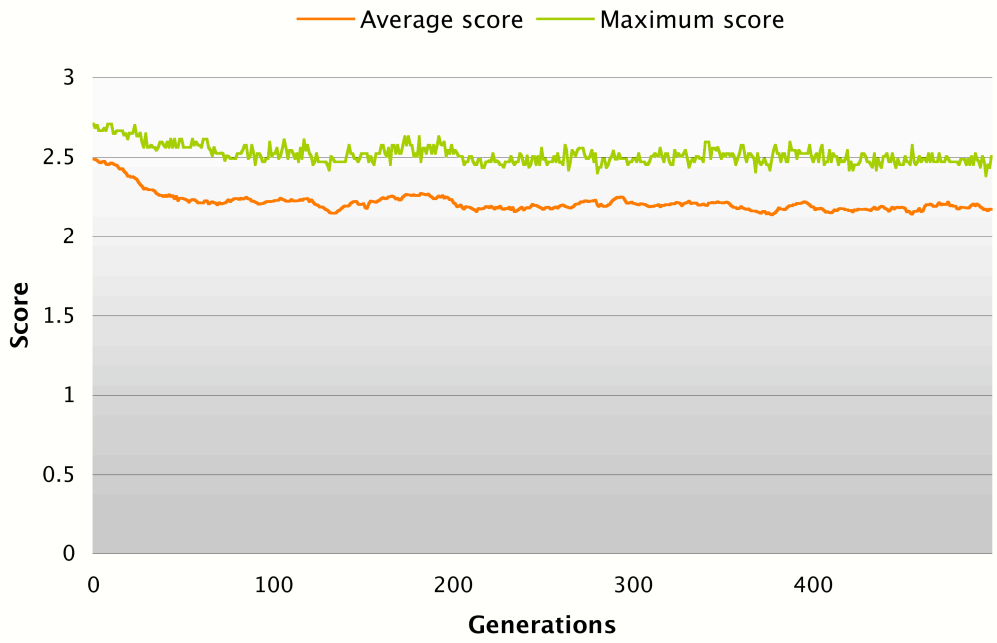
Score: **2.718045**
Appeared in generation: **35**

№ of apartments: **15**
№ of circulation blocks: **11**
Volume: **4544 m³**
Total area: **1136 m²**
Apartment area: **960 m²**
Circulation area: **176 m²**
Height: **20 m**
Footprint: **32 m × 32 m**

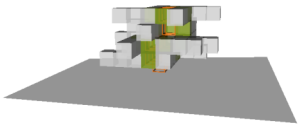
№ of apartments with balconies: **7**
Percentage of apartments with balconies: **46.666668 %**

Apartments with views to +i: **5**
Apartments with views to −i: **5**
Apartments with views to +j: **4**
Apartments with views to −j: **6**
Apartments with no views: **0**

Low-Rise Population: **200** Set mutation rate: **0.05**



Actual mutation rate: **0.0175** Final average score: **2.1703** Time elapsed: **4088 s**



Champ Nº1

Shape code: **14 4 1 20 20 3 21 5 6 18 7 17 8 5 21 8 6 7 21 4 1 10 20 3**

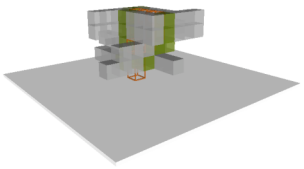
Score: **2.708177**

Appeared in generation: **10**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **20 m**
Footprint: **32 m × 32 m**

Nº of apartments with balconies: **3**
Percentage of apartments with balconies: **18.75 %**

Apartments with views to +i: **4**
Apartments with views to −i: **4**
Apartments with views to +j: **4**
Apartments with views to −j: **5**
Apartments with no views: **1**



Champ Nº2

Shape code: **2 3 1 4 21 17 16 3 18 2 21 8 6 7 5 21 7 6 12 20 20 5 22 5**

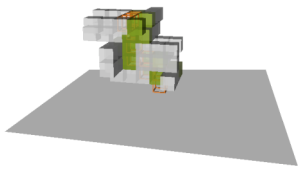
Score: **2.708177**

Appeared in generation: **2**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **20 m**
Footprint: **32 m × 32 m**

Nº of apartments with balconies: **3**
Percentage of apartments with balconies: **18.75 %**

Apartments with views to +i: **4**
Apartments with views to −i: **4**
Apartments with views to +j: **5**
Apartments with views to −j: **4**
Apartments with no views: **1**



Champ Nº3

Shape code: **12 19 11 7 21 7 13 8 19 6 21 6 5 1 21 3 16 7 21 7 8 20 20 11**

Score: **2.699248**

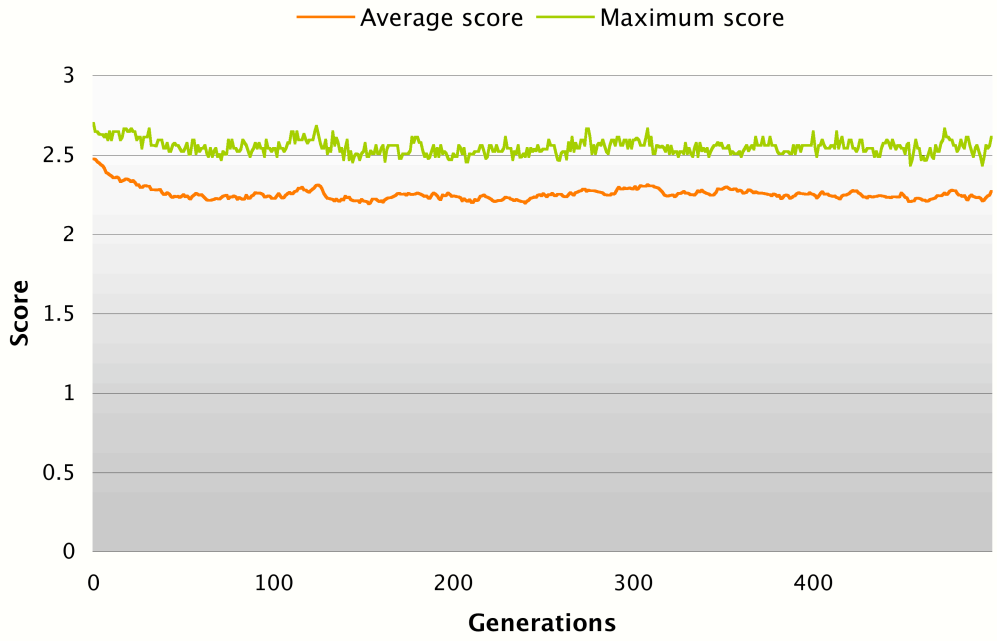
Appeared in generation: **25**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **28 m × 28 m**

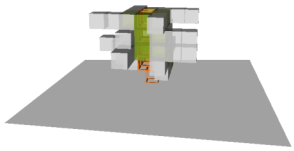
Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **31.25 %**

Apartments with views to +i: **3**
Apartments with views to −i: **5**
Apartments with views to +j: **6**
Apartments with views to −j: **7**
Apartments with no views: **0**

Low-Rise Population: **200** Set mutation rate: **0.1**



Actual mutation rate: **0.0317** Final average score: **2.2710** Time elapsed: **4393 s**



Champ Nº1

Shape code: **12 21 6 7 8 20 4 20 20 21 8 6 5 7 21 5 14 8 21 3 12 2 18 1**

Score: **2.699248**
Appeared in generation: **2**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **28 m × 40 m**

Nº of apartments with balconies: **2**
Percentage of apartments with balconies: **12.5%**

Apartments with views to +i: **6**
Apartments with views to −i: **5**
Apartments with views to +j: **3**
Apartments with views to −j: **5**
Apartments with no views: **0**

Champ Nº2

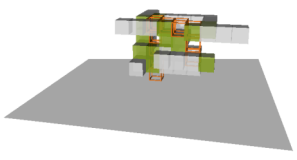
Shape code: **11 8 19 7 21 5 1 10 17 21 5 6 8 21 2 12 21 2 4 1 3 21 14 12**

Score: **2.680451**
Appeared in generation: **126**

Nº of apartments: **17**
Nº of circulation blocks: **9**
Volume: **4928m³**
Total area: **1232 m²**
Apartment area: **1088 m²**
Circulation area: **144 m²**
Height: **28 m**
Footprint: **32 m × 28 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **29.411766 %**

Apartments with views to +i: **6**
Apartments with views to −i: **5**
Apartments with views to +j: **6**
Apartments with views to −j: **7**
Apartments with no views: **0**



Champ Nº3

Shape code: **14 5 21 5 8 10 21 2 16 21 1 2 20 17 3 17 20 17 5 8 22 2 22 11**

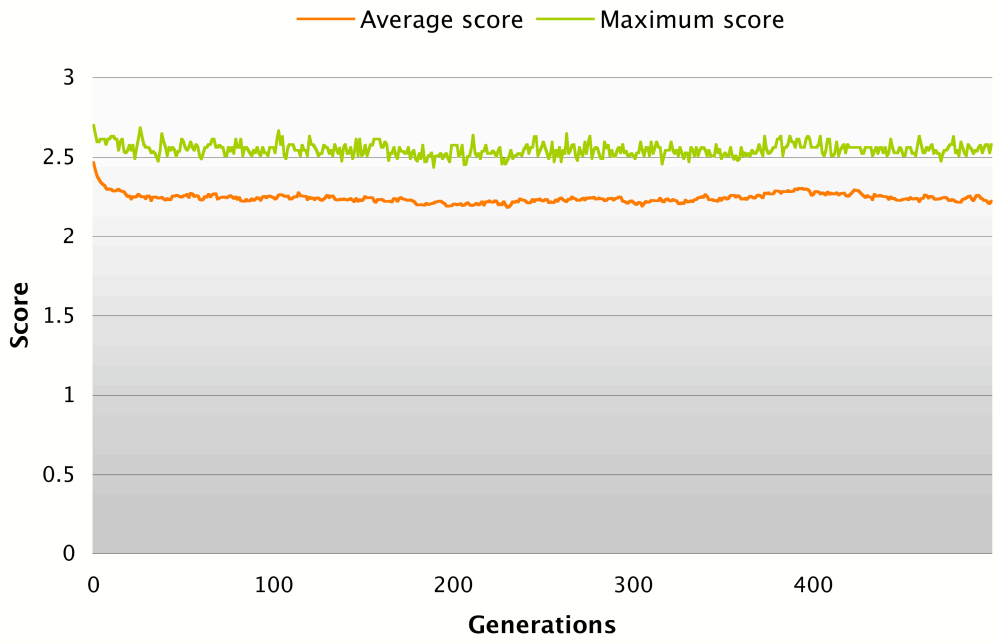
Score: **2.665414**
Appeared in generation: **475**

Nº of apartments: **14**
Nº of circulation blocks: **12**
Volume: **4352 m³**
Total area: **1088 m²**
Apartment area: **896 m²**
Circulation area: **192 m²**
Height: **20 m**
Footprint: **36 m × 28 m**

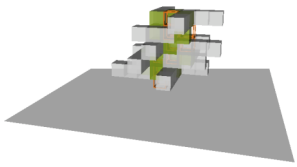
Nº of apartments with balconies: **4**
Percentage of apartments with balconies: **28.57143 %**

Apartments with views to +i: **4**
Apartments with views to −i: **7**
Apartments with views to +j: **9**
Apartments with views to −j: **3**
Apartments with no views: **0**

Low-Rise Population: **200** Set mutation rate: **0.5**



Actual mutation rate: **0.154** Final average score: **2.2235** Time elapsed: **6180 s**



Champ Nº1

Shape code: **2 1 4 3 21 15 5 6 17 8 21 9 5 6 17 21 19 10 20 11 21 2 1 4**

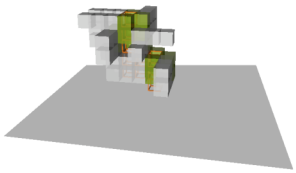
Score: **2.699248**

Appeared in generation: **2**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **32 m × 32 m**

Nº of apartments with balconies: **4**
Percentage of apartments with balconies: **25 %**

Apartments with views to +i: **4**
Apartments with views to −i: **6**
Apartments with views to +j: **5**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ Nº2

Shape code: **4 7 3 21 4 3 7 19 22 19 21 21 12 6 7 21 8 6 7 5 21 8 7 3**

Score: **2.699248**

Appeared in generation: **2**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **28 m × 28 m**

Nº of apartments with balconies: **3**
Percentage of apartments with balconies: **18.75 %**

Apartments with views to +i: **8**
Apartments with views to −i: **2**
Apartments with views to +j: **5**
Apartments with views to −j: **8**
Apartments with no views: **0**

Champ Nº3

Shape code: **2 5 1 21 7 11 21 6 15 20 20 4 22 4 20 22 19 7 13 21 20 20 2 5**

Score: **2.684211**

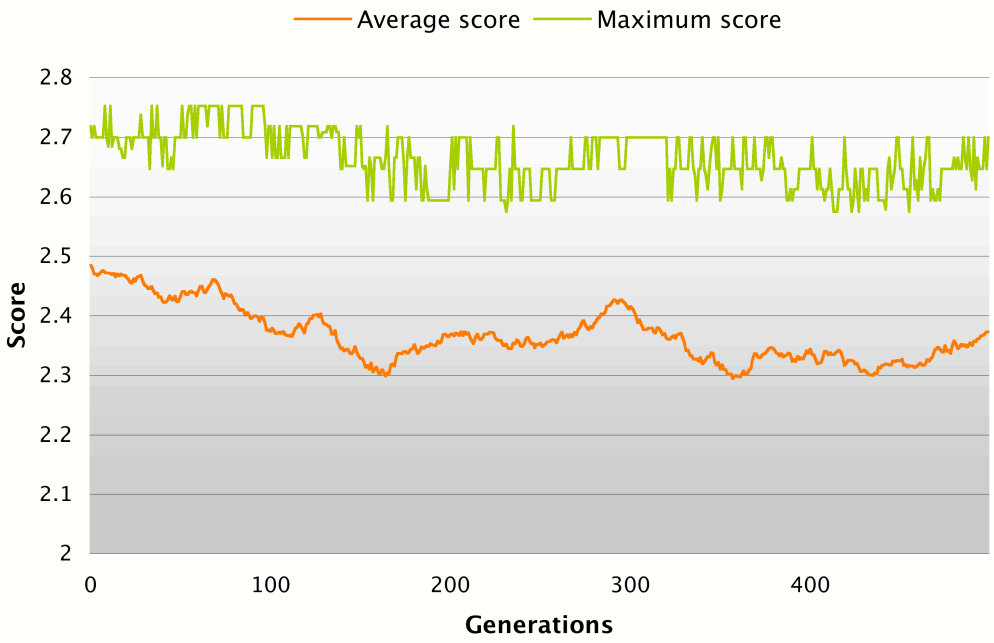
Appeared in generation: **28**

Nº of apartments: **13**
Nº of circulation blocks: **13**
Volume: **4160 m³**
Total area: **1040 m²**
Apartment area: **832 m²**
Circulation area: **208 m²**
Height: **16 m**
Footprint: **32 m × 40 m**

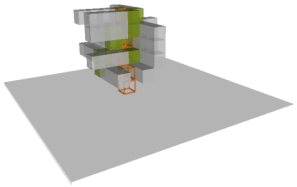
Nº of apartments with balconies: **3**
Percentage of apartments with balconies: **23.076923 %**

Apartments with views to +i: **6**
Apartments with views to −i: **6**
Apartments with views to +j: **3**
Apartments with views to −j: **4**
Apartments with no views: **0**

Low-Rise Population: **500** Set mutation rate: **0.005**



Actual mutation rate: **0.0011** Final average score: **2.373** Time elapsed: **8905 s**



Champ №1

Shape code: **18 14 12 18 1 21 2 17 1 4 3 21 2 1 4 21 2 4 3 1 21 7 8 3**

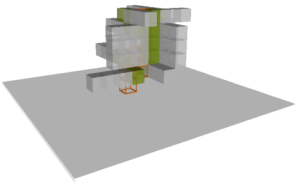
Score: **2.75188**

Appeared in generation: **94**

№ of apartments: **17**
№ of circulation blocks: **9**
Volume: **4928 m³**
Total area: **1232 m²**
Apartment area: **1088 m²**
Circulation area: **144 m²**
Height: **24 m**
Footprint: **28 m × 28 m**

№ of apartments with balconies: **5**
Percentage of apartments with balconies: **29.411766 %**

Apartments with views to +i: **6**
Apartments with views to −i: **5**
Apartments with views to +j: **5**
Apartments with views to −j: **6**
Apartments with no views: **0**



Champ №2

Shape code: **3 17 12 18 1 21 2 17 1 4 3 21 3 15 4 21 2 4 3 1 21 14 8 5**

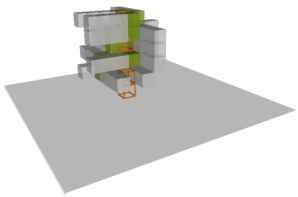
Score: **2.75188**

Appeared in generation: **93**

№ of apartments: **17**
№ of circulation blocks: **9**
Volume: **4928 m³**
Total area: **1232 m²**
Apartment area: **1088 m²**
Circulation area: **144 m²**
Height: **24 m**
Footprint: **36 m × 28 m**

№ of apartments with balconies: **4**
Percentage of apartments with balconies: **23.529411 %**

Apartments with views to +i: **6**
Apartments with views to −i: **4**
Apartments with views to +j: **5**
Apartments with views to −j: **5**
Apartments with no views: **0**



Champ №3

Shape code: **18 14 12 18 15 21 2 17 1 4 3 21 2 1 4 21 2 4 3 1 21 7 8 3**

Score: **2.75188**

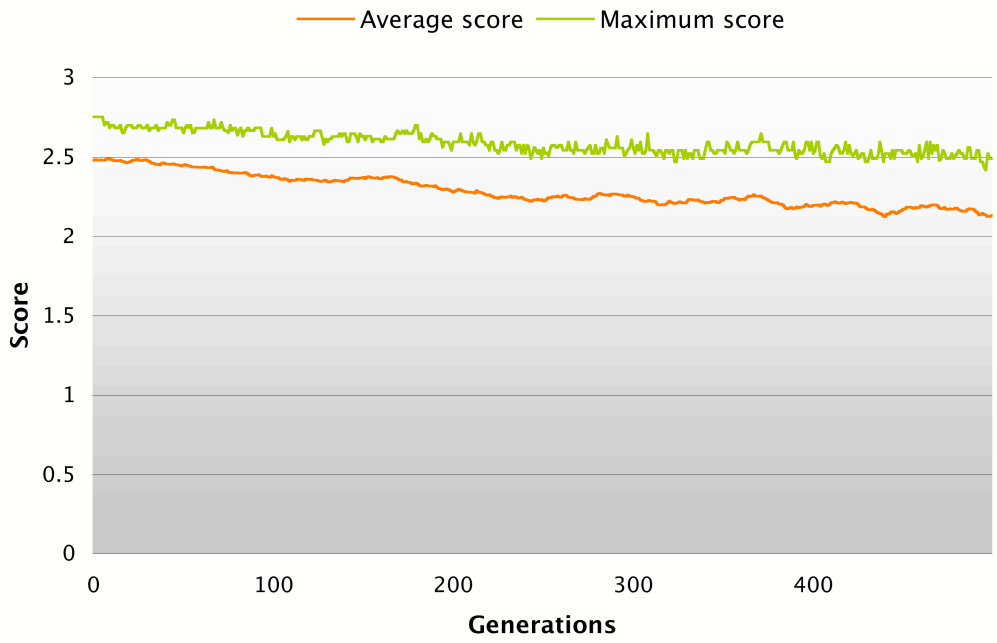
Appeared in generation: **93**

№ of apartments: **17**
№ of circulation blocks: **9**
Volume: **4928 m³**
Total area: **1232 m²**
Apartment area: **1088 m²**
Circulation area: **144 m²**
Height: **24 m**
Footprint: **28 m × 28 m**

№ of apartments with balconies: **5**
Percentage of apartments with balconies: **29.411766 %**

Apartments with views to +i: **7**
Apartments with views to −i: **4**
Apartments with views to +j: **5**
Apartments with views to −j: **5**
Apartments with no views: **0**

Low-Rise Population: **500** Set mutation rate: **0.01**



Actual mutation rate: **0.0029** Final average score: **2.1328** Time elapsed: **8497 s**

Champ №1

Shape code: **10 4 1 21 6 5 18 2 16 21 2 12 3 18 21 11 2 8 21 15 6 5 17 8**

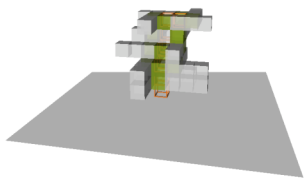
Score: **2.75188**

Appeared in generation: **6**

№ of apartments: **17**
№ of circulation blocks: **9**
Volume: **4928 m³**
Total area: **1232 m²**
Apartment area: **1088 m²**
Circulation area: **144 m²**
Height: **24 m**
Footprint: **28 m × 36 m**

№ of apartments with balconies: **4**
Percentage of apartments with balconies: **23.529411 %**

Apartments with views to +i: **6**
Apartments with views to −i: **6**
Apartments with views to +j: **5**
Apartments with views to −j: **3**
Apartments with no views: **0**



Champ №2

Shape code: **9 6 5 21 6 5 18 2 16 21 2 12 3 18 21 11 2 8 21 15 6 5 17 8**

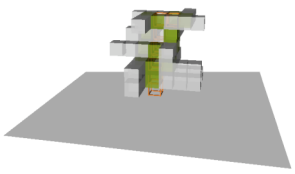
Score: **2.75188**

Appeared in generation: **5**

№ of apartments: **17**
№ of circulation blocks: **9**
Volume: **4928 m³**
Total area: **1232 m²**
Apartment area: **1088 m²**
Circulation area: **144 m²**
Height: **24 m**
Footprint: **28 m × 36 m**

№ of apartments with balconies: **4**
Percentage of apartments with balconies: **23.529411 %**

Apartments with views to +i: **5**
Apartments with views to −i: **7**
Apartments with views to +j: **5**
Apartments with views to −j: **3**
Apartments with no views: **0**



Champ №3

Shape code: **5 1 6 21 6 5 18 2 16 21 2 12 3 18 21 11 2 8 21 15 6 5 17 8**

Score: **2.75188**

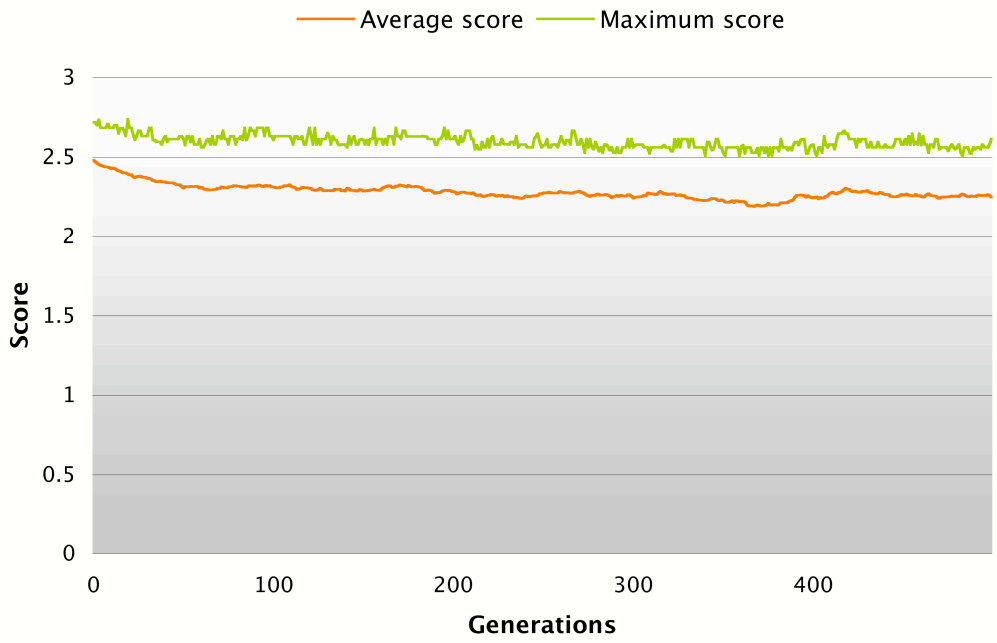
Appeared in generation: **4**

№ of apartments: **17**
№ of circulation blocks: **9**
Volume: **4928 m³**
Total area: **1232 m²**
Apartment area: **1088 m²**
Circulation area: **144 m²**
Height: **24 m**
Footprint: **28 m × 36 m**

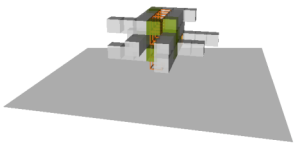
№ of apartments with balconies: **4**
Percentage of apartments with balconies: **23.529411 %**

Apartments with views to +i: **5**
Apartments with views to −i: **7**
Apartments with views to +j: **5**
Apartments with views to −j: **4**
Apartments with no views: **0**

Low-Rise Population: **500** Set mutation rate: **0.05**



Actual mutation rate: **0.0139** Final average score: **2.2481** Time elapsed: **9439 s**



Champ Nº1

Shape code: **4 1 3 21 6 4 21 3 8 18 22 8 2 18 21 14 18 9 22 22 7 11 17 1**

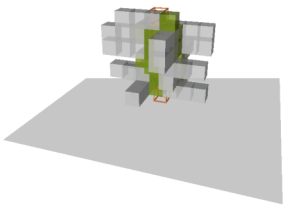
Score: **2.736842**

Appeared in generation: **21**

Nº of apartments: **14**
Nº of circulation blocks: **12**
Volume: **4352 m³**
Total area: **1088m²**
Apartment area: **896 m²**
Circulation area: **192 m²**
Height: **16 m**
Footprint: **32 m × 32 m**

Nº of apartments with balconies: **3**
Percentage of apartments with balconies: **21.428572 %**

Apartments with views to +i: **6**
Apartments with views to −i: **5**
Apartments with views to +j: **3**
Apartments with views to −j: **3**
Apartments with no views: **0**



Champ Nº2

Shape code: **8 5 7 6 21 12 3 2 21 10 8 20 5 21 3 1 4 21 3 1 4 2 21 11**

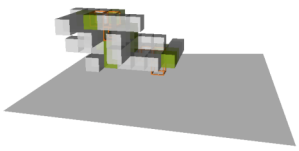
Score: **2.733083**

Appeared in generation: **5**

Nº of apartments: **18**
Nº of circulation blocks: **8**
Volume: **5120 m³**
Total area: **1280 m²**
Apartment area: **1152 m²**
Circulation area: **128 m²**
Height: **28 m**
Footprint: **28 m × 32 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **27.777779 %**

Apartments with views to +i: **6**
Apartments with views to −i: **6**
Apartments with views to +j: **7**
Apartments with views to −j: **5**
Apartments with no views: **0**



Champ Nº3

Shape code: **8 11 7 19 19 19 6 20 5 21 4 1 2 3 21 7 4 6 20 21 7 19 6 5**

Score: **2.718045**

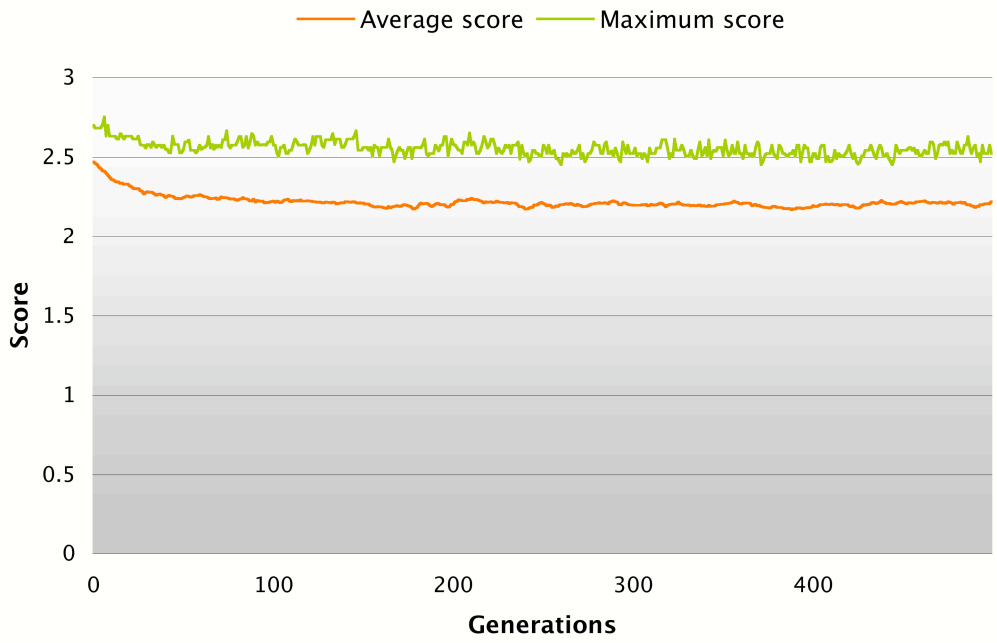
Appeared in generation: **2**

Nº of apartments: **15**
Nº of circulation blocks: **11**
Volume: 4544 **m³**
Total area: **1136 m²**
Apartment area: **960 m²**
Circulation area: **176 m²**
Height: **20 m**
Footprint: **32 m × 36 m**

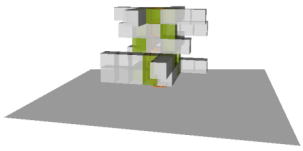
Nº of apartments with balconies: **6**
Percentage of apartments with balconies: **40 %**

Apartments with views to +i: **6**
Apartments with views to −i: **4**
Apartments with views to +j: **4**
Apartments with views to −j: **6**
Apartments with no views: **0**

Low-Rise Population: **500** Set mutation rate: **0.1**



Actual mutation rate: **0.0344** Final average score: **2.2187** Time elapsed: **9274 s**



Champ Nº1

Shape code: **1 4 20 20 3 2 21 3 4 18 15 21 13 2 8 21 11 1 2 21 8 7 5 6**

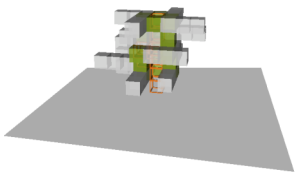
Score: **2.75188**

Appeared in generation: **8**

Nº of apartments: **17**
Nº of circulation blocks: **9**
Volume: **4928 m³**
Total area: **1232 m²**
Apartment area: **1088m²**
Circulation area: **144 m²**
Height: **24 m**
Footprint: **28 m × 32 m**

Nº of apartments with balconies: **4**
Percentage of apartments with balconies: **23.529411 %**

Apartments with views to +i: **6**
Apartments with views to −i: **5**
Apartments with views to +j: **5**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ Nº2

Shape code: **12 7 6 21 8 19 15 3 21 20 5 17 8 18 7 21 11 1 2 21 8 7 5 6**

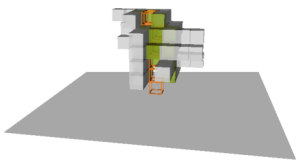
Score: **2.699248**

Appeared in generation: **10**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **32 m × 28 m**

Nº of apartments with balconies: **3**
Percentage of apartments with balconies: **18.75 %**

Apartments with views to +i: **6**
Apartments with views to −i: **4**
Apartments with views to +j: **5**
Apartments with views to −j:6
Apartments with no views: **0**



Champ Nº3

Shape code: **14 8 21 4 14 1 21 8 5 14 18 7 21 8 5 6 18 7 19 21 16 14 18 7**

Score: **2.699248**

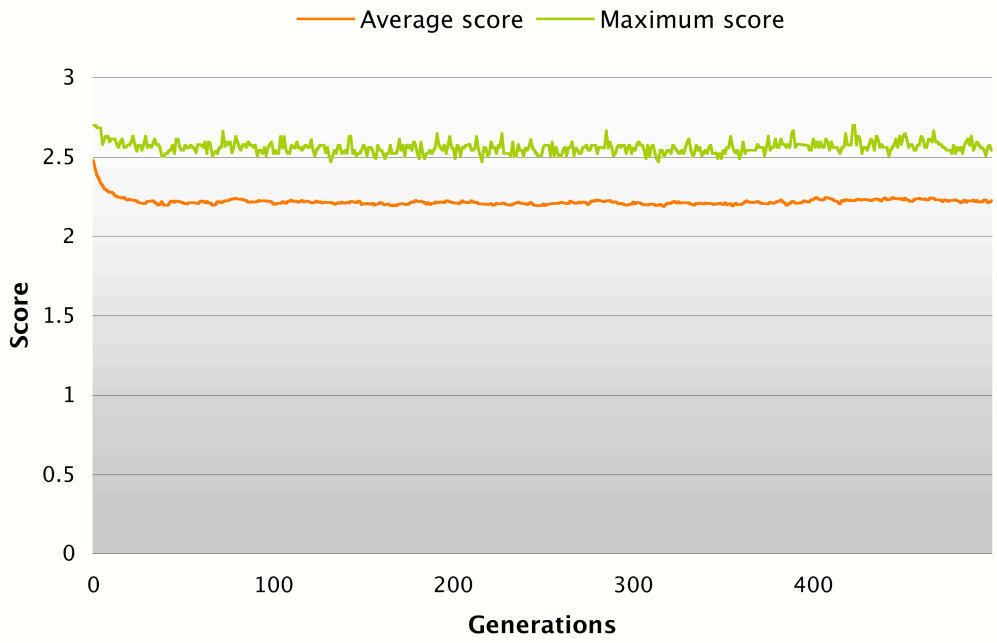
Appeared in generation: **7**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **32 m × 32 m**

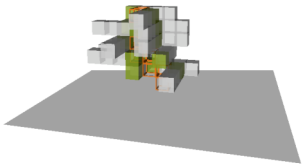
Nº of apartments with balconies: **3**
Percentage of apartments with balconies: 18.75 %

Apartments with views to +i: **5**
Apartments with views to −i: **5**
Apartments with views to +j: **4**
Apartments with views to −j: **6**
Apartments with no views: **0**

Low-Rise Population: **500** Set mutation rate: **0.5**



Actual mutation rate: **0.166** Final average score: **2.2257** Time elapsed: **10401 s**



Champ Nº1

Shape code: **4 19 10 1 21 16 19 7 3 21 6 18 1 4 2 21 2 4 1 3 21 9 20 4**

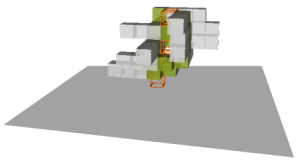
Score: **2.699248**

Appeared in generation: **425**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **32 m × 32 m**

Nº of apartments with balconies: **6**
Percentage of apartments with balconies: **37.5 %**

Apartments with views to +i: **5**
Apartments with views to −i: **5**
Apartments with views to +j: **5**
Apartments with views to −j: **5**
Apartments with no views: **0**



Champ Nº2

Shape code: **8 3 2 21 2 20 17 8 3 21 6 18 1 4 2 21 2 4 1 3 21 9 20 4**

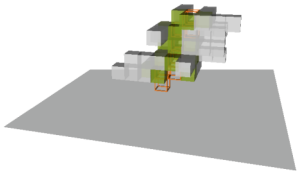
Score: **2.699248**

Appeared in generation: **424**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **32 m × 32 m**

Nº of apartments with balconies: **4**
Percentage of apartments with balconies: **25 %**

Apartments with views to +i: **7**
Apartments with views to −i: **6**
Apartments with views to +j: **6**
Apartments with views to −j: **3**
Apartments with no views: **0**



Champ Nº3

Shape code: **2 3 17 8 21 3 12 2 18 1 21 2 5 8 21 6 1 5 17 21 2 20 5 8**

Score: **2.699248**

Appeared in generation: **3**

Nº of apartments: **16**
Nº of circulation blocks: **10**
Volume: **4736 m³**
Total area: **1184 m²**
Apartment area: **1024 m²**
Circulation area: **160 m²**
Height: **24 m**
Footprint: **36 m × 28 m**

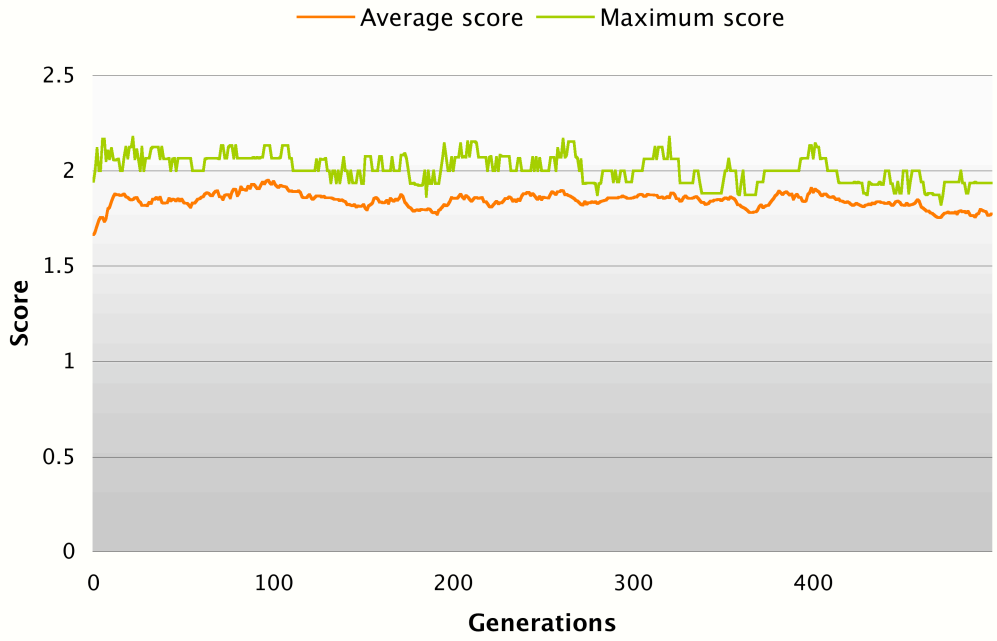
Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **43.75 %**

Apartments with views to +i: **5**
Apartments with views to −i: **8**
Apartments with views to +j: **8**
Apartments with views to −j: **3**
Apartments with no views: **0**

Appendix C
Results for Views and Balconies Scenario

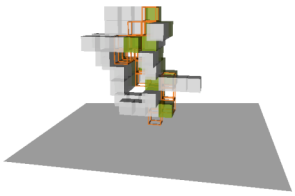
This design scenario requires 100% of apartments to afford views towards the positive *i* direction. To further maximise enjoyment of the views, it is required that 100% of apartments have balconies. Weights for the views and balcony criteria are set to 1. The genotype length is set to 32. The maximum score for this set of criteria is 3.

Views and Balconies Population: **50** Set mutation rate: **0.005**



Actual mutation rate: **0.0012** Final average score: **1.7758** Time elapsed: **690 s**

Champ Nº1



Shape code: **11 7 17 21 14 21 7 19 5 19 6 21 6 12 7 21 4 19 20 14 20 21 11 15 17 8 21 17 4 7 21 7**

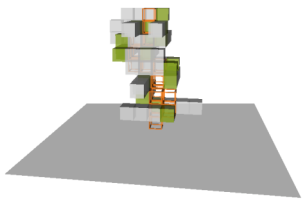
Score: **2.176471**
Appeared in generation: **322**

Nº of apartments: **17**
Nº of circulation blocks: **17**
Volume: **5440 m³**
Total area: **1360 m²**
Apartment area: **1088 m²**
Circulation area: **272 m²**
Height: **36 m**
Footprint: **32 m x 28 m**

Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **52.941177 %**

Apartments with views to +i: **11**
Apartments with views to -i: **4**
Apartments with views to +j: **4**
Apartments with views to -j: **11**
Apartments with no views: **0**

Champ Nº2



Shape code: **11 1 17 21 19 2 18 21 6 18 16 21 11 15 19 19 21 8 13 2 18 21 13 15 17 8 21 6 4 7 21 15**

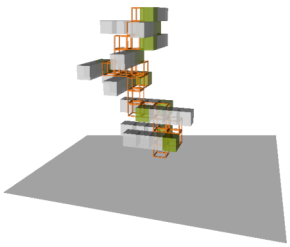
Score: **2.176471**
Appeared in generation: **24**

Nº of apartments: **17**
Nº of circulation blocks: **17**
Volume: **5440 m³**
Total area: **1360 m²**
Apartment area: **1088 m²**
Circulation area: **272 m²**
Height: **36 m**
Footprint: **32 m x 36 m**

Nº of apartments with balconies: **11**
Percentage of apartments with balconies: **64.705887 %**

Apartments with views to +i: **9**
Apartments with views to -i: **7**
Apartments with views to +j: **7**
Apartments with views to -j: **7**
Apartments with no views: **0**

Champ Nº3



Shape code: **11 7 17 21 19 21 7 19 5 19 21 21 14 18 7 21 19 19 20 14 17 21 4 21 17 8 21 18 4 7 21 16**

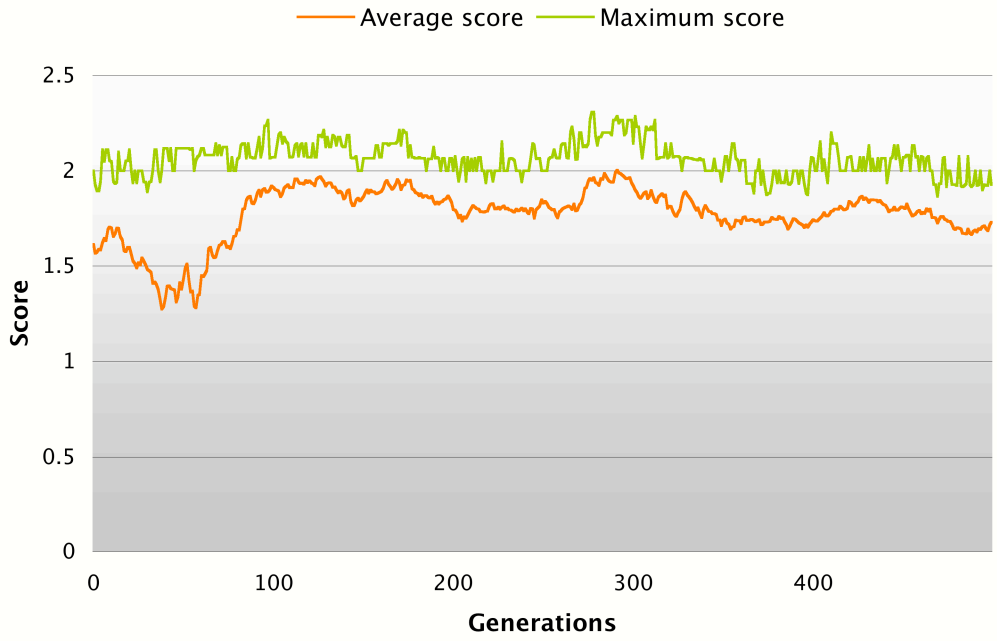
Score: **2.166667**
Appeared in generation: **263**

Nº of apartments: **12**
Nº of circulation blocks: **22**
Volume: **4480 m³**
Total area: **1120 m²**
Apartment area: **768 m²**
Circulation area: **352 m²**
Height: **44 m**
Footprint: **32 m x 28 m**

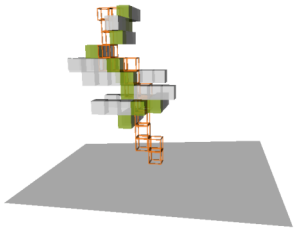
Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **41.666664 %**

Apartments with views to +i: **9**
Apartments with views to -i: **4**
Apartments with views to +j: **4**
Apartments with views to -j: **7**
Apartments with no views: **0**

Views and Balconies Population: **50** Set mutation rate: **0.01**



Actual mutation rate: **0.0031** Final average score: **1.7318** Time elapsed: **912 s**



Champ Nº1

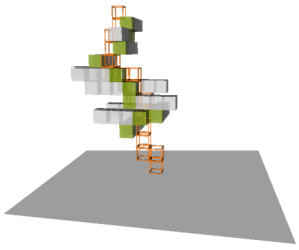
Shape code: **19 21 21 10 21 8 3 18 7 21 19 5 21 3 9 21 19 19 11 7 21 18 10 21 16 21 18 4 18 21 16 19**

Score: **2.307692**
Appeared in generation: **280**

Nº of apartments: **13**
Nº of circulation blocks: **21**
Volume: **4672m³**
Total area: **1168 m²**
Apartment area: **832 m²**
Circulation area: **336 m²**
Height: **48 m**
Footprint: **36 m × 32 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **61.538464 %**

Apartments with views to +i: **9**
Apartments with views to −i: **6**
Apartments with views to +j: **9**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ Nº2

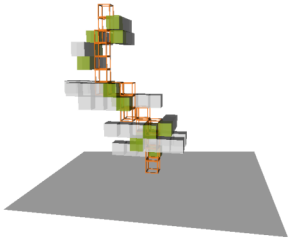
Shape code: **19 21 21 10 21 8 3 18 7 21 19 5 21 3 9 21 19 19 11 7 21 18 10 21 16 21 18 4 18 21 16 21**

Score: **2.307692**
Appeared in generation: **279**

Nº of apartments: **13**
Nº of circulation blocks: **21**
Volume: **4672 m³**
Total area: **1168 m²**
Apartment area: **832 m²**
Circulation area: **336 m²**
Height: **52 m**
Footprint: **36 m × 32 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **61.538464 %**

Apartments with views to +i: **9**
Apartments with views to −i: **6**
Apartments with views to +j: **9**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ Nº3

Shape code: **7 21 9 13 21 8 3 18 19 21 19 20 21 3 9 21 19 19 11 7 21 21 10 21 14 21 18 8 2 21 16 21**

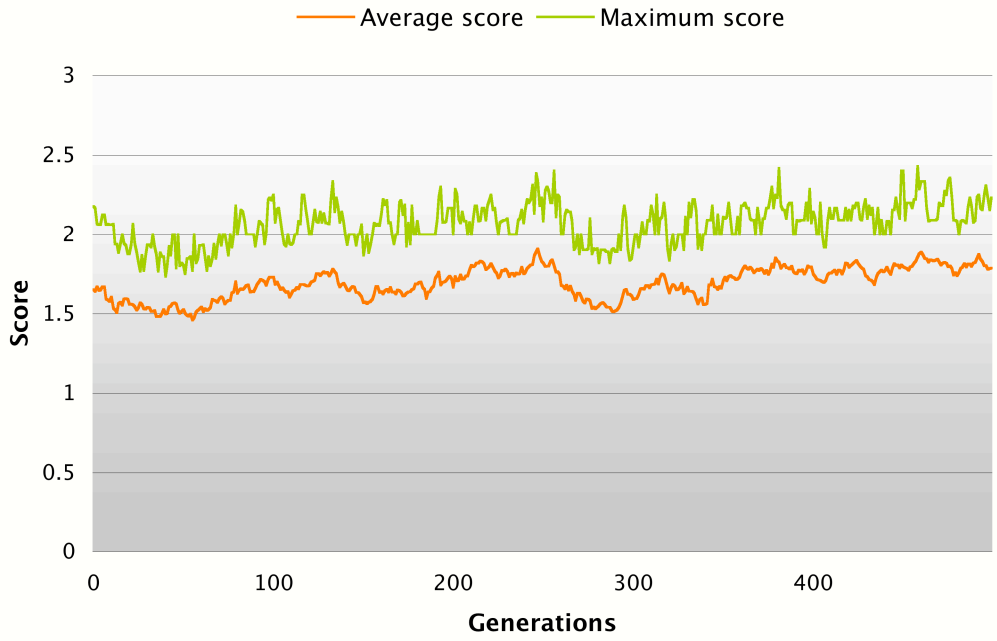
Score: **2.285714**
Appeared in generation: **303**

Nº of apartments: **14**
Nº of circulation blocks: **20**
Volume: **4864 m³**
Total area: **1216 m²**
Apartment area: **896 m²**
Circulation area: **320 m²**
Height: **52 m**
Footprint: **40 m × 28 m**

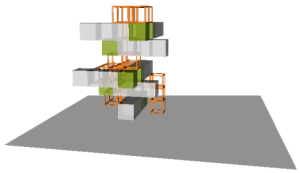
Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **57.14286 %**

Apartments with views to +i: **10**
Apartments with views to −i: **7**
Apartments with views to +j: **9**
Apartments with views to −j: **5**
Apartments with no views: **0**

Views and Balconies Population: **50** Set mutation rate: **0.05**



Actual mutation rate: **0.0137** Final average score: **1.7851** Time elapsed: **1163 s**



Champ Nº1

Shape code: **21 19 19 21 20 22 19 22 20 17 4 21 13 21 16 19 20 20 17 11 21 17 21 13 21 8 20 11 21 17 20 19**

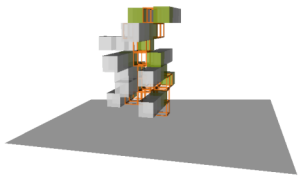
Score: **2.428571**

Appeared in generation: **460**

Nº of apartments: **7**
Nº of circulation blocks: **27**
Volume: **3520 m³**
Total area: **880 m²**
Apartment area: **448 m²**
Circulation area: **432 m²**
Height: **32 m**
Footprint: **24 m × 28 m**

Nº of apartments with balconies5
Percentage of apartments with balconies: **71.428574 %**

Apartments with views to +i: **5**
Apartments with views to −i: **2**
Apartments with views to +j: **4**
Apartments with views to −j: **2**
Apartments with no views: **0**



Champ Nº2

Shape code: **21 18 19 15 20 22 19 14 20 17 4 21 14 21 4 14 21 20 8 6 21 7 21 17 21 10 20 16 22 19 20 4**

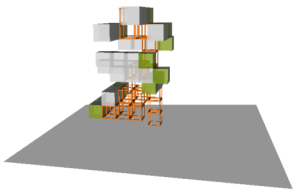
Score: **2.416667**

Appeared in generation: **383**

Nº of apartments: **12**
Nº of circulation blocks: **22**
Volume: **4480m³**
Total area: **1120 m²**
Apartment area: **768 m²**
Circulation area: **352 m²**
Height: **32 m**
Footprint: **24 m × 40 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **66.666672 %**

Apartments with views to +i: **9**
Apartments with views to −i: **6**
Apartments with views to +j: **3**
Apartments with views to −j: **6**
Apartments with no views: **0**



Champ Nº3

Shape code: **21 19 19 14 20 22 19 10 20 17 17 21 18 21 16 21 7 20 17 11 21 3 21 15 21 4 20 20 19 14 21 4**

Score: **2.4**

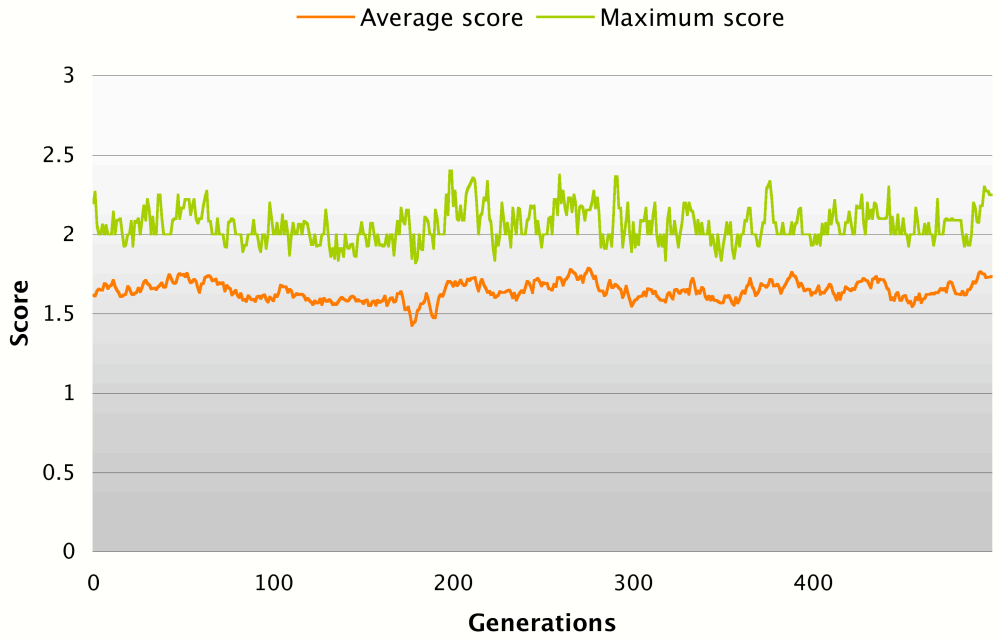
Appeared in generation: **451**

Nº of apartments: **10**
Nº of circulation blocks: **24**
Volume: **4096 m³**
Total area: **1024 m²**
Apartment area: **640 m²**
Circulation area: **384 m²**
Height: **36 m**
Footprint: **24 m × 36 m**

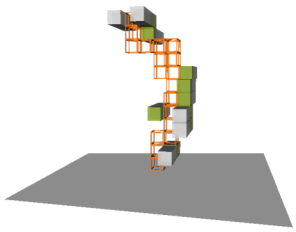
Nº of apartments with balconies: **6**
Percentage of apartments with balconies: **60.000004 %**

Apartments with views to +i: **8**
Apartments with views to −i: **4**
Apartments with views to +j: **5**
Apartments with views to −j: **7**
Apartments with no views: **0**

Views and Balconies Population: **50** Set mutation rate: **0.1**



Actual mutation rate: **0.024** Final average score: **1.732** Time elapsed: **1358 s**



Champ Nº1

Shape code: **12 21 17 21 12 21 12 10 21 8 18 21 16 21 8 19 21 21 17 22 20 21 19 19 19 21 21 18 4 14 22 8**

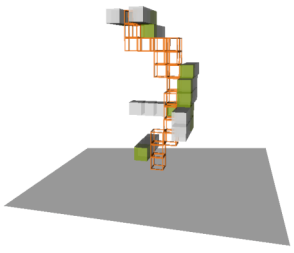
Score: **2.4**

Appeared in generation: **201**

Nº of apartments: **23**
Nº of circulation blocks: **27**
Volume: **7616 m³**
Total area: **1904 m²**
Apartment area: **1472 m²**
Circulation area: **432 m²**
Height: **64 m**
Footprint: **28 m × 24 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **50 %**

Apartments with views to +i: **9**
Apartments with views to −i: **5**
Apartments with views to +j: **5**
Apartments with views to −j: **5**
Apartments with no views: **0**



Champ Nº2

Shape code: **10 21 17 21 12 21 12 7 21 8 18 21 16 21 8 19 21 21 17 22 20 21 19 19 19 21 21 18 4 14 22 8**

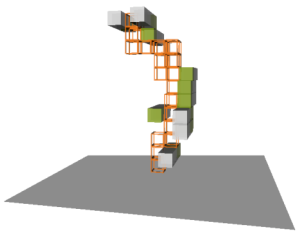
Score: **2.4**

Appeared in generation: **201**

Nº of apartments: **10**
Nº of circulation blocks: **24**
Volume: **4096 m³**
Total area: **1024 m²**
Apartment area: **640 m²**
Circulation area: **384 m²**
Height: **48 m**
Footprint: **24 m × 28 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **50 %**

Apartments with views to +i: **9**
Apartments with views to −i: **4**
Apartments with views to +j: **5**
Apartments with views to −j: **5**
Apartments with no views: **0**



Champ Nº3

Shape code: **12 21 17 21 12 21 12 10 21 8 18 21 16 21 8 19 21 21 17 22 20 21 19 19 19 21 21 18 4 14 22 8**

Score: **2.4**

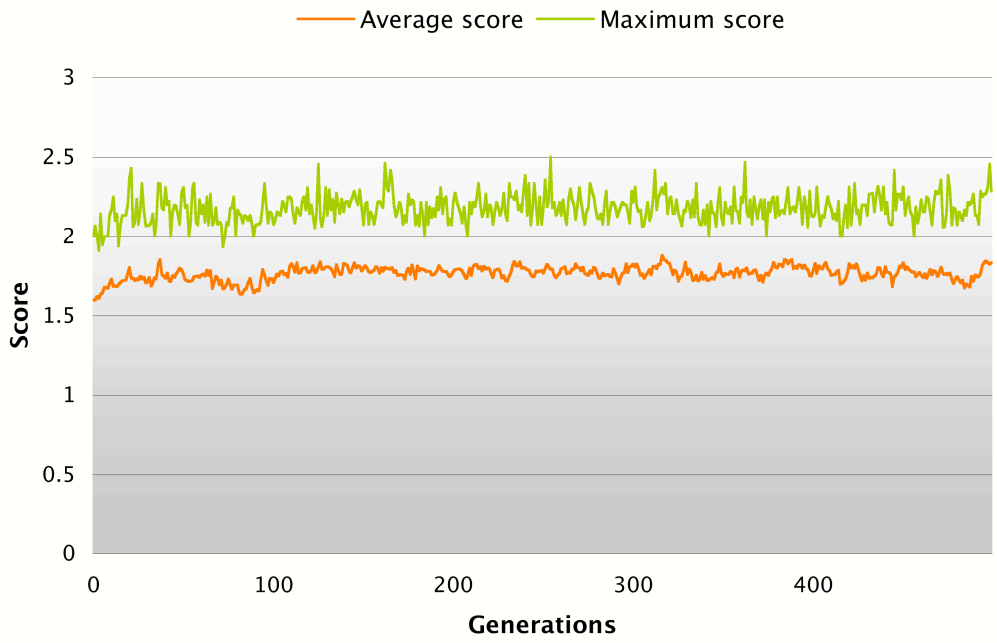
Appeared in generation: **200**

Nº of apartments: **10**
Nº of circulation blocks: **24**
Volume: **4096 m³**
Total area: **1024 m²**
Apartment area: **640 m²**
Circulation area: **384 m²**
Height: **48 m**
Footprint: **24 m × 28 m**

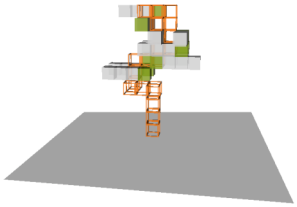
Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **50 %**

Apartments with views to +i: **9**
Apartments with views to −i: **5**
Apartments with views to +j: **5**
Apartments with views to −j: **5**
Apartments with no views: **0**

Views and Balconies Population: **50** Set mutation rate: **0.5**



Actual mutation rate: **0.1635** Final average score: **1.8322** Time elapsed: **2290 s**



Champ Nº1

Shape code: **21 21 18 19 19 18 18 21 11 15 21 17 17 9 5 21 17 11 15 21 4 14 21 21 19 7 19 20 17 22 10 18**

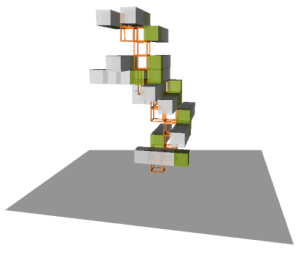
Score: **2.5**

Appeared in generation: **256**

Nº of apartments: **10**
Nº of circulation blocks: **24**
Volume: **4096 m³**
Total area: **1024 m²**
Apartment area: **640 m²**
Circulation area: **384 m²**
Height: **40 m**
Footprint: **32 m × 20 m**

Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **70 %**

Apartments with views to +i: **8**
Apartments with views to −i: **4**
Apartments with views to +j: **4**
Apartments with views to −j: **6**
Apartments with no views: **0**



Champ Nº2

Shape code: **17 11 21 2 21 4 21 8 19 21 12 18 15 21 15 19 20 4 21 20 3 8 21 8 6 21 21 16 19 21 14 4**

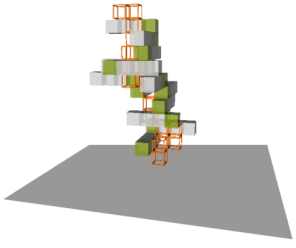
Score: **2.466667**

Appeared in generation: **364**

Nº of apartments: **15**
Nº of circulation blocks: **19**
Volume: **5056 m³**
Total area: **1264 m²**
Apartment area: **960 m²**
Circulation area: **304 m²**
Height: **48 m**
Footprint: **28 m × 32 m**

Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **60.000004 %**

Apartments with views to +i: **13**
Apartments with views to −i: **4**
Apartments with views to +j: **8**
Apartments with views to −j: **7**
Apartments with no views: **0**



Champ Nº3

Shape code: **10 18 17 21 19 9 21 11 19 21 16 21 2 8 21 20 20 12 19 7 21 13 15 21 8 18 21 12 21 15 21 19**

Score: **2.461539**

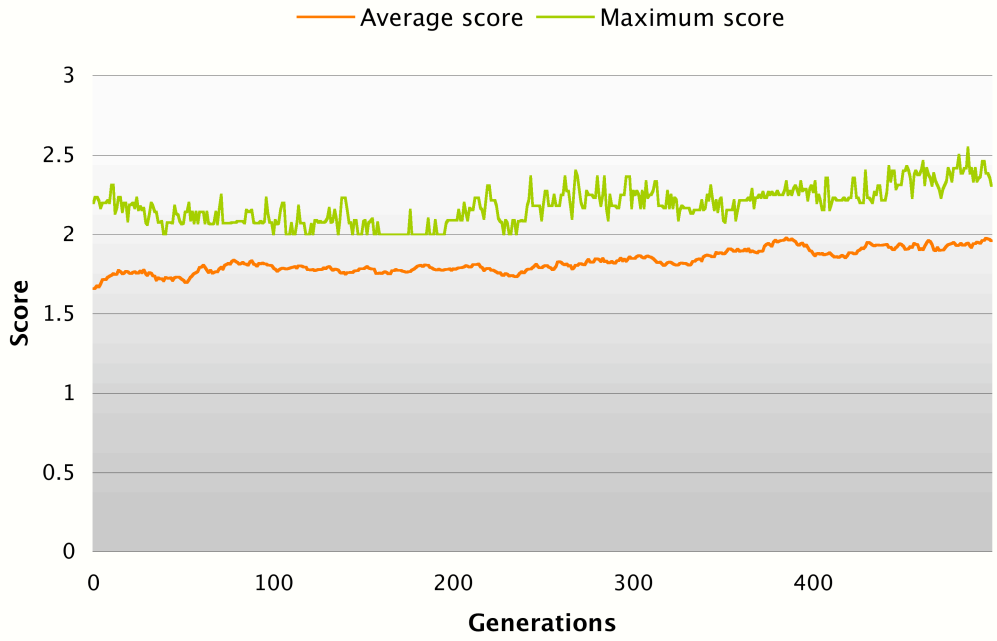
Appeared in generation: **164**

Nº of apartments: **13**
Nº of circulation blocks: **21**
Volume: **4672 m³**
Total area: **1168 m²**
Apartment area: **832 m²**
Circulation area: **336 m²**
Height: **48 m**
Footprint: **32 m × 32 m**

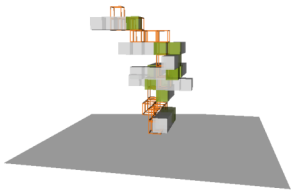
Nº of apartments with balconies: **10**
Percentage of apartments with balconies: **76.92308 %**

Apartments with views to +i: **9**
Apartments with views to −i: **6**
Apartments with views to +j: **9**
Apartments with views to −j: **7**
Apartments with no views: **0**

Views and Balconies Population: **200** Set mutation rate: **0.005**



Actual mutation rate: **0.0012** Final average score: **1.9566** Time elapsed: **4109 s**



Champ Nº1

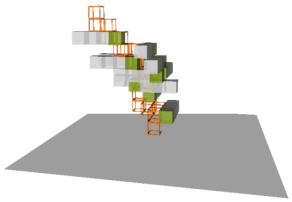
Shape code: **4 18 18 19 18 18 21 16 21 5 15 17 21 8 2 20 11 21 4 21 8 3 21 19 19 20 21 19 11 18 19 21**

Score: **2.545455**
Appeared in generation: **488**

Nº of apartments: **11**
Nº of circulation blocks: **23**
Volume: **4288 m³**
Total area: **1072 m²**
Apartment area: **704 m²**
Circulation area: **368 m²**
Height: **40 m**
Footprint: **32 m × 44 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **72.727272 %**

Apartments with views to +i: **9**
Apartments with views to −i: **3**
Apartments with views to +j: **6**
Apartments with views to −j: **3**
Apartments with no views: **0**



Champ Nº2

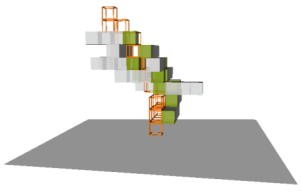
Shape code: **8 18 18 19 18 19 21 8 21 5 15 17 21 8 2 20 11 21 4 19 7 3 21 12 19 20 21 19 11 21 19 21**

Score: **2.5**
Appeared in generation: **483**

Nº of apartments: **12**
Nº of circulation blocks: **22**
Volume: **4480 m³**
Total area: **1120 m²**
Apartment area: **768 m²**
Circulation area: **352 m²**
Height: **40 m**
Footprint: **32 m × 32 m**

Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **75 %**

Apartments with views to +i: **9**
Apartments with views to −i: **3**
Apartments with views to +j: **6**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ Nº3

Shape code: **8 21 18 22 18 18 21 16 21 5 15 19 21 8 2 20 11 21 4 19 7 3 21 16 6 20 21 19 11 21 19 21**

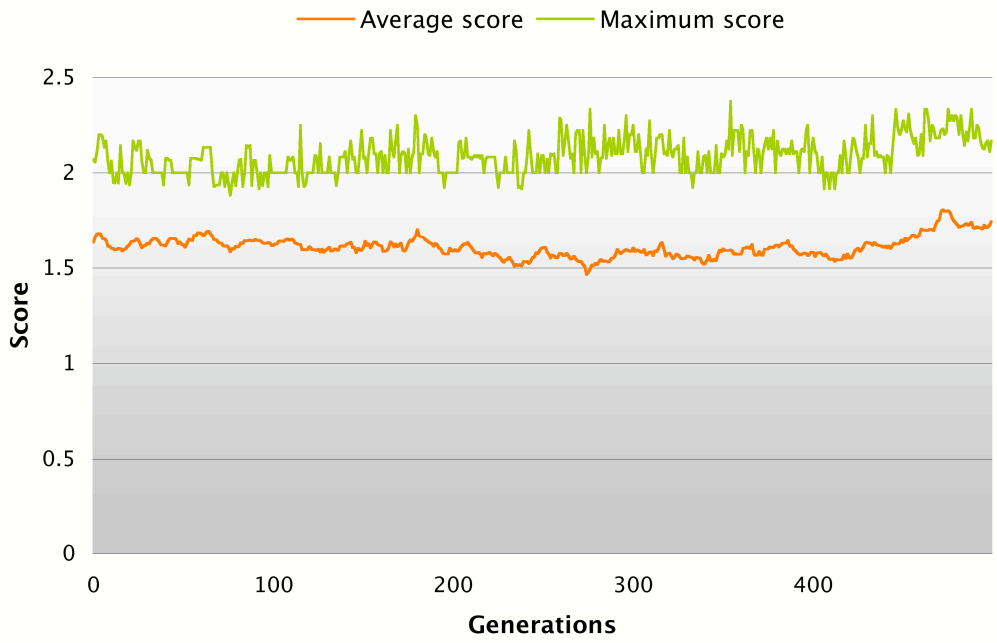
Score: **2.461539**
Appeared in generation: **497**

Nº of apartments: **13**
Nº of circulation blocks: **21**
Volume: **4672 m³**
Total area: **1168 m²**
Apartment area: **832 m²**
Circulation area: **336 m²**
Height: **40 m**
Footprint: **36 m × 32 m**

Nº of apartments with balconies: **10**
Percentage of apartments with balconies: **76.92308 %**

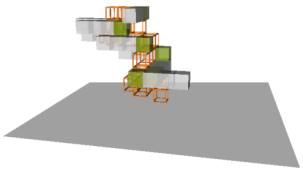
Apartments with views to +i: **9**
Apartments with views to −i: **4**
Apartments with views to +j: **7**
Apartments with views to −j: **4**
Apartments with no views: **0**

Views and Balconies Population: **200** Set mutation rate: **0.01**



Actual mutation rate: **0.0025** Final average score: **1.7445** Time elapsed: **4149 s**

Champ Nº1



Shape code: **18 19 19 20 20 17 20 21 8 5 21 18 18 15 21 3 18 19 21 19 21 12 18 19 20 22 11 7 22 18 7 19**

Score: **2.375**

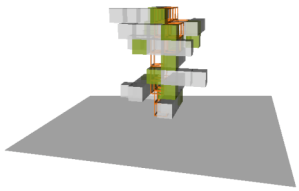
Appeared in generation: **356**

Nº of apartments: **8**
Nº of circulation blocks: **26**
Volume: **3712 m³**
Total area: **928 m²**
Apartment area: **512 m²**
Circulation area: **416 m²**
Height: **28 m**
Footprint: **40 m × 28 m**

Nº of apartments with balconies: **4**
Percentage of apartments with balconies: **50 %**

Apartments with views to +i: **7**
Apartments with views to −i: **1**
Apartments with views to +j: **4**
Apartments with views to −j: **4**
Apartments with no views: **0**

Champ Nº2



Shape code: **7 4 21 20 14 16 21 5 21 16 21 13 17 7 21 3 18 19 9 21 3 18 18 18 10 22 17 16 22 11 7 19**

Score: **2.333333**

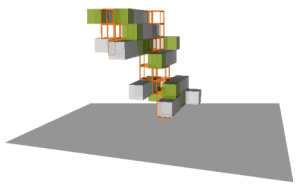
Appeared in generation: **490**

Nº of apartments: **15**
Nº of circulation blocks: **19**
Volume: **5056 m³**
Total area: **1264 m²**
Apartment area: **960 m²**
Circulation area: **304 m²**
Height: **32 m**
Footprint: **28 m × 36 m**

Nº of apartments with balconies: **10**
Percentage of apartments with balconies: **66.666672 %**

Apartments with views to +i: **10**
Apartments with views to −i: **7**
Apartments with views to +j: **6**
Apartments with views to −j: **6**
Apartments with no views: **0**

Champ Nº3



Shape code: **9 4 21 20 14 16 21 19 21 16 21 18 18 15 21 10 21 19 19 20 20 20 16 10 20 22 17 16 22 4 15 20**

Score: **2.333333**

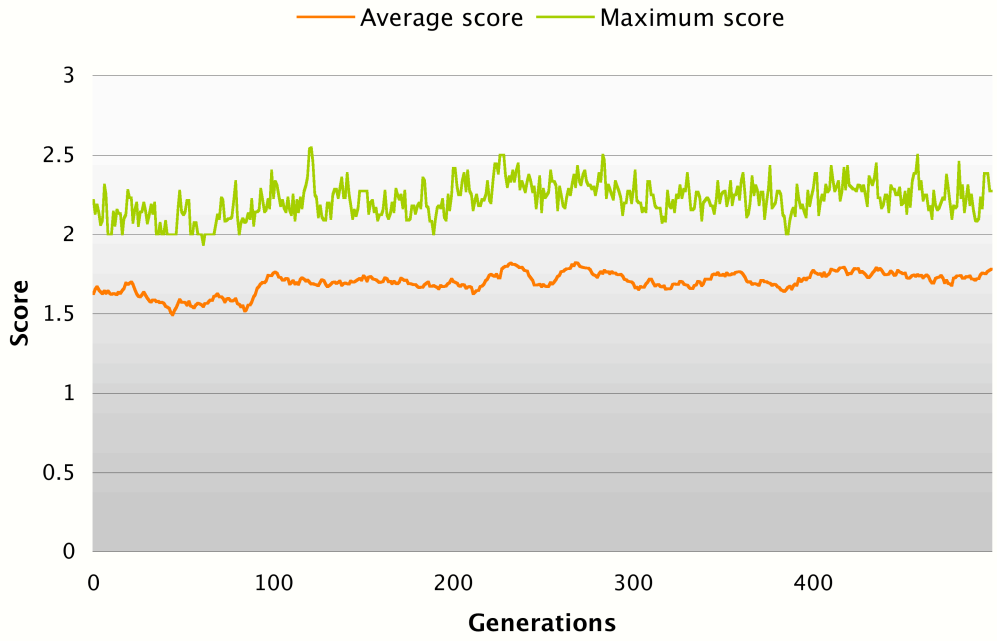
Appeared in generation: **477**

Nº of apartments: **12**
Nº of circulation blocks: **22**
Volume: **4480 m³**
Total area: **1120 m²**
Apartment area: **768 m²**
Circulation area: **352 m²**
Height: **32 m**
Footprint: **28 m × 40 m**

Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **58.333332 %**

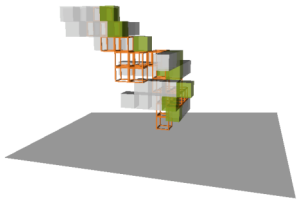
Apartments with views to +i: **9**
Apartments with views to −i: **6**
Apartments with views to +j: **7**
Apartments with views to −j: **3**
Apartments with no views: **0**

Views and Balconies Population: **200** Set mutation rate: **0.05**



Actual mutation rate: **0.0148** Final average score: **1.7789** Time elapsed: **4655 s**

Champ Nº1



Shape code: **16 21 18 17 20 20 3 21 11 19 21 7 8 21 4 20 19 20 22 19 21 21 15 19 21 11 19 18 8 21 7 11**

Score: **2.545455**

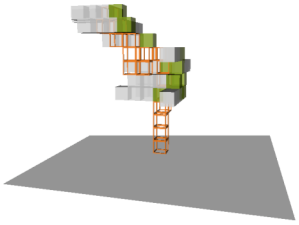
Appeared in generation: **123**

Nº of apartments: **16**
Nº of circulation blocks: **34**
Volume: **6272 m³**
Total area: **1568 m²**
Apartment area: **1024 m²**
Circulation area: **544 m²**
Height: **64 m**
Footprint: **28 m x 24 m**

Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **63.636364 %**

Apartments with views to +i: **10**
Apartments with views to -i: **1**
Apartments with views to +j: **7**
Apartments with views to -j: **7**
Apartments with no views: **0**

Champ Nº2



Shape code: **21 21 18 21 15 4 3 21 11 15 21 7 8 21 4 20 19 20 22 19 21 21 15 19 21 11 19 18 8 21 7 11**

Score: **2.538461**

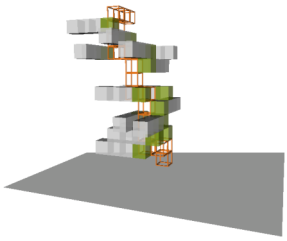
Appeared in generation: **122**

Nº of apartments: **13**
Nº of circulation blocks: **21**
Volume: **4672 m³**
Total area: **1168 m²**
Apartment area: **832 m²**
Circulation area: **336 m²**
Height: **44 m**
Footprint: **36 m x 28 m**

Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **69.230774 %**

Apartments with views to +i: **11**
Apartments with views to -i: **0**
Apartments with views to +j: **6**
Apartments with views to -j: **8**
Apartments with no views: **0**

Champ Nº3



Shape code: **20 19 3 21 11 15 21 14 4 21 8 21 15 3 19 21 21 8 21 4 19 18 1 19 6 21 4 21 3 21 18 18**

Score: **2.5**

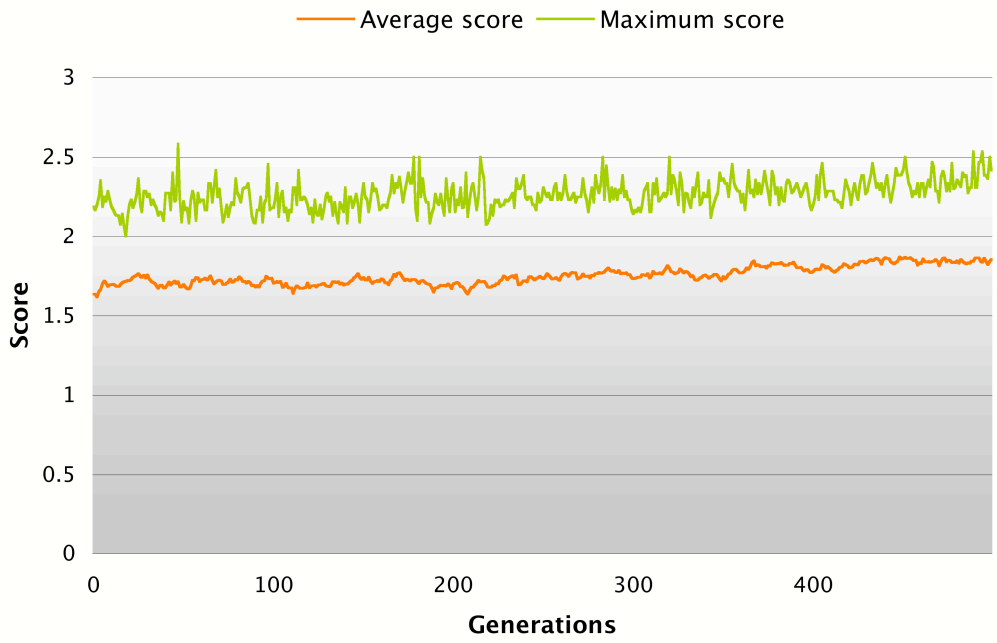
Appeared in generation: **460**

Nº of apartments: **14**
Nº of circulation blocks: **20**
Volume: **4864 m³**
Total area: **1216 m²**
Apartment area: **896 m²**
Circulation area: **320 m²**
Height: **48 m**
Footprint: **32 m x 28 m**

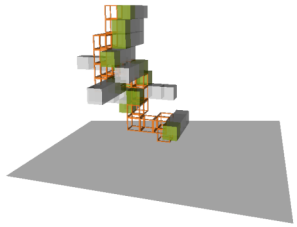
Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **64.285713 %**

Apartments with views to +i: **12**
Apartments with views to -i: **3**
Apartments with views to +j: **7**
Apartments with views to -j: **7**
Apartments with no views: **0**

Views and Balconies Population: **200** Set mutation rate: **0.1**



Actual mutation rate: **0.0315** Final average score: **1.8511** Time elapsed: **4813 s**



Champ Nº1

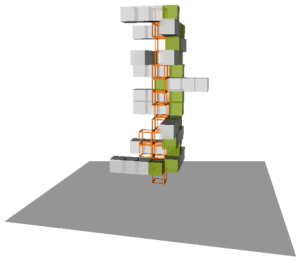
Shape code: **20 16 19 19 21 18 2 21 3 9 21 19 15 21 4 19 7 20 22 20 4 21 21 16 21 18 8 18 21 16 21 8**

Score: **2.583333**
Appeared in generation: **49**

Nº of apartments: **12**
Nº of circulation blocks: **22**
Volume: **4480 m³**
Total area: **1120 m²**
Apartment area: **768 m²**
Circulation area: **352 m²**
Height: **40 m**
Footprint: **36 m × 36 m**

Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **75 %**

Apartments with views to +i: **10**
Apartments with views to −i: **5**
Apartments with views to +j: **5**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ Nº2

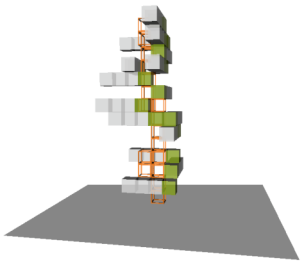
Shape code: **3 8 21 15 19 21 17 16 21 12 18 21 15 21 15 20 21 1 21 18 15 21 6 4 21 16 21 4 15 20 21 11**

Score: **2.533333**
Appeared in generation: **496**

Nº of apartments: **15**
Nº of circulation blocks: **19**
Volume: **5056 m³**
Total area: **1264 m²**
Apartment area: **960 m²**
Circulation area: **304 m²**
Height: **52 m**
Footprint: **28 m × 24 m**

Nº of apartments with balconies: **10**
Percentage of apartments with balconies: **66.666672 %**

Apartments with views to +i: **13**
Apartments with views to −i: **4**
Apartments with views to +j: **9**
Apartments with views to −j: **10**
Apartments with no views: **0**



Champ Nº3

Shape code: **11 21 8 19 21 17 15 21 21 12 21 16 3 21 19 3 18 21 16 21 3 21 14 4 21 16 21 4 15 20 21 4**

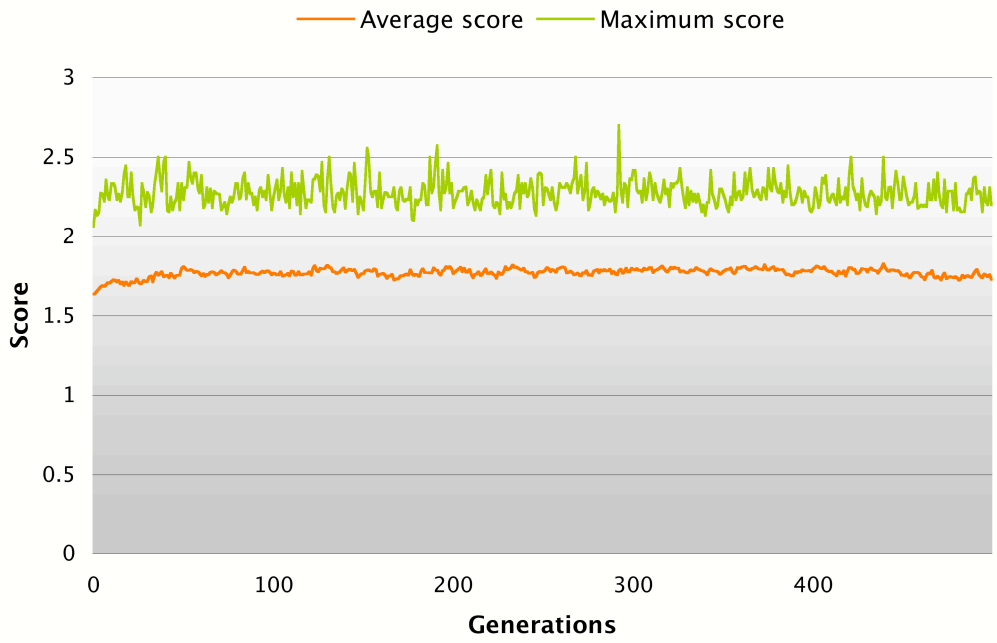
Score: **2.533333**
Appeared in generation: **491**

Nº of apartments: **15**
Nº of circulation blocks: **19**
Volume: **5056 m³**
Total area: **1264 m²**
Apartment area: **960 m²**
Circulation area: **304 m²**
Height: **56 m**
Footprint: **24 m × 28 m**

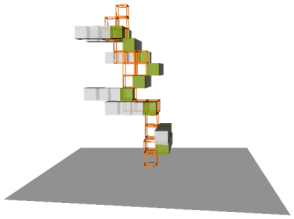
Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **60.000004 %**

Apartments with views to +i: **14**
Apartments with views to −i: **4**
Apartments with views to +j: **9**
Apartments with views to −j: **7**
Apartments with no views: **0**

Views and Balconies Population: **200** Set mutation rate: **0.5**



Actual mutation rate: **0.1649** Final average score: **1.7274** Time elapsed: **7107 s**



Champ Nº1

Shape code: **8 18 21 12 21 21 3 18 19 19 21 3 21 16 21 17 8 21 19 18 15 19 21 16 21 3 19 18 15 21 17 21**

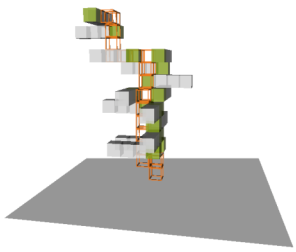
Score: **2.7**

Appeared in generation: **294**

Nº of apartments: **10**
Nº of circulation blocks: **24**
Volume: **4096 m³**
Total area: **1024 m²**
Apartment area: **640 m²**
Circulation area: **384 m²**
Height: **52 m**
Footprint: **32 m x 28 m**

Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **70 %**

Apartments with views to +i: **10**
Apartments with views to -i: **3**
Apartments with views to +j: **7**
Apartments with views to -j: **3**
Apartments with no views: **0**



Champ Nº2

Shape code: **19 21 15 11 21 4 21 16 21 7 4 21 18 8 6 21 1 21 18 15 21 16 19 3 19 19 21 21 7 21 20 10**

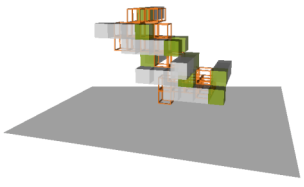
Score: **2.571429**

Appeared in generation: **193**

Nº of apartments: **14**
Nº of circulation blocks: **20**
Volume: **4864 m³**
Total area: **1216 m²**
Apartment area: **896 m²**
Circulation area: **320 m²**
Height: **52 m**
Footprint: **40 m x 32 m**

Nº of apartments with balconies: **11**
Percentage of apartments with balconies: **78.571426 %**

Apartments with views to +i: **11**
Apartments with views to -i: **4**
Apartments with views to +j: **7**
Apartments with views to -j: **8**
Apartments with no views: **0**



Champ Nº3

Shape code: **17 17 18 17 20 20 11 21 8 19 3 19 21 4 19 2 21 11 15 21 19 22 19 19 21 17 11 21 18 17 18 2**

Score: **2.555556**

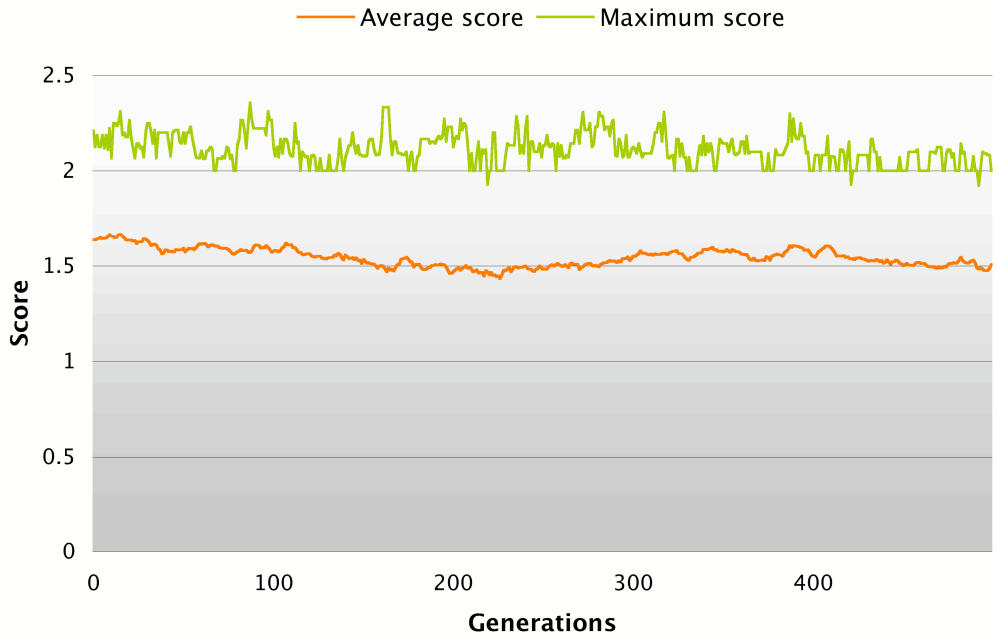
Appeared in generation: **154**

Nº of apartments: **9**
Nº of circulation blocks: **25**
Volume: **3904 m³**
Total area: **976 m²**
Apartment area: **576 m²**
Circulation area: **400 m²**
Height: **28 m**
Footprint: **36 m x 36 m**

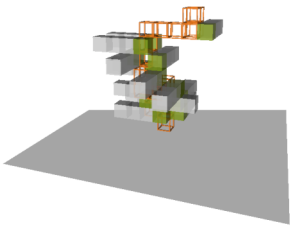
Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **77.777779 %**

Apartments with views to +i: **7**
Apartments with views to -i: **2**
Apartments with views to +j: **6**
Apartments with views to -j: **3**
Apartments with no views: **0**

Views and Balconies Population: **500** Set mutation rate: **0.005**



Actual mutation rate: **0.0009** Final average score: **1.506** Time elapsed: **10633 s**



Champ Nº1

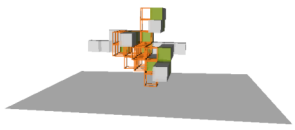
Shape code: **8 7 3 19 21 7 5 17 8 21 20 20 12 19 21 4 18 7 21 16 6 21 3 7 21 17 17 17 21 18 22 16**

Score: **2.357143**
Appeared in generation: **89**

Nº of apartments: **14**
Nº of circulation blocks: **20**
Volume: **4864 m³**
Total area: **1216 m²**
Apartment area: **896 m²**
Circulation area: **320 m²**
Height: **36 m**
Footprint: **32 m × 36 m**

Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **64.285713 %**

Apartments with views to +i: **10**
Apartments with views to −i: **2**
Apartments with views to +j: **6**
Apartments with views to −j: **7**
Apartments with no views: **0**



Champ Nº2

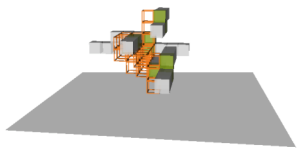
Shape code: **4 21 8 19 18 22 8 18 19 21 17 18 22 18 21 1 19 19 7 20 20 21 18 4 18 17 17 7 21 12 21 16**

Score: **2.333333**
Appeared in generation: **164**

Nº of apartments: **9**
Nº of circulation blocks: **25**
Volume: **3904 m³**
Total area: **976 m²**
Apartment area: **576 m²**
Circulation area: **400 m²**
Height: **24 m**
Footprint: **36 m × 40 m**

Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **55.555557 %**

Apartments with views to +i: **7**
Apartments with views to −i: **2**
Apartments with views to +j: **2**
Apartments with views to −j: **5**
Apartments with no views: **0**



Champ Nº3

Shape code: **4 21 8 19 18 22 8 18 19 21 17 18 22 18 21 1 19 19 7 20 20 21 18 4 18 17 17 7 21 12 21 8**

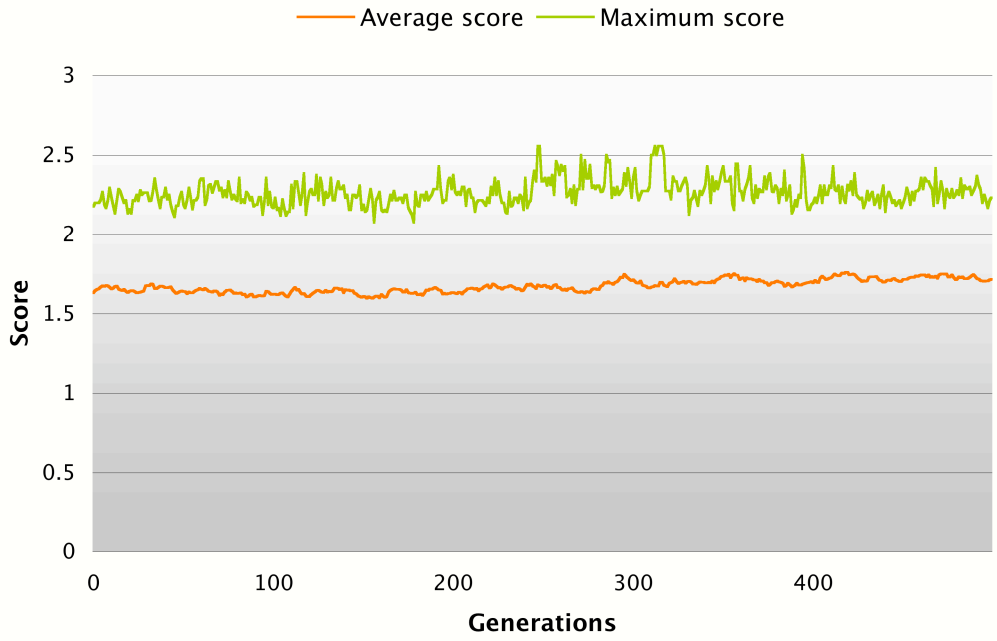
Score: **2.333333**
Appeared in generation: **163**

Nº of apartments: **9**
Nº of circulation blocks: **25**
Volume: **3904 m³**
Total area: **976 m²**
Apartment area: **576 m²**
Circulation area: **400 m²**
Height: **24 m**
Footprint: **36 m × 44 m**

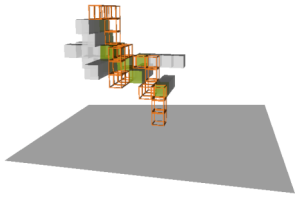
Nº of apartments with balconies: **5**
Percentage of apartments with balconies: **55.555557 %**

Apartments with views to +i: **7**
Apartments with views to −i: **1**
Apartments with views to +j: **2**
Apartments with views to −j: **5**
Apartments with no views: **0**

Views and Balconies Population: **500** Set mutation rate: **0.01**



Actual mutation rate: **0.0025** Final average score: **1.7149** Time elapsed: **10137 s**



Champ Nº1

Shape code: **21 19 18 18 16 21 21 9 19 19 3 22 16 20 20 20 21 14 21 18 18 19 3 7 21 14 21 6 21 20 20 20**

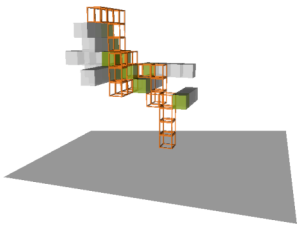
Score: **2.555556**

Appeared in generation: **317**

Nº of apartments: **9**
Nº of circulation blocks: **25**
Volume: **3904 m³**
Total area: **976 m²**
Apartment area: **576 m²**
Circulation area: **400 m²**
Height: **36 m**
Footprint: **36 m x 32 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **88.888893 %**

Apartments with views to +i: **6**
Apartments with views to -i: **4**
Apartments with views to +j: **2**
Apartments with views to -j: **4**
Apartments with no views: **0**



Champ Nº2

Shape code: **21 21 8 19 18 21 21 9 19 19 3 22 16 20 20 20 21 14 21 18 18 19 3 7 21 14 21 6 21 20 20 20**

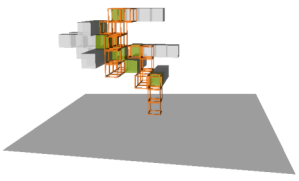
Score: **2.555556**

Appeared in generation: **316**

Nº of apartments: **9**
Nº of circulation blocks: **25**
Volume: **3904 m³**
Total area: **976 m²**
Apartment area: **576 m²**
Circulation area: **400 m²**
Height: **40 m**
Footprint: **36 m x 32 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **88.888893 %**

Apartments with views to +i: **6**
Apartments with views to -i: **4**
Apartments with views to +j3
Apartments with views to -j: **4**
Apartments with no views: **0**



Champ Nº3

Shape code: **21 19 18 18 16 21 21 9 19 19 3 22 16 20 20 20 21 14 21 18 18 19 3 7 21 14 17 20 21 17 9 20**

Score: **2.555556**

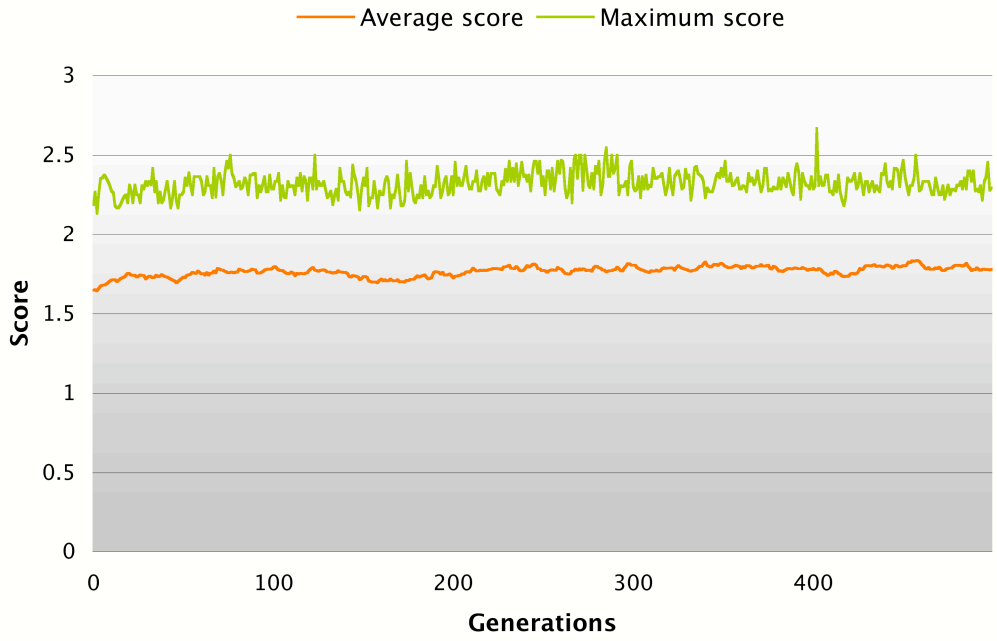
Appeared in generation: **314**

Nº of apartments: **9**
Nº of circulation blocks: **25**
Volume: **3904m³**
Total area: **976 m²**
Apartment area: **576 m²**
Circulation area: **400 m²**
Height: **32 m**
Footprint: **36 m x 32 m**

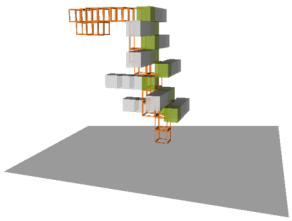
Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **88.888893 %**

Apartments with views to +i: **6**
Apartments with views to -i: **4**
Apartments with views to +j: **2**
Apartments with views to -j: **4**
Apartments with no views: **0**

Views and Balconies Population: **500** Set mutation rate: **0.05**



Actual mutation rate: **0.0162** Final average score: **1.778** Time elapsed: **10137 s**



Champ Nº1

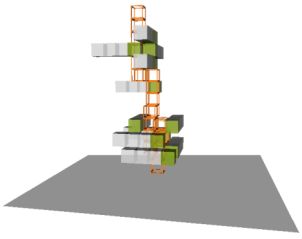
Shape code: **21 16 7 19 21 12 18 21 3 21 16 19 21 20 4 21 8 21 4 21 16 19 20 19 19 22 18 21 19 19 18 22**

Score: **2.666667**
Appeared in generation: **404**

Nº of apartments: **9**
Nº of circulation blocks: **25**
Volume: **3904 m³**
Total area: **976 m²**
Apartment area: **576 m²**
Circulation area: **400 m²**
Height: **40 m**
Footprint: **36 m x 28 m**

Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **77.777779 %**

Apartments with views to +i: **8**
Apartments with views to −i: **2**
Apartments with views to +j: **5**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ Nº2

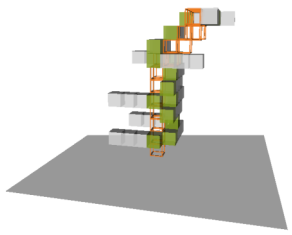
Shape code: **11 18 18 16 19 21 17 7 20 3 21 8 18 19 10 21 18 21 21 7 21 19 20 21 4 21 8 3 21 4 21 21**

Score: **2.545455**
Appeared in generation: **287**

Nº of apartments: **11**
Nº of circulation blocks: **23**
Volume: **4288 m³**
Total area: **1072 m²**
Apartment area: **704 m²**
Circulation area: **368 m²**
Height: 52 **m**
Footprint: **28 m x 28 m**

Nº of apartments with balconies: **7**
Percentage of apartments with balconies: **63.636364 %**

Apartments with views to +i: **10**
Apartments with views to −i: **1**
Apartments with views to +j: **6**
Apartments with views to −j: **4**
Apartments with no views: **0**



Champ Nº3

Shape code: **8 3 21 15 21 16 21 8 3 21 16 21 8 18 21 17 7 20 13 21 10 17 21 2 17 21 4 9 20 20 20 22**

Score: **2.5**
Appeared in generation: **459**

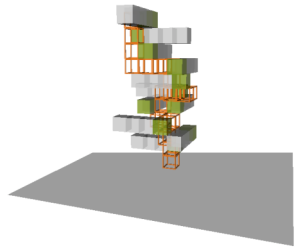
Nº of apartments: **14**
Nº of circulation blocks: **20**
Volume: **4864 m³**
Total area: **1216 m²**
Apartment area: **896 m²**
Circulation area: **320 m²**
Height: **44 m**
Footprint: **36 m x 28 m**

Nº of apartments with balconies: **10**
Percentage of apartments with balconies: **71.428574 %**

Apartments with views to +i: **11**
Apartments with views to −i: **6**
Apartments with views to +j: **12**
Apartments with views to −j: **3**
Apartments with no views: **0**

Views and Balconies Population: **500** Set mutation rate: **0.1**

Actual mutation rate: **0.0333** Final average score: **1.7872** Time elapsed: **10482 s**



Champ Nº1

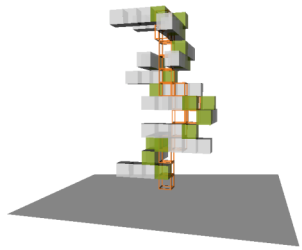
Shape code: **7 21 12 20 21 3 18 19 18 21 8 10 21 17 7 17 20 19 21 11 21 20 20 8 19 19 21 8 21 1 21 4**

Score: **2.636364**
Appeared in generation: **299**

Nº of apartments: **11**
Nº of circulation blocks: **23**
Volume: **4288m³**
Total area: **1072 m²**
Apartment area: **704 m²**
Circulation area: **368 m²**
Height: **44 m**
Footprint: **20 m × 40 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **72.727272 %**

Apartments with views to +i: **10**
Apartments with views to -i: **3**
Apartments with views to +j: **6**
Apartments with views to -j: **6**
Apartments with no views: **0**



Champ Nº2

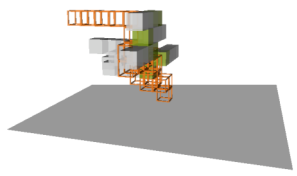
Shape code: **3 21 10 21 5 2 17 21 11 21 17 16 21 15 3 19 21 19 9 21 7 21 12 14 21 8 21 20 4 21 3 15**

Score: **2.625**
Appeared in generation: **429**

Nº of apartments: **16**
Nº of circulation blocks: **18**
Volume: **5248 m³**
Total area: **1312 m²**
Apartment area: **1024 m²**
Circulation area: **288 m²**
Height: **52 m**
Footprint: **28 m × 32 m**

Nº of apartments with balconies: **14**
Percentage of apartments with balconies: **87.5 %**

Apartments with views to +i: **12**
Apartments with views to -i: **6**
Apartments with views to +j: **12**
Apartments with views to -j:8
Apartments with no views: **0**



Champ Nº3

Shape code: **20 19 18 21 3 19 7 21 14 17 16 20 20 20 22 17 21 20 19 20 21 18 7 4 21 16 21 12 19 19 19 19**

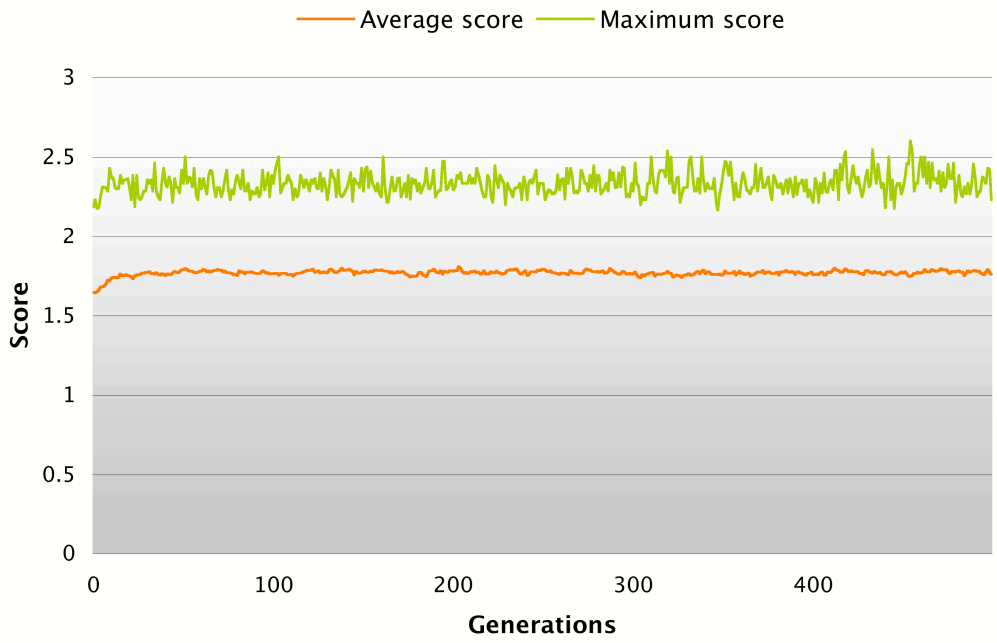
Score: **2.625**
Appeared in generation: **90**

Nº of apartments: **8**
Nº of circulation blocks: **26**
Volume: **3712 m³**
Total area: **928 m²**
Apartment area: **512 m²**
Circulation area: **416 m²**
Height: **28 m**
Footprint: **24 m × 40 m**

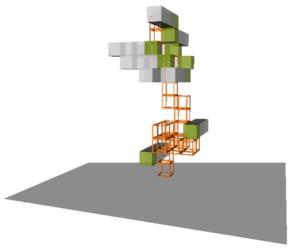
Nº of apartments with balconies: **6**
Percentage of apartments with balconies: **75 %**

Apartments with views to +i: **7**
Apartments with views to -i: **3**
Apartments with views to +j: **1**
Apartments with views to -j: **5**
Apartments with no views: **0**

Views and Balconies Population: **500** Set mutation rate: **0.5**



Actual mutation rate: **0.1654** Final average score: **1.7597** Time elapsed: **11790 s**



Champ Nº1

Shape code: **10 21 18 21 20 20 22 17 17 18 19 18 21 16 21 21 19 21 21 4 14 21 15 19 11 21 3 15 21 8 21 4**

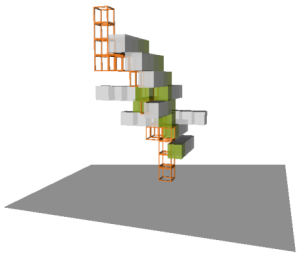
Score: **2.6**

Appeared in generation: **456**

Nº of apartments: **10**
Nº of circulation blocks: **24**
Volume: **4096 m³**
Total area: **1024 m²**
Apartment area: **640 m²**
Circulation area: **384 m²**
Height: **48 m**
Footprint: **28 m × 28 m**

Nº of apartments with balconies: **6**
Percentage of apartments with balconies: **60.000004 %**

Apartments with views to +i: **10**
Apartments with views to −i: **2**
Apartments with views to +j: **7**
Apartments with views to −j: **5**
Apartments with no views: **0**



Champ Nº2

Shape code: **21 16 21 19 18 8 21 1 11 21 10 20 20 12 21 8 3 21 4 20 20 21 4 19 21 20 12 22 19 21 21 21**

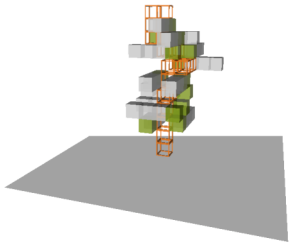
Score: **2.545455**

Appeared in generation: **435**

Nº of apartments: **11**
Nº of circulation blocks: **23**
Volume: **4288 m³**
Total area: **1072 m²**
Apartment area: **704 m²**
Circulation area: **368 m²**
Height: **48 m**
Footprint: **28 m × 44 m**

Nº of apartments with balconies: **8**
Percentage of apartments with balconies: **72.727272 %**

Apartments with views to +i: **9**
Apartments with views to −i: **3**
Apartments with views to +j: **5**
Apartments with views to −j: **5**
Apartments with no views: **0**



Champ Nº3

Shape code: **21 10 8 21 11 15 21 18 15 21 4 14 21 17 18 18 9 19 21 18 16 19 20 11 21 9 4 19 21 12 21 17**

Score: **2.538461**

Appeared in generation: **457**

Nº of apartments: **13**
Nº of circulation blocks: **21**
Volume: **4672 m³**
Total area: **1168 m²**
Apartment area: **832 m²**
Circulation area: **336 m²**
Height: **44 m**
Footprint: **28 m × 36 m**

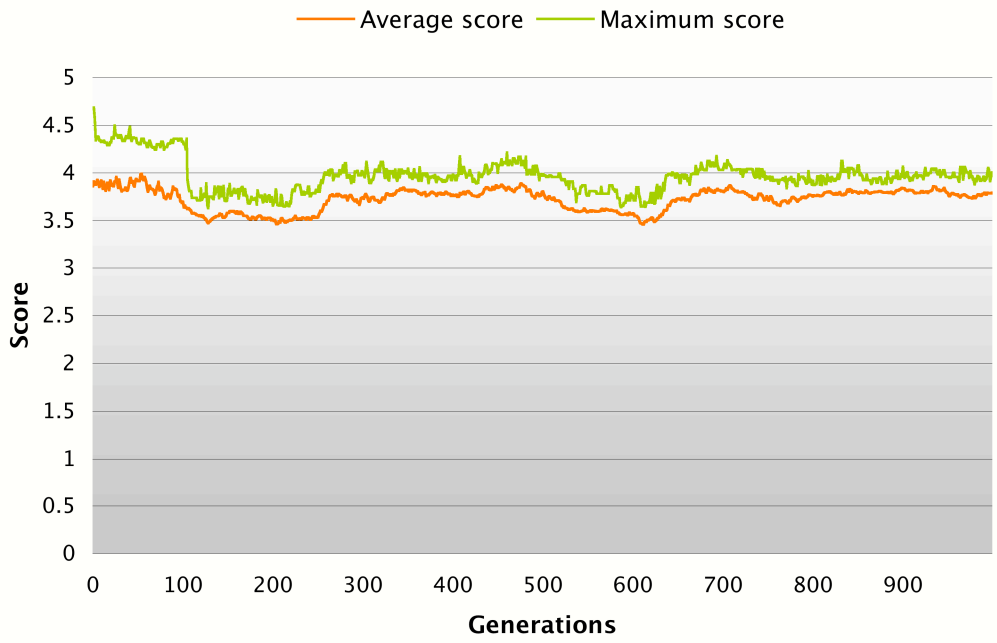
Nº of apartments with balconies: **9**
Percentage of apartments with balconies: **69.230774 %**

Apartments with views to +i: **11**
Apartments with views to −i: **5**
Apartments with views to +j: **6**
Apartments with views to −j: **7**
Apartments with no views: **0**

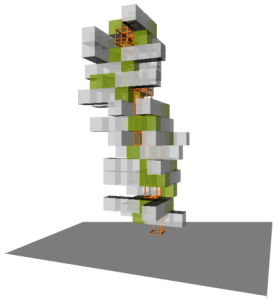
Appendix D
Results for Multiple Criteria Scenario

In this case, design goals from the previous test cases are combined. The building should be as high as possible, the percentage of apartments should be as high as possible, and as many apartments as possible should have balconies. The goal values for these three criteria are therefore set to their maximum values, at 64 metres, 80%, and 100% respectively. The goal value for apartments with views in the positive *i* direction is set to 80% (allowing, perhaps, the 20% of apartments that don't have views in that direction to be sold or rented at a lower price). The weights for the criteria mentioned so far have been set to 1. A less important goal is to give the building an oblong footprint, measuring 24 $\times$ 64 metres in the *i* and *j* directions respectively. The weights for these two criteria have been set to 0.6. This set of criteria seems likely to be satisfied by large buildings; accordingly, the genotype length for this case is set to 64. Given that some of the criteria might be conflicting, and given the impossibility of attaining the apartment percentage criterion, the best scores for this set should be lower than 6.2.

Multiple Criteria Population: **50** Set mutation rate: **0.005**



Actual mutation rate: **0.0009** Final average score: **3.7918** Time elapsed: **1823 s**



Champ Nº1

Shape code: **7 13 21 20 3 4 2 21 15 21 3 16 2 21 12 3 21 7 13 21 9 4 20 20 14 21 16 2 3 18 21 4 6 20 21 19 3 7 19 21 10 8 5 18 21 11 9 21 9 5 6 21 6 17 16 18 2 21 20 10 4 20 22 20**

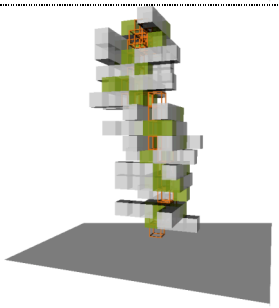
Score: **4.684771**

Appeared in generation: **3**

Nº of apartments: **36**
Nº of circulation blocks: **30**
Volume: **11136 m³**
Total area: **2784 m²**
Apartment area: **2304 m²**
Circulation area: **480 m²**
Height: **64 m**
Footprint: **28 m × 32 m**

Nº of apartments with balconies: **19**
Percentage of apartments with balconies: **52.777779 %**

Apartments with views to +i: **15**
Apartments with views to −i: **16**
Apartments with views to +j: **16**
Apartments with views to −j: **13**
Apartments with no views: **0**



Champ Nº2

Shape code: **17 13 21 20 3 4 2 21 15 21 3 16 2 21 12 3 21 7 13 21 9 4 20 20 14 21 16 2 3 18 21 4 6 20 21 19 3 7 19 21 10 8 5 18 21 11 9 21 9 5 6 21 6 17 16 18 2 21 20 10 4 20 22 20**

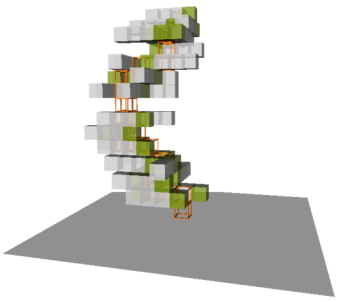
Score: **4.669131**

Appeared in generation: **2**

Nº of apartments: **35**
Nº of circulation blocks: **31**
Volume: **10944 m³**
Total area: **2736 m²**
Apartment area: **2240 m²**
Circulation area: **496 m²**
Height: **64 m**
Footprint: **28 m × 32 m**

Nº of apartments with balconies: **19**
Percentage of apartments with balconies: **54.285717 %**

Apartments with views to +i: **14**
Apartments with views to −i: **16**
Apartments with views to +j: **16**
Apartments with views to −j: **12**
Apartments with no views: **0**



Champ Nº3

Shape code: **15 19 11 19 2 21 11 2 18 8 21 4 3 18 7 19 21 19 11 7 19 21 11 2 17 21 11 9 19 21 18 21 4 1 6 20 21 18 2 1 17 4 21 14 17 5 17 21 9 21 16 3 2 18 18 18 21 3 19 18 17 4 21 11**

Score: **4.59014**

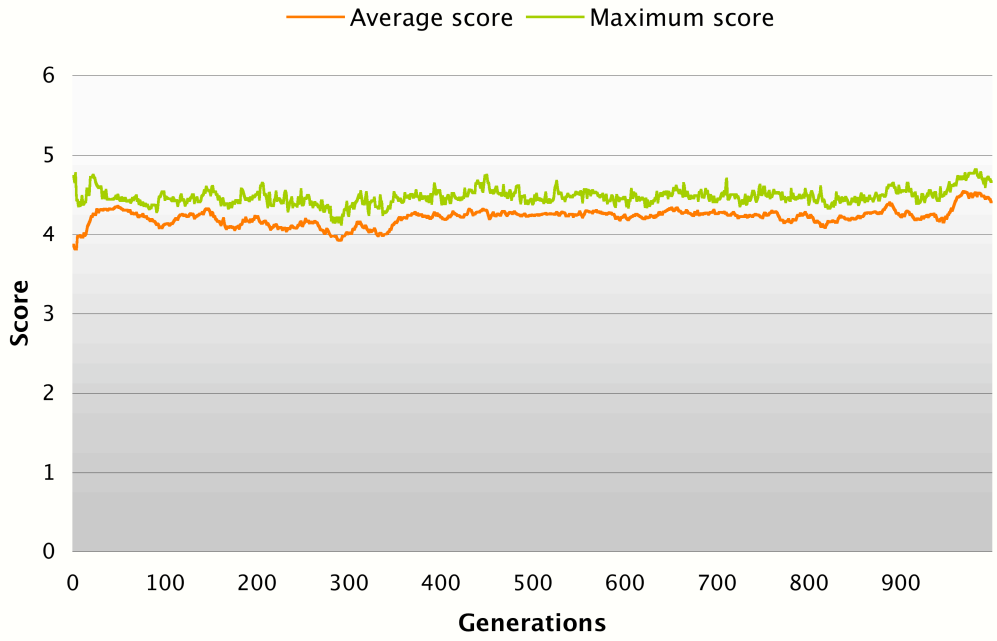
Appeared in generation: **3**

Nº of apartments: **30**
Nº of circulation blocks: **36**
Volume: **9984 m³**
Total area: **2496 m²**
Apartment area: **1920 m²**
Circulation area: **576 m²**
Height: **60 m**
Footprint: **36 m × 40 m**

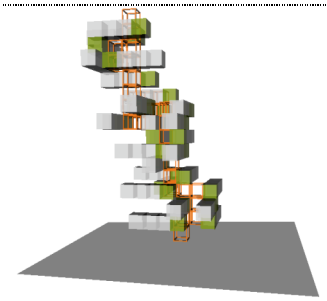
Nº of apartments with balconies: **15**
Percentage of apartments with balconies: **50 %**

Apartments with views to +i: **16**
Apartments with views to −i: **10**
Apartments with views to +j: **14**
Apartments with views to −j: **12**
Apartments with no views: **0**

Multiple Criteria Population: **50** Set mutation rate: **0.01**



Actual mutation rate: **0.0025** Final average score: **4.4106** Time elapsed: **2085 s**



Champ Nº1

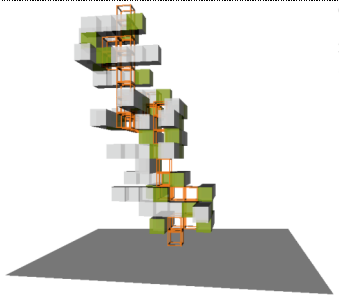
Shape code: **3 18 17 16 21 6 4 21 8 19 19 3 18 2 21 7 16 20 21 14 21 20 4 3 21 16 2 21 4 21 18 12 19 18 7 21 11 19 19 22 17 22 20 12 20 20 20 14 21 4 21 21 15 21 19 14 5 21 10 17 21 7 11 21**

Score: **4.816819**

Appeared in generation: **984**

Nº of apartments: **27**
Nº of circulation blocks: **39**
Volume: **9408 m³**
Total area: **2352 m²**
Apartment area: **1728 m²**
Circulation area: **624 m²**
Height: **64 m**
Footprint: **36 m x 44 m**

Nº of apartments with balconies: **13**
Percentage of apartments with balconies: **48.148148 %**
Apartments with views to +i: **20**
Apartments with views to −i: **8**
Apartments with views to +j: **13**
Apartments with views to −j: **15**
Apartments with no views: **0**



Champ Nº2

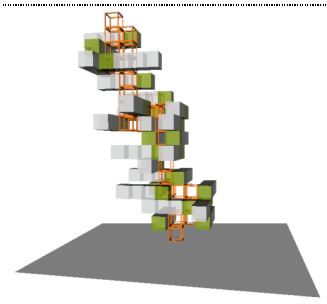
Shape code: **10 18 17 16 21 3 4 21 8 19 19 3 18 2 21 7 16 20 21 14 21 20 4 3 21 16 2 21 4 21 18 12 19 18 7 21 11 19 19 22 17 22 20 12 20 20 20 14 21 4 21 21 15 21 19 14 5 21 10 17 21 7 11 21**

Score: **4.816819**

Appeared in generation: **983**

Nº of apartments: **27**
Nº of circulation blocks: **39**
Volume: **9408m³**
Total area: **2352 m²**
Apartment area: **1728 m²**
Circulation area: **624 m²**
Height: **64 m**
Footprint: **36 m x 44 m**

Nº of apartments with balconies: **13**
Percentage of apartments with balconies: **48.148148 %**
Apartments with views to +i: **20**
Apartments with views to −i: **8**
Apartments with views to +j: **14**
Apartments with views to −j: **14**
Apartments with no views: **0**



Champ Nº3

Shape code: **10 18 17 16 21 3 4 21 8 19 19 3 18 2 21 7 16 20 21 14 21 20 4 3 21 16 2 21 4 21 18 12 19 18 7 21 11 19 19 22 17 22 20 12 20 20 20 14 21 4 21 21 15 21 19 14 5 21 10 17 21 7 19 21**

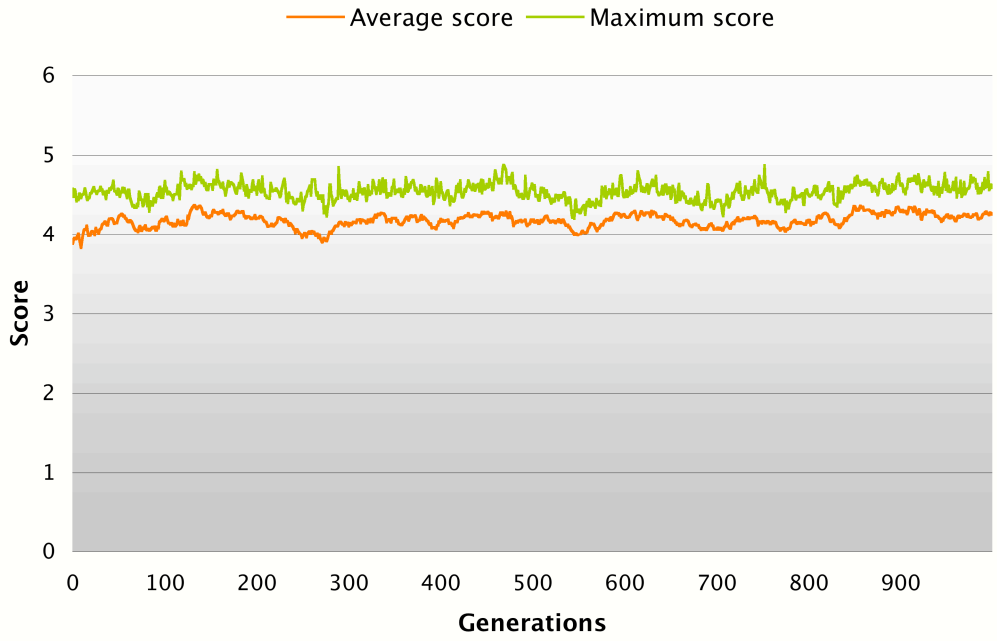
Score: **4.812881**

Appeared in generation: **984**

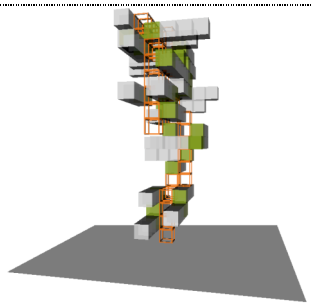
Nº of apartments: **26**
Nº of circulation blocks: **40**
Volume: **9216 m³**
Total area: **2304 m²**
Apartment area: **1664 m²**
Circulation area: **640 m²**
Height: **64 m**
Footprint: **36 m x 44 m**

Nº of apartments with balconies: **13**
Percentage of apartments with balconies: **50 %**
Apartments with views to +i: **19**
Apartments with views to −i: **8**
Apartments with views to +j: **13**
Apartments with views to −j: **13**
Apartments with no views: **0**

Multiple Criteria Population: **50** Set mutation rate: **0.05**



Actual mutation rate: **0.0136** Final average score: **4.2531** Time elapsed: **2875 s**



Champ №1

Shape code: **6 21 4 2 21 14 8 18 18 21 17 21 2 21 7 21 11 8 7 21 10 8 21 19 14 21 9 21 4 1 21 4 20 21 8 7 20 11 21 8 2 21 19 20 20 14 20 22 2 1 17 5 22 6 20 22 16 22 22 4 14 22 8 22**

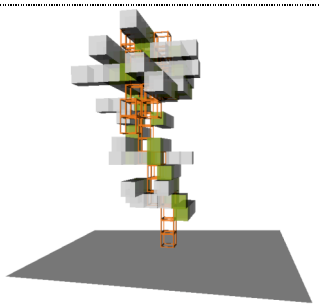
Score: **4.87365**

Appeared in generation: **470**

№ of apartments: **31**
№ of circulation blocks: **35**
Volume: **10176m³**
Total area: **2544 m²**
Apartment area: **1984 m²**
Circulation area: **560 m²**
Height: **64 m**
Footprint: **28 m × 52 m**

№ of apartments with balconies: **17**
Percentage of apartments with balconies: **54.838711 %**

Apartments with views to +i: **17**
Apartments with views to –i: **14**
Apartments with views to +j: **10**
Apartments with views to –j: **11**
Apartments with no views: **0**



Champ №2

Shape code: **21 8 21 11 7 19 21 6 12 21 16 18 21 19 11 1 2 21 6 16 21 2 21 14 8 21 3 21 2 17 17 20 12 21 16 21 20 2 16 20 13 21 19 19 20 4 22 22 17 1 20 20 6 22 2 4 3 22 17 4 22 19 20 22**

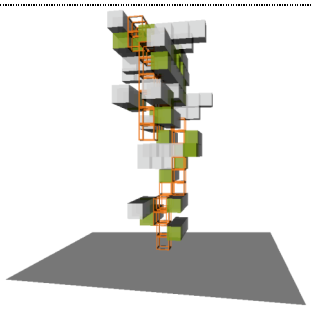
Score: **4.870448**

Appeared in generation: **754**

№ of apartments: **28**
№ of circulation blocks: **38**
Volume: **9600 m³**
Total area: **2400 m²**
Apartment area: **1792 m²**
Circulation area: **608 m²**
Height: **60 m**
Footprint: **32 m × 52 m**

№ of apartments with balconies: **18**
Percentage of apartments with balconies: **64.285713 %**

Apartments with views to +i: **15**
Apartments with views to –i: **13**
Apartments with views to +j: **9**
Apartments with views to –j: **10**
Apartments with no views: **0**



Champ №3

Shape code: **16 21 20 2 21 6 8 18 18 21 17 21 2 21 7 21 11 8 7 21 10 8 21 19 14 21 9 21 4 1 21 4 20 21 8 7 20 11 21 8 2 21 19 20 20 14 20 22 2 1 17 5 22 4 20 22 16 22 22 4 14 22 8 22**

Score: **4.861568**

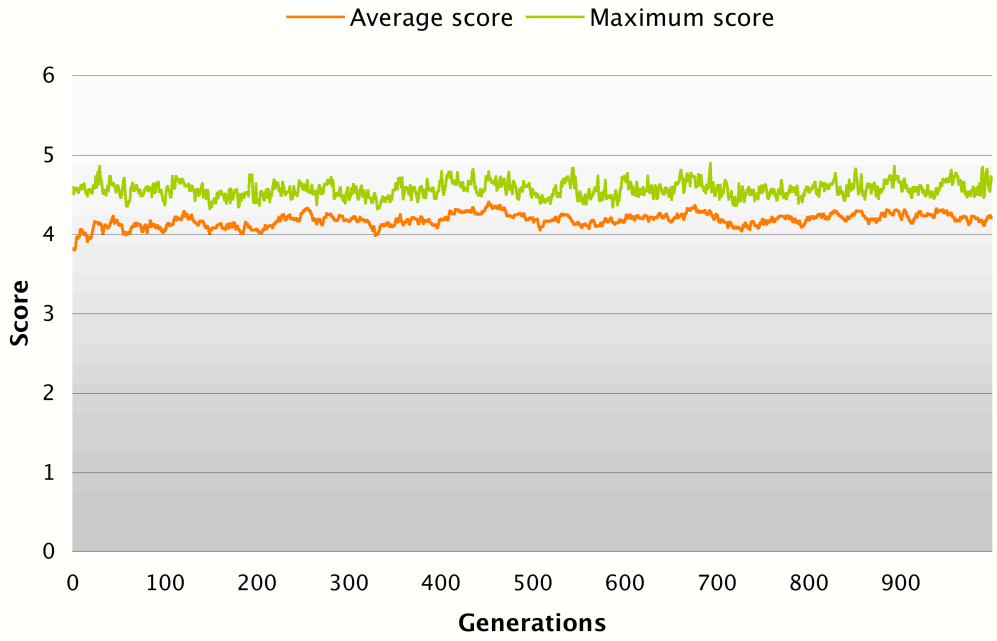
Appeared in generation: **471**

№ of apartments: **30**
№ of circulation blocks: **36**
Volume: **9984 m³**
Total area: **2496 m²**
Apartment area: **1920 m²**
Circulation area: **576 m²**
Height: **64 m**
Footprint: **28 m × 52 m**

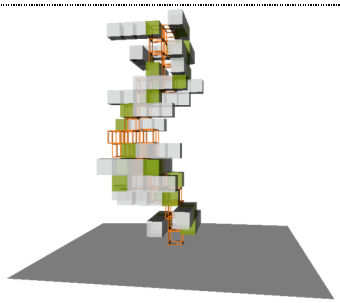
№ of apartments with balconies: **15**
Percentage of apartments with balconies: **50 %**

Apartments with views to +i: **18**
Apartments with views to –i: **13**
Apartments with views to +j: **10**
Apartments with views to –j: **9**
Apartments with no views: **2**

Multiple Criteria Population: **50** Set mutation rate: **0.1**



Actual mutation rate: **0.0235** Final average score: **4.2175** Time elapsed: **3482 s**

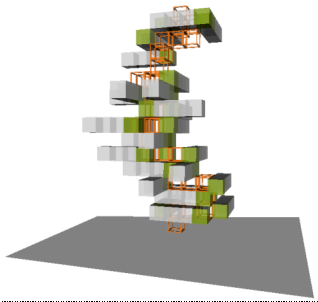


Champ Nº1

Shape code: **14 16 21 20 4 21 19 11 7 21 15 20 19 13 21 8 2 11 18 18 21 21 18 22 19 20 5 19 18 18 8 21 19 18 17 21 9 21 17 13 21 1 11 21 17 1 17 11 21 12 10 21 4 19 14 21 21 3 17 4 18 22 7 8**

Score: **4.894419** Appeared in generation: **695**

Nº of apartments: **26** Nº of apartments with balconies: **16**
Nº of circulation blocks: **40** Percentage of apartments with balconies: **61.538464 %**
Volume: **9216 m³**
Total area: **2304 m²** Apartments with views to +i: **15**
Apartment area: **1664m²** Apartments with views to −i: **10**
Circulation area: **640 m²** Apartments with views to +j: **12**
Height: **64 m** Apartments with views to −j: **12**
Footprint: **28 m × 48 m** Apartments with no views: **0**

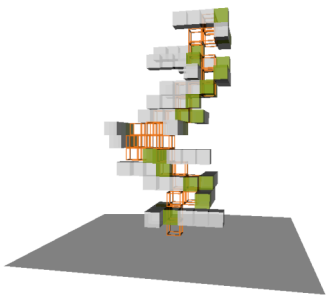


Champ Nº2

Shape code: **2 17 11 17 18 8 21 7 3 21 20 16 19 19 6 21 7 3 4 21 8 19 7 19 3 21 4 15 21 20 2 20 3 17 8 18 21 8 20 5 21 7 6 4 21 8 19 18 17 21 21 18 7 21 10 17 17 21 11 8 19 2 18 21**

Score: **4.856903** Appeared in generation: **31**

Nº of apartments: **29** Nº of apartments with balconies: **15**
Nº of circulation blocks:**37** Percentage of apartments with balconies: **51.724136 %**
Volume: **9792 m³**
Total area: **2448 m²** Apartments with views to +i: **21**
Apartment area: **1856 m²** Apartments with views to −i: **7**
Circulation area: **592 m²** Apartments with views to +j: **13**
Height: **64 m** Apartments with views to −j: **9**
Footprint: **36 m × 40 m** Apartments with no views: **0**



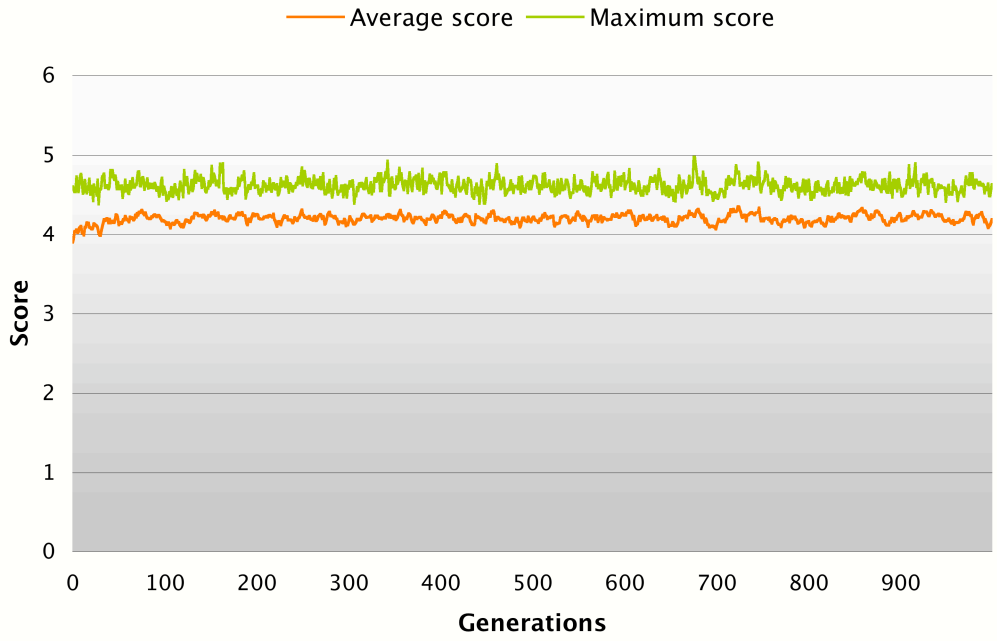
Champ Nº3

Shape code: **14 5 21 18 17 16 18 21 3 8 19 21 19 3 21 20 2 5 21 18 21 4 19 22 19 18 22 19 21 18 14 21 6 17 17 21 17 17 3 8 21 20 11 21 17 11 15 17 21 14 16 21 15 19 3 21 7 21 17 18 12 21 20 11**

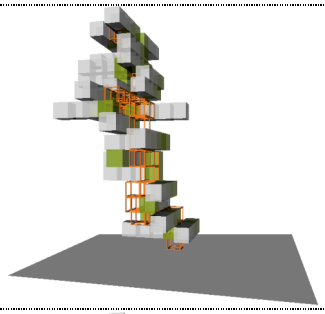
Score: **4.856197** Appeared in generation: **895**

Nº of apartments: **23** Nº of apartments with balconies: **13**
Nº of circulation blocks: **43** Percentage of apartments with balconies: **56.521736 %**
Volume: **8640 m³**
Total area: **2160 m²** Apartments with views to +i: **18**
Apartment area: **1472 m²** Apartments with views to −i: **7**
Circulation area: **688 m²** Apartments with views to +j: **15**
Height: **64 m** Apartments with views to −j: **11**
Footprint: **36 m × 40 m** Apartments with no views: **0**

Multiple Criteria Population: **50** Set mutation rate: **0.5**



Actual mutation rate: **0.1176** Final average score: **4.1905** Time elapsed: **7543 s**



Champ Nº1

Shape code: **3 12 21 19 2 20 4 20 19 21 16 20 21 17 12 21 7 21 7 20 20 10 21 15 4 21 14 21 18 19 1 19 3 2 21 18 6 18 18 18 19 18 15 17 21 6 4 21 12 19 3 21 4 15 3 21 4 7 21 8 20 14 20 4**

Score: **4.976903**

Appeared in generation: **678**

Nº of apartments: **29**
Nº of circulation blocks: **37**
Volume: **9792 m³**
Total area: **2448 m²**
Apartment area: **1856 m²**
Circulation area: **592 m²**
Height: **64 m**
Footprint: **36 m × 52 m**

Nº of apartments with balconies: **17**
Percentage of apartments with balconies: **58.620689 %**

Apartments with views to +i: 19
Apartments with views to −i: **10**
Apartments with views to +j: **6**
Apartments with views to −j: **15**
Apartments with no views: **0**

Champ Nº2

Shape code: **2 12 21 19 1 20 4 20 19 21 16 18 21 17 12 21 7 21 7 20 20 10 21 15 4 21 14 21 18 19 1 19 3 2 21 18 18 18 18 18 19 18 15 17 21 15 4 21 12 19 10 21 4 15 3 21 4 7 21 8 20 14 20 4**

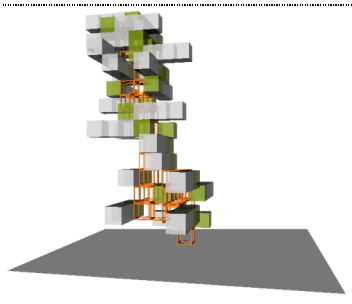
Score: **4.973305**

Appeared in generation: **677**

Nº of apartments: **28**
Nº of circulation blocks: **38**
Volume: **9600 m³**
Total area: **2400 m²**
Apartment area: **1792 m²**
Circulation area: **608 m²**
Height: **64 m**
Footprint: **40 m × 56 m**

Nº of apartments with balconies: **15**
Percentage of apartments with balconies: **53.571426 %**

Apartments with views to +i: **20**
Apartments with views to −i: **9**
Apartments with views to +j: **6**
Apartments with views to −j: **15**
Apartments with no views: **0**



Champ Nº3

Shape code: **15 21 21 16 20 21 20 22 20 21 20 4 22 22 4 18 19 19 14 21 18 18 16 21 14 20 21 12 21 4 21 3 15 21 13 19 18 21 18 7 19 21 2 17 16 21 6 15 4 21 5 19 6 21 11 15 21 13 18 18 12 21 7 3**

Score: **4.928265**

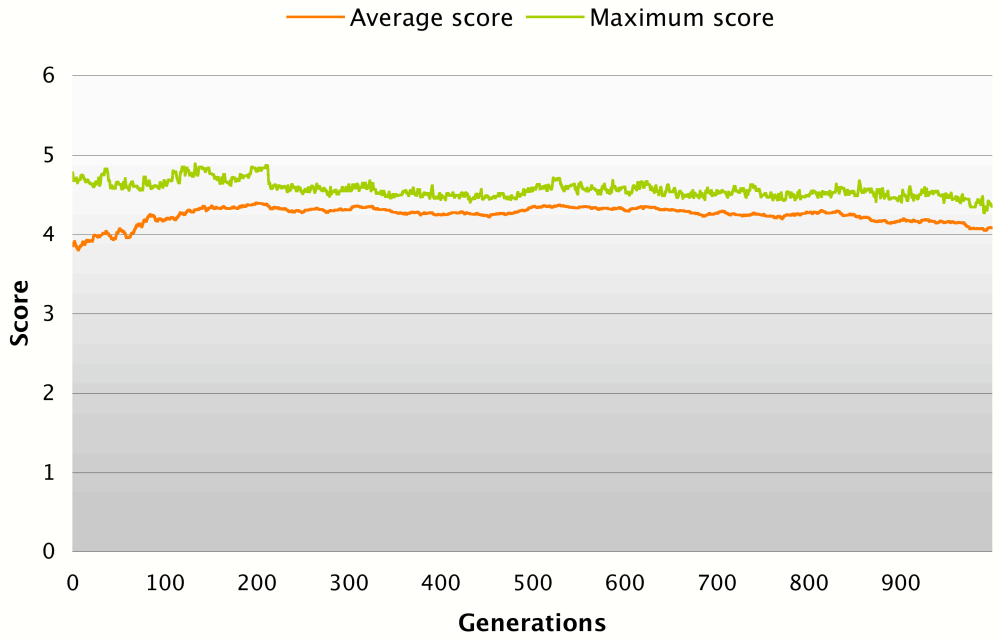
Appeared in generation: **344**

Nº of apartments: **26**
Nº of circulation blocks: **40**
Volume: **9216 m³**
Total area: **2304 m²**
Apartment area: **1664 m²**
Circulation area: **640 m²**
Height: **64 m**
Footprint: **36 m × 44 m**

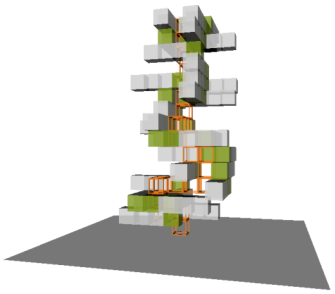
Nº of apartments with balconies: **16**
Percentage of apartments with balconies: **61.538464 %**

Apartments with views to +i: **19**
Apartments with views to −i: **10**
Apartments with views to +j: **12**
Apartments with views to −j: **15**
Apartments with no views: **0**

Multiple Criteria Population: **200** Set mutation rate: **0.005**



Actual mutation rate: **0.001** Final average score: **4.0889** Time elapsed: **8948 s**



Champ Nº1

Shape code: **9 19 20 11 21 16 20 5 10 21 18 15 17 17 17 4 21 6 16 21 8 7 19 21 13 8 19 10 21 14 18 18 7 21 14 21 4 1 10 21 3 18 1 4 21 4 21 20 11 21 10 1 4 21 18 6 4 18 18 16 18 21 16 10**

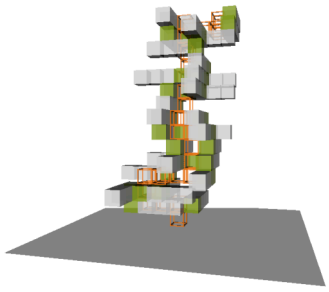
Score: **4.889483**

Appeared in generation: **135**

Nº of apartments: **33**
Nº of circulation blocks: **33**
Volume: **10560 m³**
Total area: **2640 m²**
Apartment area: **2112 m²**
Circulation area: **528 m²**
Height: **64 m**
Footprint: **28 m × 48 m**

Nº of apartments with balconies: **17**
Percentage of apartments with balconies: **51.515152 %**

Apartments with views to +i: **17**
Apartments with views to −i: **11**
Apartments with views to +j: **13**
Apartments with views to −j: **13**
Apartments with no views: **0**



Champ Nº2

Shape code: **8 19 19 13 21 16 20 5 6 21 17 18 7 17 17 4 21 16 7 6 21 10 8 21 4 9 19 10 21 14 18 18 12 21 14 21 4 1 10 21 3 18 1 4 21 12 21 20 11 21 10 1 4 21 18 6 18 17 17 16 18 21 16 18**

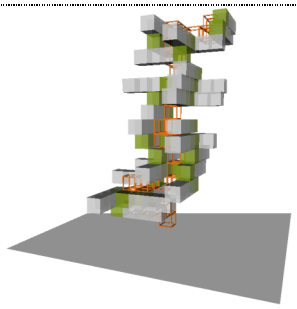
Score: **4.868701**

Appeared in generation: **212**

Nº of apartments: **32**
Nº of circulation blocks: **34**
Volume: **10368 m³**
Total area: **2592 m²**
Apartment area: **2048 m²**
Circulation area: **544 m²**
Height: **64 m**
Footprint: **32 m × 48 m**

Nº of apartments with balconies: **18**
Percentage of apartments with balconies: **56.25 %**

Apartments with views to +i: **17**
Apartments with views to −i: **14**
Apartments with views to +j: **13**
Apartments with views to −j: **15**
Apartments with no views: **0**



Champ Nº3

Shape code: **20 19 19 13 21 16 20 5 6 21 17 18 7 17 17 4 21 16 7 6 21 10 8 21 4 9 19 10 21 14 18 18 12 21 14 21 4 1 10 21 3 18 1 4 21 12 21 20 11 21 10 1 4 21 18 6 18 17 17 16 18 21 16 18**

Score: **4.858166**

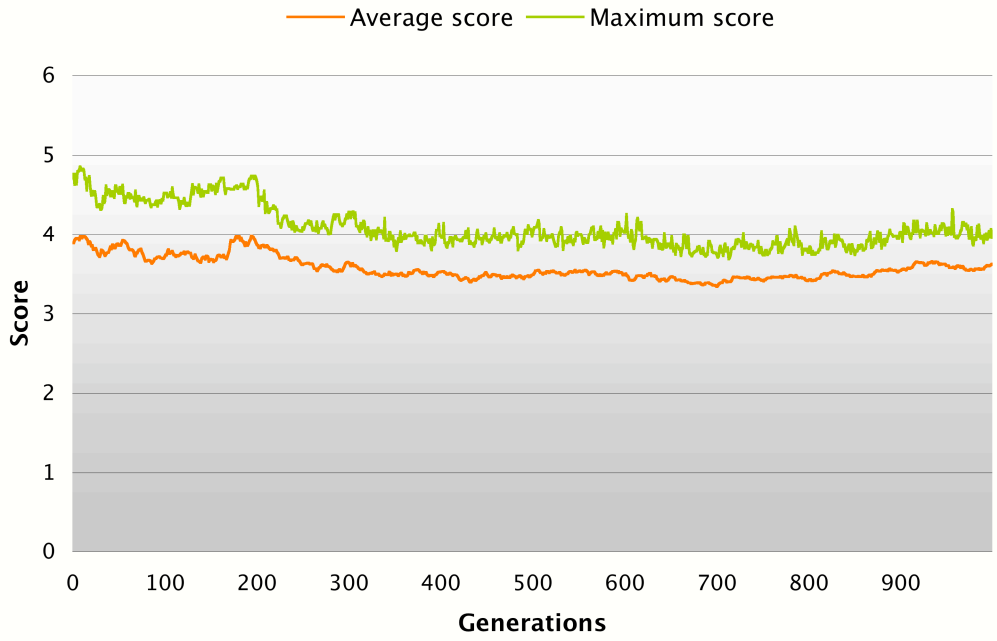
Appeared in generation: **214**

Nº of apartments: **31**
Nº of circulation blocks: **35**
Volume: **10176 m³**
Total area: **2544 m²**
Apartment area: **1984 m²**
Circulation area: **560 m²**
Height: **64 m**
Footprint: **32 m × 48 m**

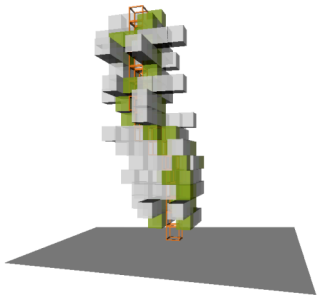
Nº of apartments with balconies: **18**
Percentage of apartments with balconies: **58.064514 %**

Apartments with views to +i: **16**
Apartments with views to −i: **14**
Apartments with views to +j: **12**
Apartments with views to −j: **15**
Apartments with no views: **0**

Multiple Criteria Population: **200** Set mutation rate: **0.01**



Actual mutation rate: **0.002** Final average score: **3.6234** Time elapsed: **10156 s**



Champ Nº1

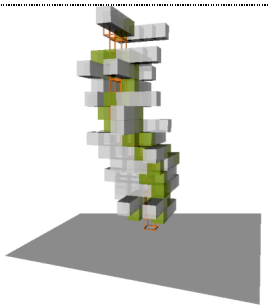
Shape code: **16 10 21 6 7 4 21 11 7 8 21 16 3 18 2 18 21 12 6 20 20 20 11 21 7 12 3 19 21 3 8 7 19 21 4 1 2 21 4 20 2 21 5 14 18 7 16 21 14 20 8 21 6 4 9 21 8 10 5 21 16 7 21 15**

Score: **4.854313**
Appeared in generation: **10**

Nº of apartments: **40**
Nº of circulation blocks: **26**
Volume: **11904 m³**
Total area: **2976 m²**
Apartment area: **2560 m²**
Circulation area: **416 m²**
Height: **64 m**
Footprint: **28 m x 40 m**

Nº of apartments with balconies: **19**
Percentage of apartments with balconies: **47.5 %**

Apartments with views to +i: **19**
Apartments with views to −i: **13**
Apartments with views to +j: **17**
Apartments with views to −j: **15**
Apartments with no views: **0**



Champ Nº2

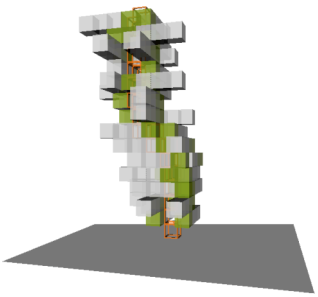
Shape code: **16 10 21 6 7 4 21 11 7 8 21 16 3 18 2 18 21 12 6 20 20 20 11 21 7 12 3 19 21 3 8 7 19 21 4 1 2 21 4 20 2 21 5 14 18 7 16 21 14 20 8 21 6 4 9 21 8 10 5 21 21 1 20 4**

Score: **4.82727**
Appeared in generation: **13**

Nº of apartments: **39**
Nº of circulation blocks: **27**
Volume: **11712 m³**
Total area: **2928 m²**
Apartment area: **2496 m²**
Circulation area: **432 m²**
Height: **64 m**
Footprint: **28 m x 44 m**

Nº of apartments with balconies: **17**
Percentage of apartments with balconies: **43.589745 %**

Apartments with views to +i: **18**
Apartments with views to −i: **13**
Apartments with views to +j: **15**
Apartments with views to −j: **14**
Apartments with no views: **0**



Champ Nº3

Shape code: **16 10 21 6 7 4 21 11 7 8 21 16 3 18 2 18 21 12 6 20 20 20 11 21 7 12 3 19 21 3 8 7 19 21 4 1 2 21 4 20 2 21 5 14 18 7 16 21 14 20 8 21 6 4 9 21 8 10 5 21 16 3 18 21**

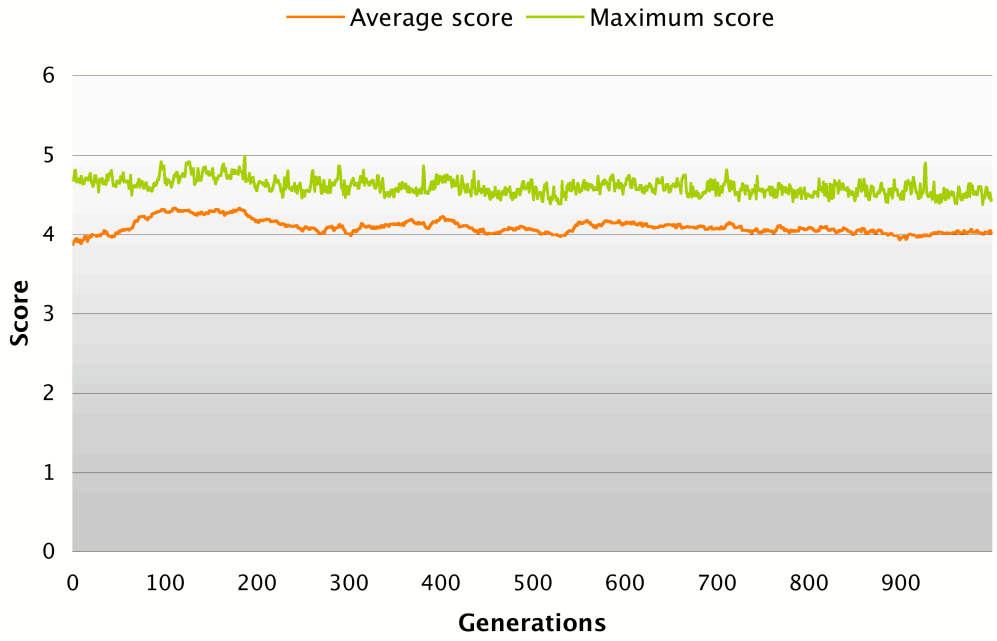
Score: **4.812911**
Appeared in generation: **9**

Nº of apartments: **39**
Nº of circulation blocks: **27**
Volume: **11712 m³**
Total area: **2928 m²**
Apartment area: **2496 m²**
Circulation area: **432 m²**
Height: **64 m**
Footprint: **28 m x 40 m**

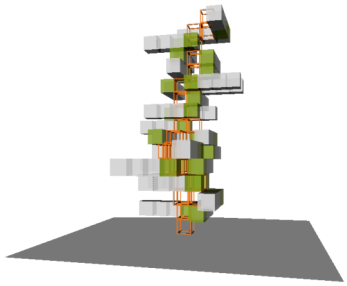
Nº of apartments with balconies: **18**
Percentage of apartments with balconies: **46.153847 %**

Apartments with views to +i: **18**
Apartments with views to −i: **12**
Apartments with views to +j: **17**
Apartments with views to −j: **13**
Apartments with no views: **0**

Multiple Criteria Population: **200** Set mutation rate: **0.05**



Actual mutation rate: **0.0121** Final average score: **4.0189** Time elapsed: **11411 s**



Champ Nº1

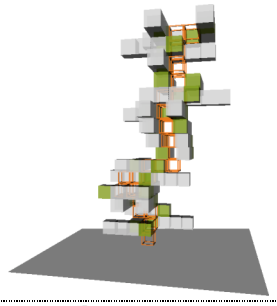
Shape code: **16 7 21 20 17 12 21 3 7 19 21 8 20 20 3 21 12 18 18 19 18 18 21 10 18 22 22 18 7 8 21 17 8 7 21 7 12 21 19 7 21 9 11 21 8 21 2 17 1 5 21 14 8 21 10 8 21 3 2 21 17 4 15 21**

Score: **4.981568**

Appeared in generation: **189**

Nº of apartments: **30**
Nº of circulation blocks: **36**
Volume: **9984 m³**
Total area: **2496 m²**
Apartment area: **1920 m²**
Circulation area: **576 m²**
Height: **64 m**
Footprint: **32 m × 48 m**

Nº of apartments with balconies: **17**
Percentage of apartments with balconies: **56.666668 %**
Apartments with views to +i: **20**
Apartments with views to −i: **8**
Apartments with views to +j: **13**
Apartments with views to −j: **12**
Apartments with no views: **0**



Champ Nº2

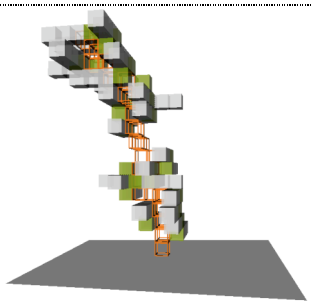
Shape code: **3 18 10 1 21 15 19 21 6 4 21 20 16 20 19 13 18 21 17 17 5 7 21 2 11 17 17 18 21 12 2 21 4 20 20 14 20 21 20 14 8 21 7 6 4 20 21 17 1 11 21 8 10 21 20 14 21 3 9 21 6 17 7 4**

Score: **4.91494**

Appeared in generation: **98**

Nº of apartments: **31**
Nº of circulation blocks: **35**
Volume: **10176 m³**
Total area: **2544 m²**
Apartment area: **1984 m²**
Circulation area: **560 m²**
Height: **64 m**
Footprint: **40 m × 52 m**

Nº of apartments with balconies: **20**
Percentage of apartments with balconies: **64.516129 %**
Apartments with views to +i: **17**
Apartments with views to −i: **14**
Apartments with views to +j: **11**
Apartments with views to −j: **17**
Apartments with no views: **0**



Champ Nº3

Shape code: **18 8 21 20 10 12 21 6 20 4 21 18 4 18 19 19 2 21 6 5 8 7 21 2 4 18 17 8 21 4 19 21 20 20 21 14 20 21 12 2 21 1 20 14 12 20 21 15 3 19 21 4 20 19 7 6 21 8 18 21 3 4 7 19**

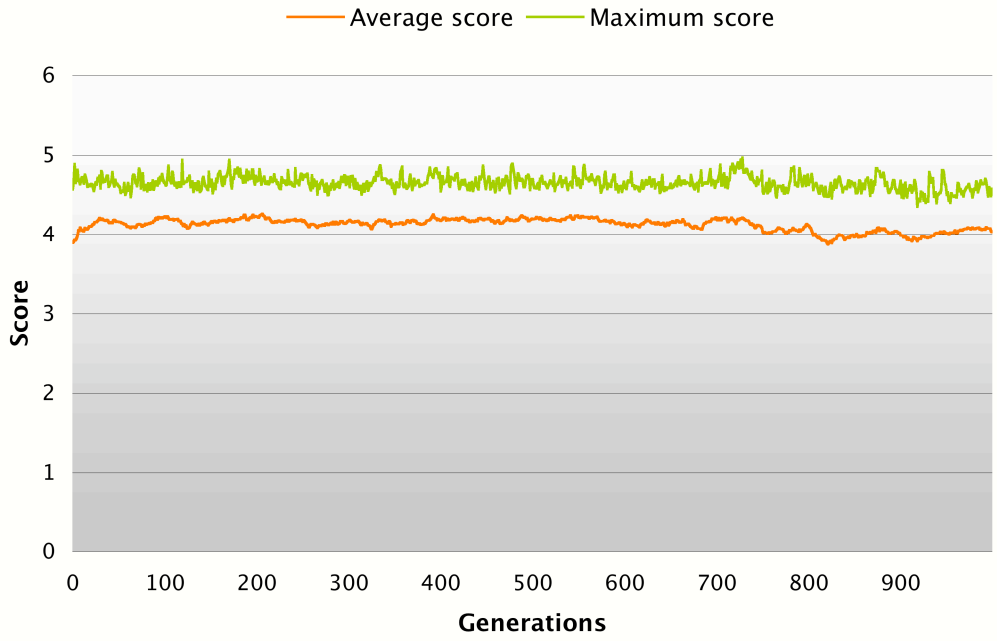
Score: **4.914902**

Appeared in generation: **129**

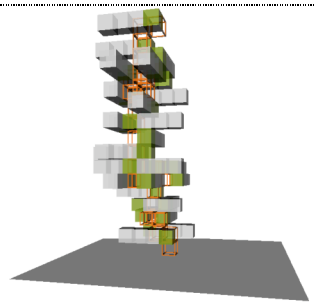
Nº of apartments: **30**
Nº of circulation blocks: **36**
Volume: **9984 m³**
Total area: **2496 m²**
Apartment area: **1920 m²**
Circulation area: **576 m²**
Height: **64 m**
Footprint: **36 m × 52 m**

Nº of apartments with balconies: **18**
Percentage of apartments with balconies: **60.000004 %**
Apartments with views to +i: **17**
Apartments with views to −i: **10**
Apartments with views to +j: **11**
Apartments with views to −j: **15**
Apartments with no views: **0**

Multiple Criteria Population: **200** Set mutation rate: **0.1**



Actual mutation rate: **0.0247** Final average score: **4.0305** Time elapsed: **13318 s**



Champ Nº1

Shape code: **19 11 18 9 21 19 18 8 18 2 21 13 8 21 7 21 14 17 8 20 21 7 20 8 3 19 22 20 20 20 14 16 21 5 7 21 7 11 21 16 21 14 5 21 4 21 1 14 20 21 17 15 22 20 21 4 6 21 8 21 7 21 17 3**

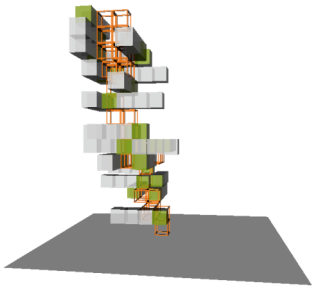
Score: **4.968235**

Appeared in generation: **730**

Nº of apartments: **30**
Nº of circulation blocks: **36**
Volume: **9984 m³**
Total area: **2496 m²**
Apartment area: **1920 m²**
Circulation area: **576 m²**
Height: **64 m**
Footprint: **28 m × 56 m**

Nº of apartments with balconies: **14**
Percentage of apartments with balconies: **46.666668 %**

Apartments with views to +i: **19**
Apartments with views to −i: **10**
Apartments with views to +j: **11**
Apartments with views to −j: **13**
Apartments with no views: **0**



Champ Nº2

Shape code: **19 11 18 18 21 19 18 8 18 2 21 3 8 21 7 21 14 17 20 20 21 7 20 8 3 19 22 20 20 20 14 16 21 5 7 21 7 11 21 19 21 14 5 21 4 21 1 14 20 21 17 15 22 20 21 4 19 14 21 14 17 21 19 10**

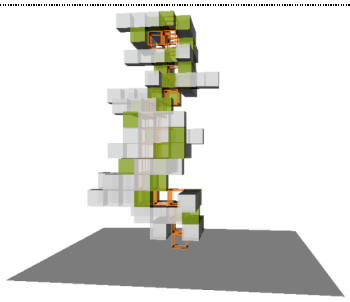
Score: **4.94299**

Appeared in generation: **728**

Nº of apartments: **26**
Nº of circulation blocks: **40**
Volume: **9216 m³**
Total area: **2304 m²**
Apartment area: **1664 m²**
Circulation area: **640 m²**
Height: **60 m**
Footprint: **28 m × 60 m**

Nº of apartments with balconies: **13**
Percentage of apartments with balconies: **50 %**

Apartments with views to +i: **18**
Apartments with views to −i: **9**
Apartments with views to +j: **9**
Apartments with views to −j: **11**
Apartments with no views: **0**



Champ Nº3

Shape code: **14 12 21 7 20 11 21 4 19 19 3 7 21 7 11 21 13 18 16 18 21 20 17 16 3 21 11 19 1 21 13 18 21 11 2 18 18 21 3 15 17 21 5 18 6 16 21 2 16 21 12 3 2 21 8 21 20 10 20 3 8 22 19 10**

Score: **4.940392**

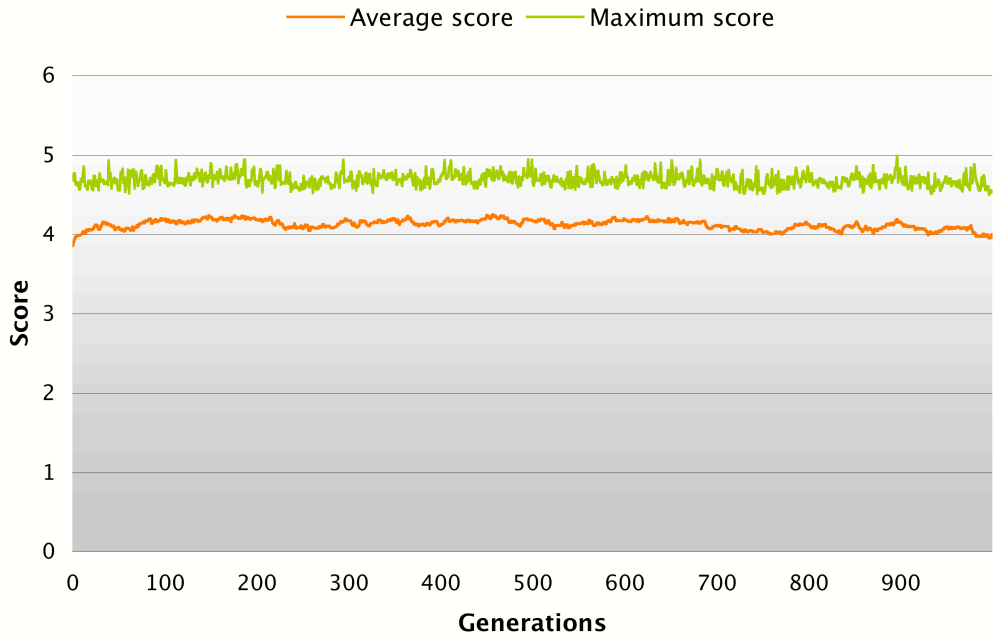
Appeared in generation: **121**

Nº of apartments: **33**
Nº of circulation blocks: **33**
Volume: **10560 m³**
Total area: **2640 m²**
Apartment area: **2112 m²**
Circulation area: **528 m²**
Height: **64 m**
Footprint: **32 m × 48 m**

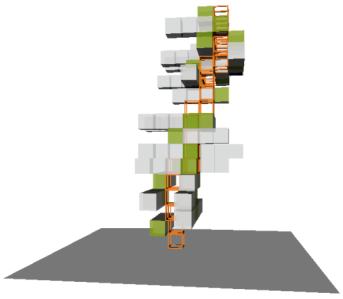
Nº of apartments with balconies: **17**
Percentage of apartments with balconies: **51.515152 %**

Apartments with views to +i: **20**
Apartments with views to −i: **11**
Apartments with views to +j: **18**
Apartments with views to −j: **15**
Apartments with no views: **0**

Multiple Criteria Population: **200** Set mutation rate: **0.5**



Actual mutation rate: **0.1251** Final average score: **4.0015** Time elapsed: **23608 s**



Champ №1

Shape code: **10 21 8 20 4 21 4 6 21 18 2 12 21 17 3 18 9 21 3 1 21 13 21 16 7 3 21 17 21 15 21 14 21 9 17 4 20 21 10 16 21 3 21 2 18 18 22 10 18 18 15 22 8 19 7 3 22 19 18 17 12 18 19 7**

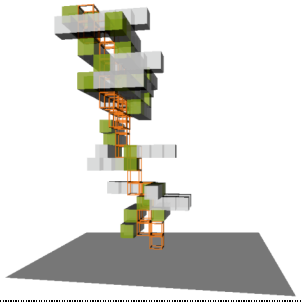
Score: **4.988235**

Appeared in generation: **898**

№ of apartments: **30**
№ of circulation blocks: **36**
Volume: **9984 m³**
Total area: **2496 m²**
Apartment area: **1920 m²**
Circulation area: **576 m²**
Height: **64 m**
Footprint: **28 m x 48 m**

№ of apartments with balconies: **17**
Percentage of apartments with balconies: **56.666668 %**

Apartments with views to +i: **19**
Apartments with views to −i: **11**
Apartments with views to +j: **14**
Apartments with views to −j: **13**
Apartments with no views: **0**



Champ №2

Shape code: **19 10 21 18 10 8 21 1 5 21 7 4 19 21 21 3 2 21 5 19 14 21 2 21 10 21 21 17 11 21 20 21 14 8 20 22 20 8 22 22 20 8 22 20 21 20 20 21 13 2 8 21 5 21 10 8 21 20 11 8 19 21 2 1**

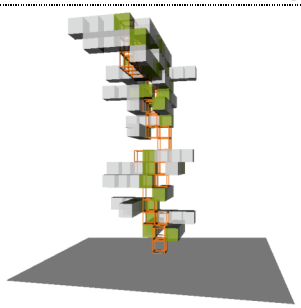
Score: **4.941876**

Appeared in generation: **497**

№ of apartments: **28**
№ of circulation blocks: **38**
Volume: **9600 m³**
Total area: **2400 m²**
Apartment area: **1792 m²**
Circulation area: **608 m²**
Height: **64 m**
Footprint: **32 m x 52 m**

№ of apartments with balconies: **17**
Percentage of apartments with balconies: **60.714287 %**

Apartments with views to +i: **16**
Apartments with views to −i: **14**
Apartments with views to +j: **15**
Apartments with views to −j: **7**
Apartments with no views: **0**



Champ №3

Shape code: **18 16 2 21 19 1 21 7 6 21 16 21 3 17 21 18 12 19 19 11 21 17 17 13 21 21 19 14 21 15 21 14 20 16 21 3 18 8 10 21 6 12 21 1 20 20 12 10 21 16 20 20 20 12 22 22 15 20 12 20 21 11 21 3**

Score: **4.941876**

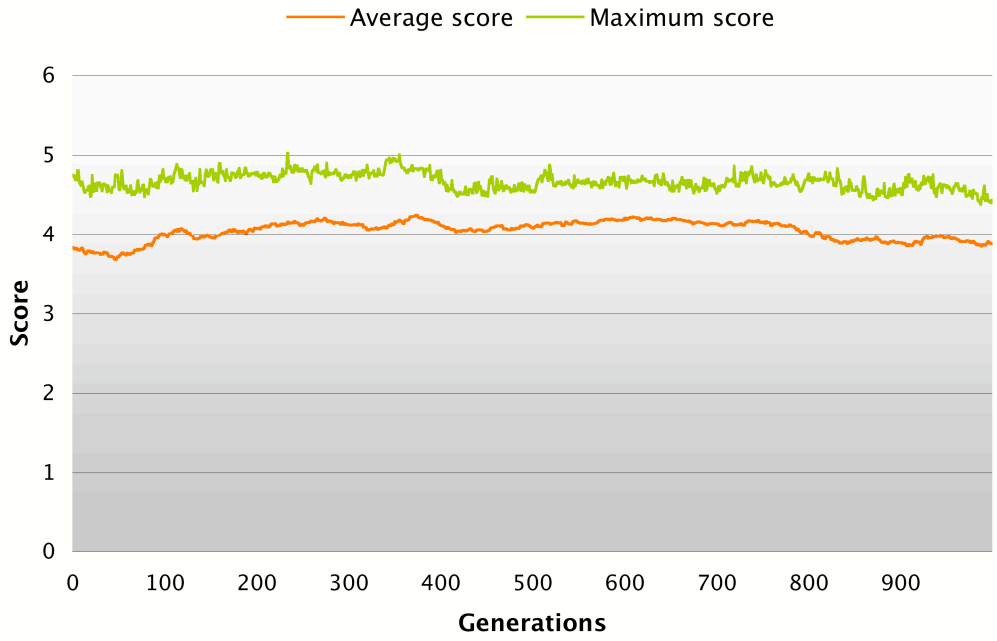
Appeared in generation: **189**

№ of apartments: **28**
№ of circulation blocks: **38**
Volume: **9600 m³**
Total area: **2400 m²**
Apartment area: **1792 m²**
Circulation area: **608 m²**
Height: **64 m**
Footprint: **28 m x 48 m**

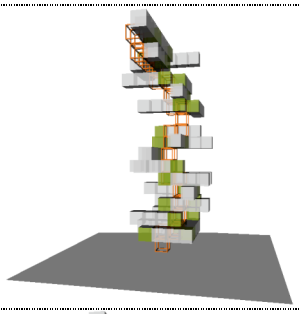
№ of apartments with balconies: **15**
Percentage of apartments with balconies: **53.571426 %**

Apartments with views to +i: **18**
Apartments with views to −i: **13**
Apartments with views to +j: **12**
Apartments with views to −j: **9**
Apartments with no views: **0**

Multiple Criteria Population: **500** Set mutation rate: **0.005**



Actual mutation rate: **0.0011** Final average score: **3.8737** Time elapsed: **23807 s**



Champ Nº1

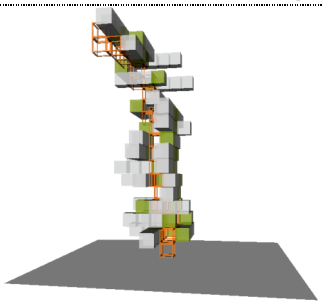
Shape code: **13 17 8 18 7 21 3 18 15 21 13 18 15 21 7 4 20 21 13 19 1 21 11 7 21 6 18 21 4 2 1 21 9 17 21 21 3 8 18 19 21 4 20 21 16 20 3 19 21 14 21 1 2 20 20 12 22 4 20 20 20 21 4 20**

Score: **5.02242**

Appeared in generation: **235**

Nº of apartments: **29**
Nº of circulation blocks: **37**
Volume: **9792m³**
Total area: **2448 m²**
Apartment area: **1856 m²**
Circulation area: **592 m²**
Height: **64 m**
Footprint: **28 m × 52 m**

Nº of apartments with balconies: **17**
Percentage of apartments with balconies: **58.620689 %**
Apartments with views to +i: **18**
Apartments with views to −i: **9**
Apartments with views to +j: **13**
Apartments with views to −j: **15**
Apartments with no views: **0**



Champ Nº2

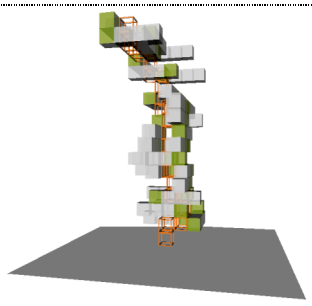
Shape code: **18 6 8 18 7 21 3 15 21 19 13 18 15 21 4 21 7 12 6 20 20 21 4 7 21 6 15 4 21 14 16 21 9 4 21 4 18 8 18 19 21 4 2 21 21 11 1 19 21 10 21 1 20 20 20 16 22 4 20 20 20 19 21 12**

Score: **5.002682**

Appeared in generation: **357**

Nº of apartments: **31**
Nº of circulation blocks:35
Volume: **10176 m³**
Total area: **2544 m²**
Apartment area: **1984 m²**
Circulation area: **560 m²**
Height: **64 m**
Footprint: **24 m × 48 m**

Nº of apartments with balconies: **16**
Percentage of apartments with balconies: **51.6129 %**
Apartments with views to +i: **20**
Apartments with views to −i: **12**
Apartments with views to +j: **9**
Apartments with views to −j: **15**
Apartments with no views: **0**



Champ Nº3

Shape code: **18 17 8 18 7 21 3 15 21 19 13 18 15 21 4 21 7 12 6 20 20 21 11 7 21 6 15 4 21 14 16 21 9 4 21 4 18 8 18 19 21 4 2 21 21 11 1 19 21 10 21 1 20 20 20 16 22 4 20 20 20 13 21 10**

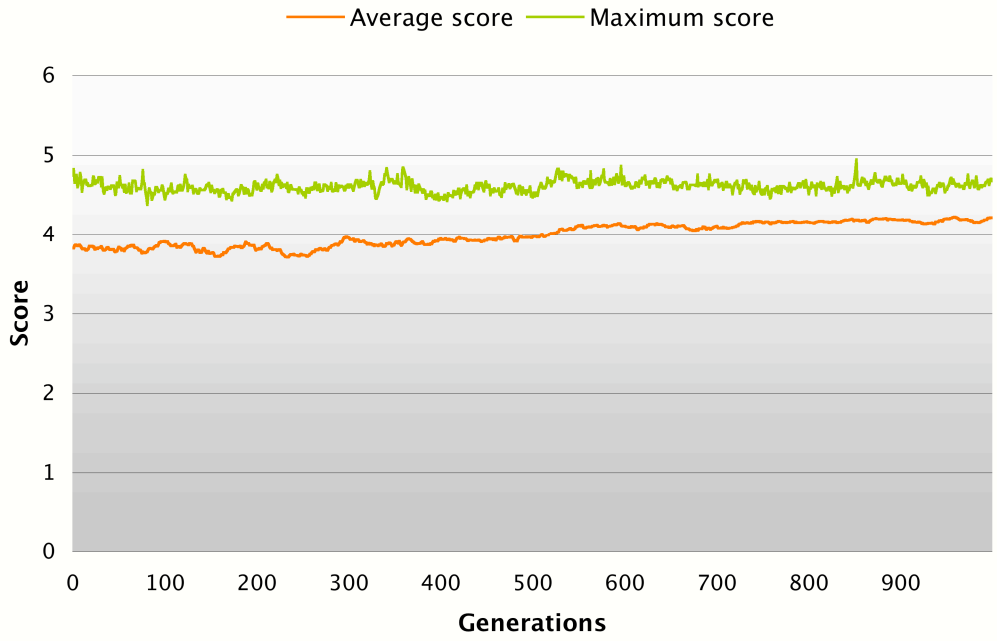
Score: **4.962682**

Appeared in generation: **346**

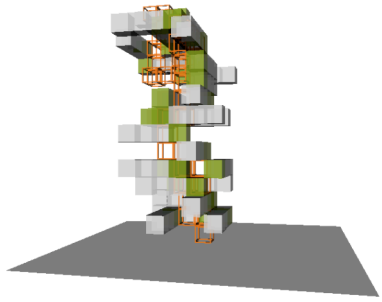
Nº of apartments: **31**
Nº of circulation blocks: **35**
Volume: **10176 m³**
Total area: **2544 m²**
Apartment area: **1984 m²**
Circulation area: **560m²**
Height: **64 m**
Footprint: **24 m × 44 m**

Nº of apartments with balconies: **16**
Percentage of apartments with balconies: **51.6129 %**
Apartments with views to +i: **20**
Apartments with views to −i: **11**
Apartments with views to +j: **12**
Apartments with views to −j: **15**
Apartments with no views: **0**

Multiple Criteria Population: **500** Set mutation rate: **0.01**



Actual mutation rate: **0.0022** Final average score: **4.209** Time elapsed: **25277 s**



Champ Nº1

Shape code: **12 19 6 15 21 15 4 21 9 20 17 3 21 16 19 3 2 21 4 19 21 11 21 2 1 17 13 21 12 10 21 16 21 16 21 20 4 21 2 3 18 18 18 18 22 19 22 1 19 19 18 21 9 21 6 18 16 21 18 21 16 21 15 19**

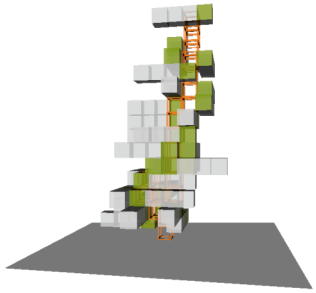
Score: **4.941876**

Appeared in generation: **854**

Nº of apartments: **28**
Nº of circulation blocks: **38**
Volume: **9600 m³**
Total area: **2400 m²**
Apartment area: **1792 m²**
Circulation area: **608 m²**
Height: **64 m**
Footprint: **32 m × 52 m**

Nº of apartments with balconies: **18**
Percentage of apartments with balconies: **64.285713 %**

Apartments with views to +i: **15**
Apartments with views to −i: **12**
Apartments with views to +j: **13**
Apartments with views to −j: **13**
Apartments with no views: **0**



Champ Nº2

Shape code: **12 19 3 18 21 18 10 21 20 16 21 14 20 20 20 5 22 6 20 5 17 22 12 6 18 7 21 17 21 21 16 20 5 10 21 3 8 18 21 7 11 21 7 12 21 7 17 16 21 14 8 21 3 21 10 18 16 21 10 21 21 11 18 22**

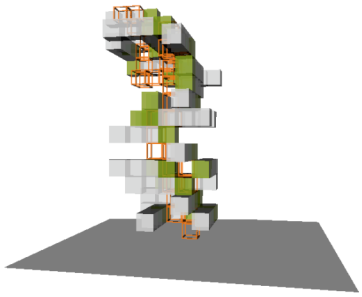
Score: **4.862421**

Appeared in generation: **598**

Nº of apartments: **29**
Nº of circulation blocks: **37**
Volume: **9792 m³**
Total area: **2448 m²**
Apartment area: **1856 m²**
Circulation area: **592m²**
Height: **64 m**
Footprint: **36 m × 44 m**

Nº of apartments with balconies: **17**
Percentage of apartments with balconies: **58.620689 %**

Apartments with views to +i: **18**
Apartments with views to −i:11
Apartments with views to +j: **15**
Apartments with views to −j: **12**
Apartments with no views: **0**



Champ Nº3

Shape code: **12 19 6 15 21 15 4 21 9 20 17 3 21 16 19 3 2 21 4 19 21 11 21 2 1 17 13 21 12 10 21 16 21 16 21 20 4 21 2 3 18 18 18 18 22 19 22 1 19 19 18 21 9 21 6 18 16 21 15 21 19 21 15 19**

Score: **4.861876**

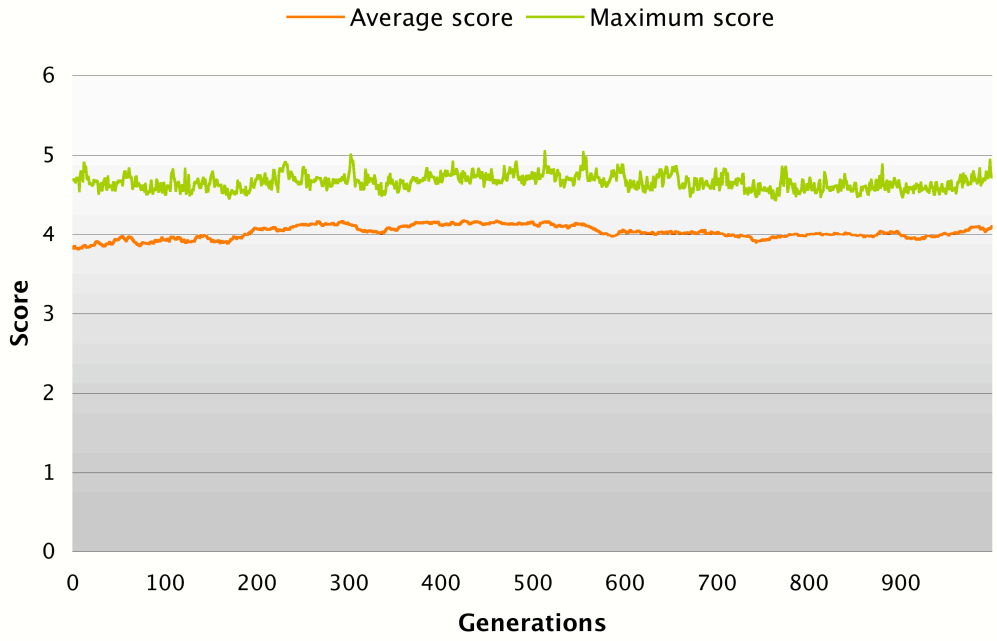
Appeared in generation: **853**

Nº of apartments: **28**
Nº of circulation blocks: **38**
Volume: **9600m³**
Total area: **2400 m²**
Apartment area: **1792 m²**
Circulation area: **608m²**
Height: **64 m**
Footprint: **36 m × 48 m**

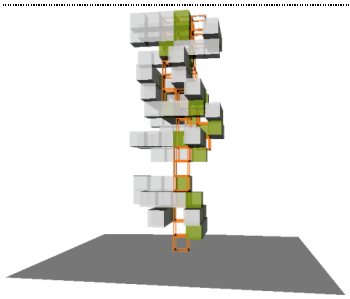
Nº of apartments with balconies: **18**
Percentage of apartments with balconies: **64.285713 %**

Apartments with views to +i: **15**
Apartments with views to −i: **11**
Apartments with views to +j: **13**
Apartments with views to −j: **14**
Apartments with no views: **0**

Multiple Criteria Population: **500** Set mutation rate: **0.05**



Actual mutation rate: **0.013** Final average score: **4.1029** Time elapsed: **27186 s**



Champ Nº1

Shape code: **16 21 6 12 21 11 7 21 7 21 21 18 15 21 4 3 7 21 21 2 11 21 4 19 14 21 6 12 21 4 21 5 21 21 11 17 7 17 22 12 19 3 18 2 18 22 3 22 18 22 14 22 11 18 22 7 17 20 4 22 16 7 22 18**

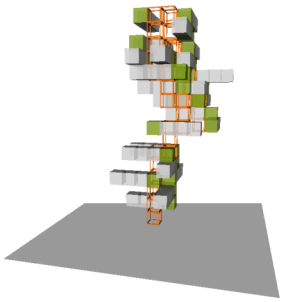
Score: **5.048235**

Appeared in generation: **515**

Nº of apartments: **30**
Nº of circulation blocks: **36**
Volume: **9984 m³**
Total area: **2496 m²**
Apartment area: **1920 m²**
Circulation area: **576 m²**
Height: **64 m**
Footprint: **24 m × 40 m**

Nº of apartments with balconies: **17**
Percentage of apartments with balconies: **56.666668 %**

Apartments with views to +i: **22**
Apartments with views to −i: **10**
Apartments with views to +j: **9**
Apartments with views to −j: **15**
Apartments with no views: **0**



Champ Nº2

Shape code: **15 21 18 4 21 8 3 21 12 21 15 11 17 21 21 13 17 21 21 20 19 21 15 21 15 11 19 21 4 14 21 10 21 21 2 17 18 8 22 12 22 18 18 8 14 22 12 18 7 22 17 1 22 12 22 8 20 2 11 17 22 11 22 8**

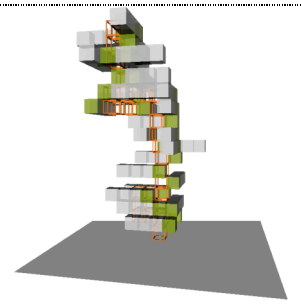
Score: **5.036162**

Appeared in generation: **557**

Nº of apartments: **28**
Nº of circulation blocks: **38**
Volume: **9600 m³**
Total area: **2400 m²**
Apartment area: **1792 m²**
Circulation area: **608 m²**
Height: **64 m**
Footprint: **36 m × 44 m**

Nº of apartments with balconies: **17**
Percentage of apartments with balconies: **60.714287 %**

Apartments with views to +i: **22**
Apartments with views to −i: **10**
Apartments with views to +j: **14**
Apartments with views to −j: **12**
Apartments with no views: **0**



Champ Nº3

Shape code: **11 18 15 19 19 21 21 11 17 15 22 17 16 3 21 21 8 21 3 21 15 21 1 4 20 21 4 14 20 21 15 21 12 21 4 19 18 19 7 22 20 8 20 19 20 10 21 5 17 21 8 5 21 20 6 20 5 8 21 8 21 18 7 4**

Score: **5.004734**

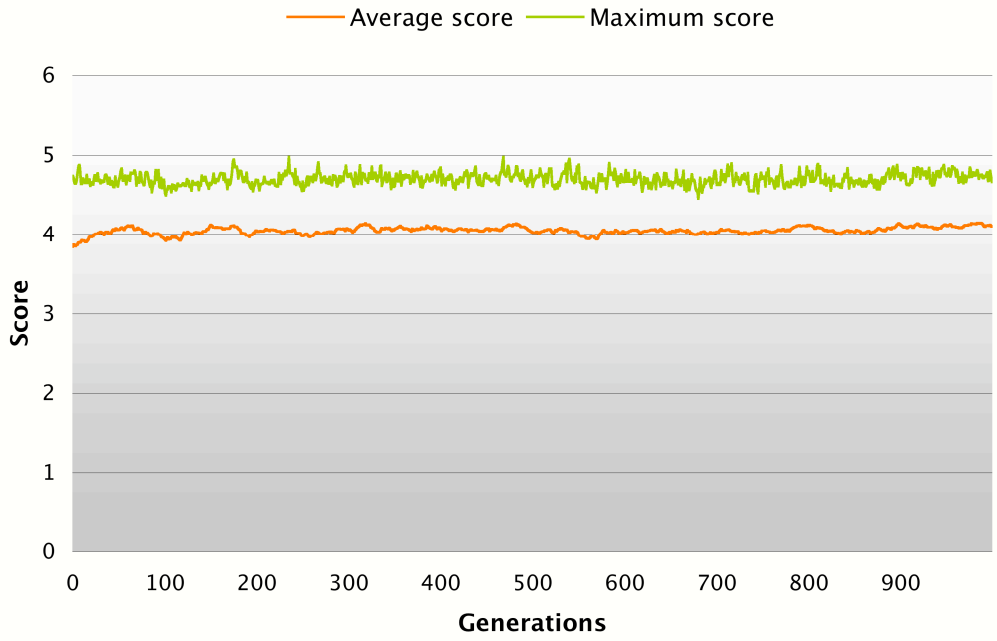
Appeared in generation: **304**

Nº of apartments: **28**
Nº of circulation blocks: **38**
Volume: 9600 **m³**
Total area: **2400 m²**
Apartment area: **1792 m²**
Circulation area: **608 m²**
Height: **64 m**
Footprint: **36 m × 48 m**

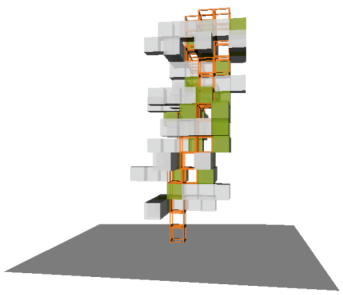
Nº of apartments with balconies: **16**
Percentage of apartments with balconies: **57.142860 %**

Apartments with views to +i: **21**
Apartments with views to −i: **6**
Apartments with views to +j: **14**
Apartments with views to −j: **13**
Apartments with no views: **0**

Multiple Criteria Population: **500** Set mutation rate: **0.1**



Actual mutation rate: **0.0259** Final average score: **4.0988** Time elapsed: **28887 s**



Champ №1

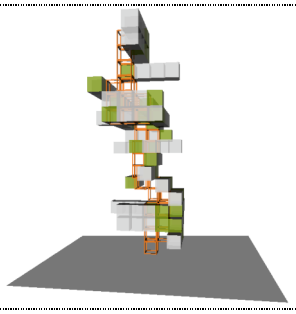
Shape code: **21 6 9 21 5 21 7 20 17 8 21 12 20 6 21 17 8 21 11 15 21 16 18 10 21 15 3 21 9 21 15 17 21 12 21 11 21 19 18 19 15 22 8 7 22 4 14 18 18 18 22 22 8 14 22 3 19 22 2 8 22 16 22 11**

Score: **4.976807**

Appeared in generation: **470**

№ of apartments: **30**
№ of circulation blocks: **36**
Volume: **9984 m³**
Total area: **2496 m²**
Apartment area: **1920 m²**
Circulation area: **576 m²**
Height: **60 m**
Footprint: **32 m × 48 m**

№ of apartments with balconies: **21**
Percentage of apartments with balconies: **70 %**
Apartments with views to +i: **18**
Apartments with views to −i: **12**
Apartments with views to +j: **11**
Apartments with views to −j: **15**
Apartments with no views: **0**



Champ №2

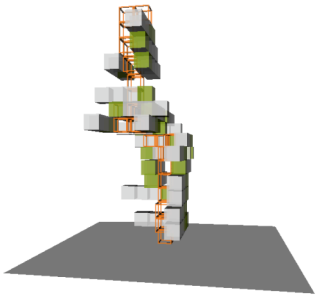
Shape code: **17 12 21 16 19 11 21 7 17 8 3 21 18 12 18 19 21 4 19 2 20 21 20 4 21 8 21 17 13 18 7 21 15 19 21 8 6 21 20 20 20 10 8 20 20 17 11 22 7 8 19 19 5 21 21 10 17 20 21 1 21 4 21 4**

Score: **4.975957**

Appeared in generation: **237**

№ of apartments: **26**
№ of circulation blocks: **40**
Volume: **9216 m³**
Total area: **2304 m²**
Apartment area: **1664 m²**
Circulation area: **640 m²**
Height: **64 m**
Footprint: **24 m × 56 m**

№ of apartments with balconies: **15**
Percentage of apartments with balconies: **57.692307 %**
Apartments with views to +i: **16**
Apartments with views to −i: **8**
Apartments with views to +j: **11**
Apartments with views to −j:12
Apartments with no views: **1**



Champ №3

Shape code: **15 20 12 21 12 18 21 12 7 21 2 12 21 10 8 21 4 18 18 21 18 18 7 4 22 15 19 13 21 19 21 9 11 21 8 2 20 6 20 20 20 22 20 12 19 6 21 17 13 8 21 11 21 21 14 20 16 21 20 16 21 12 21 17**

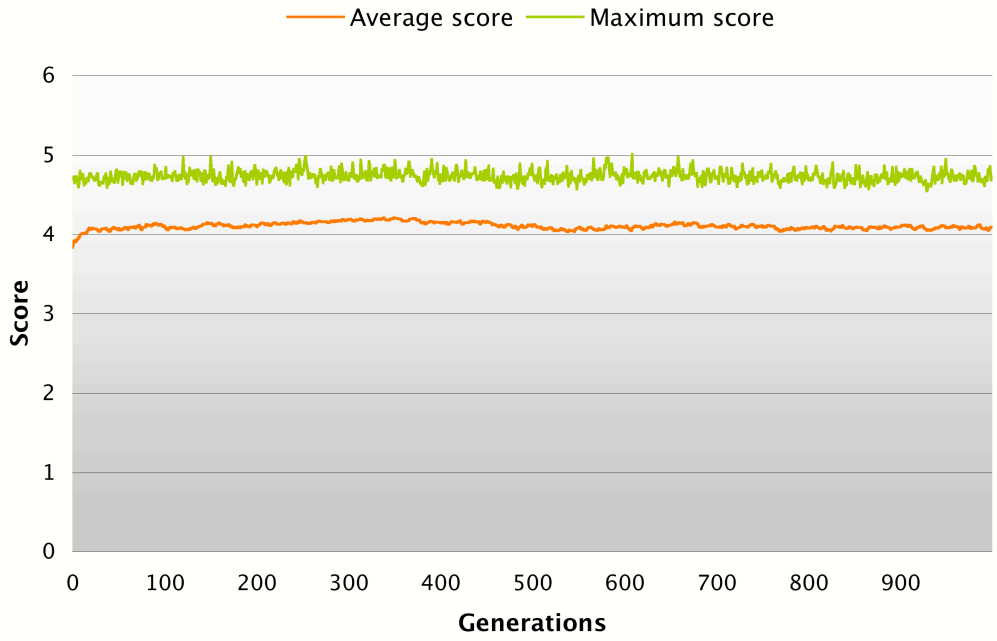
Score: **4.950448**

Appeared in generation: **542**

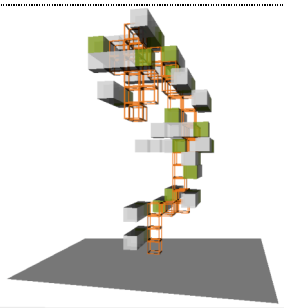
№ of apartments: **28**
№ of circulation blocks: **38**
Volume: **9600 m³**
Total area: **2400 m²**
Apartment area: **1792 m²**
Circulation area: **608 m²**
Height: **64 m**
Footprint: **24 m × 52 m**

№ of apartments with balconies: **12**
Percentage of apartments with balconies: **42.857143 %**
Apartments with views to +i: **19**
Apartments with views to −i: **11**
Apartments with views to +j: **8**
Apartments with views to −j: **12**
Apartments with no views: **0**

Multiple Criteria Population: **500** Set mutation rate: **0.5**



Actual mutation rate: **0.1247** Final average score: **4.0945** Time elapsed: **34515 s**



Champ Nº1

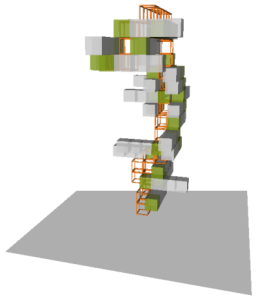
Shape code: **6 21 21 6 17 18 18 2 17 21 2 20 4 20 21 21 16 21 18 4 21 3 17 4 15 21 14 20 20 20 11 21 10 21 12 19 21 20 20 12 21 19 15 21 8 19 20 21 15 20 12 22 19 10 22 17 11 22 17 22 19 14 22 18**

Score: **5.011373**

Appeared in generation: **610**

Nº of apartments: **22**
Nº of circulation blocks: **44**
Volume: **8448 m³**
Total area: **2112 m²**
Apartment area: **1408 m²**
Circulation area: **704 m²**
Height: **64 m**
Footprint: **28 m × 56 m**

Nº of apartments with balconies: **12**
Percentage of apartments with balconies: **54.545456 %**
Apartments with views to +i: **17**
Apartments with views to −i: **10**
Apartments with views to +j: **8**
Apartments with views to −j: **12**
Apartments with no views: **0**



Champ Nº2

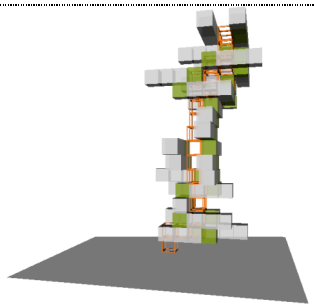
Shape code: **16 18 18 21 5 8 19 21 17 16 21 7 17 21 12 3 21 4 21 18 16 21 18 12 14 21 3 8 2 21 4 14 18 7 21 20 20 4 2 21 7 21 21 2 20 21 20 20 20 20 20 22 20 8 13 19 22 6 16 22 19 5 18 10**

Score: **4.982745**

Appeared in generation: **255**

Nº of apartments: **27**
Nº of circulation blocks: **39**
Volume: **9408 m³**
Total area: **2352 m²**
Apartment area: **1728 m²**
Circulation area: **624 m²**
Height: **64 m**
Footprint: **28 m × 60 m**

Nº of apartments with balconies: **15**
Percentage of apartments with balconies: **55.555557 %**
Apartments with views to +i: **16**
Apartments with views to −i: **13**
Apartments with views to +j: **12**
Apartments with views to −j: **10**
Apartments with no views: **0**



Champ Nº3

Shape code: **17 17 11 1 21 15 13 21 18 14 21 15 13 21 7 20 12 20 21 12 14 21 8 6 21 4 18 21 4 21 8 21 13 21 3 17 21 18 17 15 20 13 21 21 6 4 18 18 16 22 3 18 16 22 8 22 19 11 15 22 4 18 15 19**

Score: **4.974902**

Appeared in generation: **660**

Nº of apartments: **30**
Nº of circulation blocks: **36**
Volume: **9984 m³**
Total area: **2496 m²**
Apartment area: **1920 m²**
Circulation area: **576 m²**
Height: **64 m**
Footprint: **32 m × 44 m**

Nº of apartments with balconies: **16**
Percentage of apartments with balconies: **53.333336 %**
Apartments with views to +i: **22**
Apartments with views to −i: **9**
Apartments with views to +j: **10**
Apartments with views to −j: **14**
Apartments with no views: **0**

Appendix E
Shape Evolution Source Code

The C++ source code for the Shape Evolution prototype is included here for reference. This is working code and includes redundancies and debugging routines used to facilitate the development of the programme. The source code has been colour-coded to aid comprehensibility: comments are rendered in grey, strings are lime green, numbers are blue, keywords are orange, and preprocessor commands are pink.

```
#include <iostream.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>

// attempt to load configuration from file (only works after recompile)
// #include "config.cfg"

//#include "arrays.cpp"

// rounding up or down command

#define round(x) ((int)((x)<0?ceil((x)-0.5):floor((x)+0.5)))

// Array classes


// Single dimensional array

template <class T>
class array1D
{
protected:
    T * buffer;
    unsigned int s1;

public:
    array1D (const unsigned int num_elements) {
        assert (num_elements > 0);
        s1 = num_elements;
        buffer = new T [num_elements];
    }

    ~array1D () { delete buffer; }

    inline const T & get (const unsigned int i1) {
        assert (i1 < s1);
        return buffer [i1];
    }

    inline void set (const unsigned int i1, const T & value) {
        assert (i1 < s1);
        buffer [i1] = value;
    }
};


// 2D array

// Modus operandi for N-dimensional arrays: allocate a flat (1D) array, then
// convert indices (i1, i2, ..., iN) from N-tuples to an 1D array 'flat'
index
// i. For the trivial case, N=1, the index and flat index coincide, so i=i1.
//
// For the first non-trivial case, N=2, given an array of s1 x s2 elements,
// location (i1,i2) is mapped to flat index i = i1 + s1 i2. Intuitively,
// rows/columns are converted to left-to-right-then-top-to-bottom cell
numbers.
//
// For the general case, given an array of s1 x s2 x ... x sN elements, the
// mapping between elements and flat indices is defined by a recurrence
// relation, each iteration of which augments the space by one dimension.
//
// f (1, (i1),    (s1))    = i1
// f (2, (i1,i2), (s1,s2)) = i1 + s1 * i2 = f (1, (i1), (s1)) + s1 * i2
// ...
```

```
// f (N, (i1,i2,...,iN), (s1,s2,...,sN) =
//                          = f (N-1, (i1,i2,...,i{N-1}), (s1,s2,...,s{N-1}))
+
//                            + s1 s2 ... s{N-1} iN


// 2D: i = i1 + s1 * i2;

template <class T>
class array2D
{
 public:
    T * buffer;
    unsigned int s1, s2;

    array2D (const unsigned int n1,
         const unsigned int n2) {
        assert (n1 > 0);
        assert (n2 > 0);
        s1 = n1;
        s2 = n2;
        buffer = new T [n1 * n2];
    }

    ~array2D () { delete buffer; }

    inline const T & get (const unsigned int i1,
                  const unsigned int i2) {
        assert (i1 < s1);
        assert (i2 < s2);
        return buffer [i1 + s1 * i2];
    }

    inline void set (const unsigned int i1,
              const unsigned int i2,
              const T & value) {
        assert (i1 < s1);
        assert (i2 < s2);
        buffer [i1 + s1 * i2] = value;
    }
};


// 3D: i = i1  +  s1 * i2  +  s1 * s2 * i3;

template <class T>
class array3D
{
 public:
    T * buffer;
    unsigned int s1, s2, s3;

    array3D (const unsigned int n1,
           const unsigned int n2,
           const unsigned int n3) {
        assert (n1 > 0);
        assert (n2 > 0);
        assert (n3 > 0);
        s1 = n1;
        s2 = n2;
        s3 = n3;
        buffer = new T [n1 * n2 * n3];
    }

    ~array3D () { delete buffer; }

    inline const T & get (const unsigned int i1,
                  const unsigned int i2,
```

```
                const unsigned int i3) {
        assert (i1 < s1);
        assert (i2 < s2);
        assert (i3 < s3);
        return buffer [i1  +  s1 * i2  +  s1 * s2 * i3];
    }

    inline void set (const unsigned int i1,
             const unsigned int i2,
             const unsigned int i3,
             const T & value) {
        assert (i1 < s1);
        assert (i2 < s2);
        assert (i3 < s3);
        buffer [i1 +  s1 * i2  +  s1 * s2 * i3 ] = value;
    }
};


// 4D: i = i1  +  s1 * i2  +  s1 * s2 * i3  +  s1 * s2 * s3 * i4;

template <class T>
class array4D
{
public:
    T * buffer;
    unsigned int s1, s2, s3, s4;

    array4D (const unsigned int n1,
         const unsigned int n2,
         const unsigned int n3,
         const unsigned int n4) {
        assert (n1 > 0);
        assert (n2 > 0);
        assert (n3 > 0);
        assert (n4 > 0);
        s1 = n1;
        s2 = n2;
        s3 = n3;
        s4 = n4;
        buffer = new T [n1 * n2 * n3 * n4];
    }

    ~array4D () { delete buffer; }

    inline const T & get (const unsigned int i1,
                  const unsigned int i2,
                  const unsigned int i3,
                  const unsigned int i4) {
        assert (i1 < s1);
        assert (i2 < s2);
        assert (i3 < s3);
        assert (i4 < s4);
        return buffer [i1  +
                   s1 * i2  +
                   s1 * s2 * i3  +
                   s1 * s2 * s3 * i4];
    }

    inline void set (const unsigned int i1,
             const unsigned int i2,
             const unsigned int i3,
             const unsigned int i4,
             const T & value) {
        assert (i1 < s1);
        assert (i2 < s2);
        assert (i3 < s3);
```

```
        assert (i4 < s4);
        buffer [i1 +
            s1 * i2  +
            s1 * s2 * i3 +
            s1 * s2 * s3 * i4] = value;
    }
};
```

```
// GA-related globals
//int population=100;
//int *indivArray;
//int *indivShapeCode;


//globals initialised with values from file
int iterations;
int population;
int generations;
float sufficientscore;
float mutationrate;

int deltatee;

int *shapeCode;
float *indivScore;

float *avScoreMonitor;
float *bestScoreMonitor;

int *sorted;


// variables for keeping champion information
int champPool; // number of champions
array2D<int> * champCode;
array4D<int> * champArray;
// int champArray[16][16][16];
// float champScore=0;
float *champScore;
int *champGen;



int *intermediate;

int *xoverStack;
int *xshuffle;

float maxScore;

float avScore;
float *fitness;
```

```
int mutacounter=0;


// int *indivArray;
// int *indivShapeCode;

array2D<int> * indivShapeCode;
//array2D<int> * xoverShapeCode;
array4D<int> * indivArray;

// initialising globals
int i,j,k;
int array[16][16][16];
int markeri,markerj,markerk;

int a,b;
int selectRule;
int flatRule[4][3];
int circRule[3];

//iterations is the number of rules to apply
//int iterations=12;
int iter;

//set size of shapeCode equal to iterations
//int shapeCode[32];

int stack[22];
int stackmark;

//basic evaluation variables
int volume;
int circarea;
int totalarea;
int flatarea;
int flatno;
int circno;

//balcony variables
int balcno;

//views variables
int foo;
int viewsip=0;
int viewsim=0;
int viewsjp=0;
int viewsjm=0;
int noviews=0;

// evalextents variables
int kmaxextent=-1;
int iminextent=-1;
int imaxextent=-1;
int jminextent=-1;
int jmaxextent=-1;

// scoring variables

//int goalcircno;
float goalflatno;
float goalbalcno;
int goalheight;
int goalfooti;
int goalfootj;
float goalviewsip;
float goalviewsjp;
float goalviewsim;
float goalviewsjm;
```

```
float goalnoviews;

float wtcircno;
float wtflatno;
float wtbalcno;
float wtheight;
float wtfooti;
float wtfootj;
float wtviewsip;
float wtviewsjp;
float wtviewsim;
float wtviewsjm;
float wtnoviews;

// intialising rule matrices
int rule[22][4][3] = { {
    {0,-1,0},
    {1,-1,0},
    {2,-1,0},
    {3,-1,0}
},
{
    {-1,0,0},
    {-1,-1,0},
    {-1,-2,0},
    {-1,-3,0}
},
{
    {0,1,0},
    {-1,1,0},
    {-2,1,0},
    {-3,1,0}
},
{
    {1,0,0},
    {1,1,0},
    {1,2,0},
    {1,3,0}
},
{
    {0,1,0},
    {1,1,0},
    {2,1,0},
    {3,1,0}
},
{
    {-1,0,0},
    {-1,1,0},
    {-1,2,0},
    {-1,3,0}
},
{
    {0,-1,0},
    {-1,-1,0},
    {-2,-1,0},
    {-3,-1,0}
},
{
    {1,0,0},
    {1,-1,0},
    {1,-2,0},
    {1,-3,0}
},
{
    {-1,-1,0},
    {0,-1,0},
    {1,-1,0},
    {2,-1,0}
```

```
},
{
    {-1,1,0},
    {-1,0,0},
    {-1,-1,0},
    {-1,-2,0}
},
{
    {1,1,0},
    {0,1,0},
    {-1,1,0},
    {-2,1,0}
},
{
    {1,-1,0},
    {1,0,0},
    {1,1,0},
    {1,2,0}
},
{
    {-1,1,0},
    {0,1,0},
    {1,1,0},
    {2,1,0}
},
{
    {-1,-1,0},
    {-1,0,0},
    {-1,1,0},
    {-1,2,0}
},
{
    {1,-1,0},
    {0,-1,0},
    {-1,-1,0},
    {-2,-1,0}
},
{
    {1,1,0},
    {1,0,0},
    {1,-1,0},
    {1,-2,0}
},
{
    {1,0,0},
    {0,0,0},
    {0,0,0},
    {0,0,0}
},
{
    {0,-1,0},
    {0,0,0},
    {0,0,0},
    {0,0,0}
},
{
    {-1,0,0},
    {0,0,0},
    {0,0,0},
    {0,0,0}
},
{
    {0,1,0},
    {0,0,0},
    {0,0,0},
    {0,0,0}
},
{
```

```
    {0,0,1},
    {0,0,0},
    {0,0,0},
    {0,0,0}
},
{
    {0,0,-1},
    {0,0,0},
    {0,0,0},
    {0,0,0}
} };


// initialising functions
int txtview();
int txtexport();
int vrmlexport();
int generate();

int applyflatrule();
int applycircrule();



int fileinput() {

//    int inputvalues[2];

    FILE *inputfile;

    inputfile = fopen("input.txt","r");
    if (!inputfile) { return -1; cout << "File not found.\n"; }

//    for (i=0;i<=1;i++) fscanf(inputfile, "%d", &inputvalues[i] );

    fscanf(inputfile, "%d", &population);
    fscanf(inputfile, "%d", &iterations);
    fscanf(inputfile, "%d", &generations);
    fscanf(inputfile, "%f", &sufficientscore);
    fscanf(inputfile, "%f", &mutationrate);

    fscanf(inputfile, "%d", &champPool);

    fscanf(inputfile, "%f", &goalflatno);
    fscanf(inputfile, "%f", &goalbalcno);
    fscanf(inputfile, "%d", &goalheight);
    fscanf(inputfile, "%d", &goalfooti);
    fscanf(inputfile, "%d", &goalfootj);
    fscanf(inputfile, "%f", &goalviewsip);
    fscanf(inputfile, "%f", &goalviewsim);
    fscanf(inputfile, "%f", &goalviewsjp);
    fscanf(inputfile, "%f", &goalviewsjm);
    fscanf(inputfile, "%f", &goalnoviews);

    fscanf(inputfile, "%f", &wtflatno);
    fscanf(inputfile, "%f", &wtbalcno);
    fscanf(inputfile, "%f", &wtheight);
    fscanf(inputfile, "%f", &wtfooti);
    fscanf(inputfile, "%f", &wtfootj);
    fscanf(inputfile, "%f", &wtviewsip);
    fscanf(inputfile, "%f", &wtviewsim);
    fscanf(inputfile, "%f", &wtviewsjp);
    fscanf(inputfile, "%f", &wtviewsjm);
    fscanf(inputfile, "%f", &wtnoviews);


//    iterations = inputvalues[0];
//    population = inputvalues[1];
```

```
    shapeCode = new int[iterations];

// champCode = new int[iterations];

    champScore = new float[champPool];
    champGen = new int[champPool];

    champCode = new array2D<int> (champPool, iterations);
    champArray = new array4D<int> (champPool, 16, 16, 16);


//    indivArray = (int*) new int[population][16][16][16];
//    indivShapeCode = (int*) new int[population][iterations];
    indivScore = new float[population];

    fitness = new float[population];

    sorted = new int[population];

    intermediate = new int[population];

    xoverStack = new int[iterations-1];
    xshuffle = new int[population];

    avScoreMonitor = new float[generations];
    bestScoreMonitor = new float[generations];

    indivShapeCode = new array2D<int> (population*2, iterations);
    //xoverShapeCode = new array2D<int> (population, iterations);
    indivArray = new array4D<int> (population*2, 16, 16, 16);


    return 0;

}


int sortcompare (const void *theone, const void *theother)
{
    int theonei = * (int *) theone;
    int theotheri = * (int *) theother;

    // This comparison scheme looks up the ordering map and accesses the
    // fitness_array. The side effect is that, at the end of the qsort()
call, the
    // order array contains indices that can be used to access arrays in
order of
    // *DECREASING* fitness.

        if (indivScore[theotheri] < indivScore[theonei]) return -1;
        else if (indivScore[theotheri] > indivScore[theonei]) return 1;
    else return 0;
}



int sortpop() {


    int alan;

// This initialises the sorted array with sorted[x]=x

    for (alan=0;alan<=population;alan++) {
        sorted[alan]=alan;
    }
```

```
    qsort (sorted, population, sizeof (int), sortcompare);

    return 1;

}



int clearonearray(int clearmember) {

    // initialises array 'clearmember' with zeros

        for (i=0;i<=15;i++)
            {
            for (j=0;j<=15;j++)
                {
                for (k=0;k<=15;k++)
                    {
                        indivArray->set(clearmember,i,j,k,0);
                    }
                }
            }

    return 1;

}




int embryogenesis(int genmember) {


    int itercounter;


    clearonearray(genmember);


        indivArray->set(genmember,7,7,0,1);
    indivArray->set(genmember,7,7,1,1);

        markeri=7;
        markerj=7;
        markerk=1;

        for (itercounter=0;itercounter<=iterations-1;itercounter++) {

            if ((indivShapeCode-
>get(genmember,itercounter)>=1)&&(indivShapeCode-
>get(genmember,itercounter)<=16)) {
        //apply flat rule



        // make sure the rule does not try to exceed limits

        if (
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][0][0]>15)||
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][0][1]>15)||
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][0][2]>15)||
```

```
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][1][0]>15)||
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][1][1]>15)||
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][1][2]>15)||
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][2][0]>15)||
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][2][1]>15)||
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][2][2]>15)||
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][3][0]>15)||
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][3][1]>15)||
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][3][2]>15)||
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][0][0]<0)||
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][0][1]<0)||
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][0][2]<1)||
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][1][0]<0)||
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][1][1]<0)||
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][1][2]<1)||
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][2][0]<0)||
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][2][1]<0)||
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][2][2]<1)||
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][3][0]<0)||
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][3][1]<0)||
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][3][2]<1)) {
            // cout << "Rule would exceed limits; not applied.\n";
            return -2;
            }


        // check for validity of rule application
        // make sure the rule puts new blocks in empty spaces

        if (!(
        (indivArray->get(genmember,
            markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][0][0],
            markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][0][1],
            markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][0][2])==0)&&
        (indivArray->get(genmember,
            markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][1][0],
            markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][1][1],
            markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][1][2])==0)&&
        (indivArray->get(genmember,
            markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][2][0],
```

```
            markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][2][1],
            markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][2][2])==0)&&
        (indivArray->get(genmember,
            markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][3][0],
            markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][3][1],
            markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][3][2])==0))) {
            // cout << "Rule would overlap; not applied.\n";
            return -1;
            }



        indivArray->set(genmember,
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-1][0][0]),
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-1][0][1]),
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][0][2]),2);

        indivArray->set(genmember,
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-1][1][0]),
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-1][1][1]),
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][1][2]),3);

        indivArray->set(genmember,
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-1][2][0]),
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-1][2][1]),
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][2][2]),4);

        indivArray->set(genmember,
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-1][3][0]),
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-1][3][1]),
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][3][2]),5);

        // cout << "#";

        }

            if ((indivShapeCode-
>get(genmember,itercounter)>=17)&&(indivShapeCode-
>get(genmember,itercounter)<=22)) {


        // check for validity

        if ((markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][0][0]>15)||
            (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][0][1]>15)||
            (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][0][2]>15)||
            (markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][0][0]<0)||
            (markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][0][1]<0)||
            (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][0][2]<1)) {
            // cout << "Rule would exceed limits; not applied.\n";
            return -4;
        }
```

```
        if (!(indivArray->get(genmember,
            markeri+rule[indivShapeCode->get(genmember,itercounter)-
1][0][0],
            markerj+rule[indivShapeCode->get(genmember,itercounter)-
1][0][1],
            markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][0][2])==0))
        {
            // cout << "Rule would overlap; not applied.\n";
            return -3;
        }


        //apply circ rule
        indivArray->set(genmember,
        (markeri+rule[indivShapeCode->get(genmember,itercounter)-1][0][0]),
        (markerj+rule[indivShapeCode->get(genmember,itercounter)-1][0][1]),
        (markerk+rule[indivShapeCode->get(genmember,itercounter)-
1][0][2]),1);

        //move markers
        markeri += rule[indivShapeCode->get(genmember,itercounter)-1][0][0];
        markerj += rule[indivShapeCode->get(genmember,itercounter)-1][0][1];
        markerk += rule[indivShapeCode->get(genmember,itercounter)-1][0][2];

        // cout << "@";


        }


    }


    return 1;

}



int cleararray() {

    // initialises the array with zeros

    for (i=0;i<=15;i++)
        {
        for (j=0;j<=15;j++)
            {
            for (k=0;k<=15;k++)
                {
                    array[i][j][k] = 0;
                }
            }
        }

    for (i=0;i<=(iter-1);i++) shapeCode[i] = 0;

    return 0;

}
```

```
int initstack() {

    //initialise a stack[22] of rules with a random order, then start a loop
    //that is "iterations" long that picks a rule, and the below ensues.


    int n, ntemp;

    for(n=0;n<=21;n++) stack[n]=n;

    for(n=0;n<=21;n++) {
        int pos = ( rand() >> 8) % 22;


        ntemp = stack[n];
        stack[n] = stack[pos];
        stack[pos] = ntemp;
    }

    return 0;

}



//applyflatrule applies the flatrule.

int applyflatrule()
{


    if (!(

(array[markeri+flatRule[0][0]][markerj+flatRule[0][1]][markerk+flatRule[0][2
]]==0)&&

(array[markeri+flatRule[1][0]][markerj+flatRule[1][1]][markerk+flatRule[1][2
]]==0)&&

(array[markeri+flatRule[2][0]][markerj+flatRule[2][1]][markerk+flatRule[2][2
]]==0)&&

(array[markeri+flatRule[3][0]][markerj+flatRule[3][1]][markerk+flatRule[3][2
]]==0)))
     {
         // cout << "Rule " << (selectRule + 1) << " would overlap; not
applied.\n";
         return 0;
     }



    if (
    (markeri+flatRule[0][0]>15)||
    (markerj+flatRule[0][1]>15)||
    (markerk+flatRule[0][2]>15)||
    (markeri+flatRule[1][0]>15)||
    (markerj+flatRule[1][1]>15)||
    (markerk+flatRule[1][2]>15)||
    (markeri+flatRule[2][0]>15)||
    (markerj+flatRule[2][1]>15)||
    (markerk+flatRule[2][2]>15)||
    (markeri+flatRule[3][0]>15)||
    (markerj+flatRule[3][1]>15)||
```

```
    (markerk+flatRule[3][2]>15)||
    (markeri+flatRule[0][0]<0)||
    (markerj+flatRule[0][1]<0)||
    (markerk+flatRule[0][2]<1)||
    (markeri+flatRule[1][0]<0)||
    (markerj+flatRule[1][1]<0)||
    (markerk+flatRule[1][2]<1)||
    (markeri+flatRule[2][0]<0)||
    (markerj+flatRule[2][1]<0)||
    (markerk+flatRule[2][2]<1)||
    (markeri+flatRule[3][0]<0)||
    (markerj+flatRule[3][1]<0)||
    (markerk+flatRule[3][2]<1)) {
        // cout << "Rule " << (selectRule + 1) << " would exceed limits; not
applied.\n";
        return 0;
    }



    shapeCode[iter] = selectRule + 1;

     // cout << "* Rule " << (selectRule + 1) << " applied.\n";
    //cout << "@";

     iter++;


array[markeri+flatRule[0][0]][markerj+flatRule[0][1]][markerk+flatRule[0][2]
]=2;

array[markeri+flatRule[1][0]][markerj+flatRule[1][1]][markerk+flatRule[1][2]
]=3;

array[markeri+flatRule[2][0]][markerj+flatRule[2][1]][markerk+flatRule[2][2]
]=4;

array[markeri+flatRule[3][0]][markerj+flatRule[3][1]][markerk+flatRule[3][2]
]=5;

     return 1;
}




//applycircrule applies the circrule.

int applycircrule()
{

    if
(!(array[markeri+circRule[0]][markerj+circRule[1]][markerk+circRule[2]]==0))
    {
         // cout << "Rule " << (selectRule + 1) << " would overlap; not
applied.\n";
         return 0;
    }


    if ((markeri+circRule[0]>15)||
    (markerj+circRule[1]>15)||
    (markerk+circRule[2]>15)||
    (markeri+circRule[0]<0)||
    (markerj+circRule[1]<0)||
    (markerk+circRule[2]<1)) {
```

```
        // cout << "Rule " << (selectRule + 1) << " would exceed limits; not
applied.\n";
        return 0;
    }


    shapeCode[iter]= selectRule + 1;

    // cout << "* Rule " << (selectRule + 1) << " applied.\n";
    //cout << ".";

    iter++;

    array[markeri+circRule[0]][markerj+circRule[1]][markerk+circRule[2]]=1;


    //move the marker, as a new circulation block has been placed
    markeri += circRule[0];
    markerj += circRule[1];
    markerk += circRule[2];

    initstack();
    stackmark=0;

    return 1;
}




// ok, this should be the generator function, called "generate"
// calls rules
int generate() {


    //set the size of shapeCode equal to iterations

    //shapeCode = new int[iterations];

    // these lines initialise the generation process by placing the
    // initial shape and setting the marker.



    array[7][7][0]=1;
    array[7][7][1]=1;

    markeri=7;
    markerj=7;
    markerk=1;

    //if this is uncommented rulea will not be applied.
    //array[7][6][1]=7;



    //if a rule cannot be applied, do nothing and move on to the next rule.
    //if no rules can be applied (eg. when the design has backed itself into
a corner)
    //then stop rule generation and print out a warning.


    stackmark = 0;

    iter=0;
```

```
    while (iter<=iterations-1) {

        //select a random rule
        //selectRule=(rand()%21);

        //selectRule = 16;

        //select stack[stackmark]

        if (stackmark>21) {
            // cout << "Rule stack exhausted, no rules applied.\nNo more
rules can be applied to the last circulation block.\n";
            //cout << "FAIL\n";
            return -1;
        }

        selectRule=stack[stackmark];
        stackmark++;


        // cout << "Trying rule " << (selectRule + 1) << "...\n";


        if ((selectRule>=0)&&(selectRule<=15)) {

            for (a=0;a<=3;a++)
            {
                for (b=0;b<=2;b++)
                {
                    flatRule[a][b]=rule[selectRule][a][b];
                }
            }

            applyflatrule();
        }

        if ((selectRule>=16)&&(selectRule<=21)) {


            for (a=0;a<=2;a++)
            {
                circRule[a]=rule[selectRule][0][a];
            }

            applycircrule();

        }

    }



    //cout << "END\n";

    return 0;
}




// txtexport saves a particular design as a text file
```

```
int txtexport (int member)
{
    FILE* textfile = NULL;

    textfile = fopen("design.txt","w");
    if (!textfile) return -1;

        for (k=0;k<=15;k++)
        {
            fprintf (textfile, "Level %d \n", k);

            for (i=0;i<=15;i++)
            {
                for (j=0;j<=15;j++)
                {
                    fprintf (textfile, "%d", indivArray->get(member,i,j,k)
);
                }
                fprintf (textfile, "\n");
            }
            fprintf (textfile, "\n");
        }

    fclose(textfile);

    return 1;

}


// vrmlexport saves a particular design as a VRML file

int vrmlexport (int member)
{
    FILE* vrmlfile = NULL;

    char filename[] = "foo.wrl";

    sprintf(filename, "design%d.wrl", member+1);

    vrmlfile = fopen(filename,"w");
    if (!vrmlfile) return -1;

    fprintf (vrmlfile, "#VRML V2.0 utf8\n\n");

    fprintf (vrmlfile, "DEF Helicopter Viewpoint {
  position 23.5 7.5 7.5
  orientation 0.1354 -0.9815 -0.1354 -1.589
  fieldOfView 0.8859
  description \"Helicopter\"
}
DEF Helicopter-TIMER TimeSensor { loop TRUE cycleInterval 3.333 },
DEF Helicopter-POS-INTERP PositionInterpolator {
  key [0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, 0.24, 0.27, 0.3,
      0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.51, 0.54, 0.57, 0.6, 0.63,
      0.66, 0.69, 0.72, 0.75, 0.78, 0.81, 0.84, 0.87, 0.9, 0.93, 0.96,
      0.99, 1, ]
  keyValue [23.5 7.5 7.5, 23.03 7.5 4.236, 22.16 7.5 1.328, 20.88 7.5 -
1.225,
    19.18 7.5 -3.43, 16.93 7.5 -5.364, 14.29 7.5 -6.934,
    11.5 7.5 -7.992, 8.666 7.5 -8.417, 5.563 7.5 -8.259,
    2.464 7.5 -7.587, -0.3014 7.5 -6.463, -2.666 7.5 -4.838,
    -4.793 7.5 -2.686, -6.514 7.5 -0.2056, -7.663 7.5 2.417,
    -8.251 7.5 5.379, -8.335 7.5 8.572, -7.914 7.5 11.65,
    -6.966 7.5 14.35, -5.416 7.5 16.89, -3.405 7.5 19.17,
    -1.102 7.5 20.99, 1.495 7.5 22.26, 4.52 7.5 23.09,
    7.658 7.5 23.43, 10.58 7.5 23.2, 13.37 7.5 22.35, 16.09 7.5 20.93,
    18.49 7.5 19.1, 20.36 7.5 16.99, 21.78 7.5 14.56, 22.79 7.5 11.77,
```

```
    23.39 7.5 8.627, 23.5 7.5 7.5, ] },
DEF Helicopter-ROT-INTERP OrientationInterpolator {
  key [0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, 0.24, 0.27, 0.3,
      0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.51, 0.54, 0.57, 0.6, 0.63,
      0.66, 0.69, 0.72, 0.75, 0.78, 0.81, 0.84, 0.87, 0.9, 0.93, 0.96,
      0.99, 1, ]
  keyValue [0.1354 -0.9815 -0.1354 -1.589, 0.1111 -0.9843 -0.1368 -1.793,
    0.09087 -0.9864 -0.1368 -1.982, 0.07394 -0.9879 -0.1365 -2.159,
    0.05918 -0.9889 -0.1365 -2.331, 0.04481 -0.9896 -0.1369 -2.515,
    0.03062 -0.9901 -0.137 -2.706, 0.01735 -0.9905 -0.1366 -2.891,
    0.005009 -0.9906 -0.137 -3.069, -0.008427 -0.9904 -0.1376 -3.263,
    -0.02233 -0.9903 -0.1374 -3.461, -0.03558 -0.99 -0.1366 -3.646,
    -0.04903 -0.9894 -0.1366 -3.824, -0.06424 -0.9885 -0.1367 -4.012,
    -0.08058 -0.9874 -0.1363 -4.199, -0.09789 -0.9859 -0.1361 -4.375,
    0.1194 0.9834 0.1365 -1.721, 0.1458 0.9799 0.1362 -1.523,
    0.1759 0.9751 0.1348 -1.332, 0.2109 0.9683 0.1336 -1.158,
    0.2596 0.9566 0.1322 -0.9785, 0.3278 0.9358 0.1293 -0.7977,
    0.4242 0.8971 0.1237 -0.6287, 0.5741 0.811 0.1123 -0.4734,
    0.8239 0.5614 0.07804 -0.3343, 0.9993 -0.03576 -0.004955 -0.2755,
    0.815 -0.574 -0.07917 -0.3353, 0.5834 -0.8045 -0.1112 -0.4652,
    0.4246 -0.8969 -0.1241 -0.6303, 0.3247 -0.9369 -0.1294 -0.8042,
    0.2615 -0.9561 -0.132 -0.9716, 0.2157 -0.9672 -0.134 -1.142,
    0.1785 -0.9746 -0.1355 -1.323, 0.1457 -0.98 -0.1358 -1.52,
    0.1354 -0.9815 -0.1354 -1.589, ] },
ROUTE Helicopter-TIMER.fraction_changed TO Helicopter-POS-
INTERP.set_fraction
ROUTE Helicopter-POS-INTERP.value_changed TO Helicopter.set_position
ROUTE Helicopter-TIMER.fraction_changed TO Helicopter-ROT-
INTERP.set_fraction
ROUTE Helicopter-ROT-INTERP.value_changed TO Helicopter.set_orientation
DEF Sniper Viewpoint {
  position 20 7.5 20
  orientation 0.2874 -0.9504 -0.1191 -0.822
  fieldOfView 0.8859
  description \"Sniper\"
}
DEF Pedestrian Viewpoint {
  position 21.5 0.5 7.5
  orientation -0.0879 -0.9922 0.0879 -1.579
  fieldOfView 0.8859
  description \"Pedestrian\"
}
DEF Pedestrian-TIMER TimeSensor { loop TRUE cycleInterval 3.333 },
DEF Pedestrian-POS-INTERP PositionInterpolator {
  key [0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, 0.24, 0.27, 0.3,
      0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.51, 0.54, 0.57, 0.6, 0.63,
      0.66, 0.69, 0.72, 0.75, 0.78, 0.81, 0.84, 0.87, 0.9, 0.93, 0.96,
      0.99, 1, ]
  keyValue [21.5 0.5 7.5, 21.09 0.5 4.644, 20.33 0.5 2.099, 19.21 0.5 -
0.1345,
    17.72 0.5 -2.064, 15.75 0.5 -3.756, 13.44 0.5 -5.13,
    11 0.5 -6.055, 8.52 0.5 -6.427, 5.805 0.5 -6.289, 3.093 0.5 -5.701,
    0.6738 0.5 -4.718, -1.395 0.5 -3.296, -3.256 0.5 -1.413,
    -4.763 0.5 0.7576, -5.767 0.5 3.052, -6.282 0.5 5.644,
    -6.356 0.5 8.438, -5.987 0.5 11.13, -5.158 0.5 13.49,
    -3.802 0.5 15.72, -2.042 0.5 17.71, -0.02711 0.5 19.31,
    2.245 0.5 20.42, 4.892 0.5 21.14, 7.638 0.5 21.43,
    10.19 0.5 21.24, 12.64 0.5 20.49, 15.02 0.5 19.25,
    17.12 0.5 17.65, 18.76 0.5 15.81, 20 0.5 13.68, 20.88 0.5 11.24,
    21.4 0.5 8.486, 21.5 0.5 7.5, ] },
DEF Pedestrian-ROT-INTERP OrientationInterpolator {
  key [0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, 0.24, 0.27, 0.3,
      0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.51, 0.54, 0.57, 0.6, 0.63,
      0.66, 0.69, 0.72, 0.75, 0.78, 0.81, 0.84, 0.87, 0.9, 0.93, 0.96,
      0.99, 1, ]
  keyValue [-0.0879 -0.9922 0.0879 -1.579, -0.072 -0.9935 0.0887 -1.784,
    -0.05883 -0.9943 0.0886 -1.975, -0.04783 -0.9949 0.08829 -2.153,
    -0.03825 -0.9954 0.08821 -2.327, -0.02895 -0.9957 0.08847 -2.512,
```

```
    -0.01978 -0.9959 0.08848 -2.703, -0.01121 -0.996 0.08823 -2.89,
    -0.003234 -0.9961 0.08846 -3.069, 0.005443 -0.996 0.0889 -3.263,
    0.01442 -0.996 0.08874 -3.462, 0.02298 -0.9958 0.08825 -3.649,
    0.03168 -0.9956 0.08827 -3.828, 0.04153 -0.9952 0.08835 -4.017,
    0.05213 -0.9947 0.08816 -4.205, 0.06339 -0.9941 0.0881 -4.383,
    0.07743 -0.9931 0.08855 -4.572, -0.09474 0.9916 -0.08854 -1.512,
    -0.1146 0.9895 -0.08786 -1.318, -0.138 0.9866 -0.08738 -1.141,
    -0.171 0.9814 -0.0871 -0.9574, -0.2187 0.972 -0.08624 -0.7709,
    -0.2896 0.9534 -0.08445 -0.5937, -0.4125 0.9074 -0.0807 -0.4247,
    -0.6845 0.7261 -0.06483 -0.2594, -0.9984 -0.05563 0.00495 -0.1778,
    -0.6723 -0.7374 0.06531 -0.262, -0.4208 -0.9036 0.08021 -0.4158,
    -0.2898 -0.9533 0.08475 -0.5952, -0.2164 -0.9725 0.08625 -0.7777,
    -0.1723 -0.9812 0.08699 -0.9504, -0.1412 -0.9861 0.08773 -1.124,
    -0.1164 -0.9893 0.08832 -1.309, -0.0947 -0.9916 0.08822 -1.508,
    -0.0879 -0.9922 0.0879 -1.579, ] },
ROUTE Pedestrian-TIMER.fraction_changed TO Pedestrian-POS-
INTERP.set_fraction
ROUTE Pedestrian-POS-INTERP.value_changed TO Pedestrian.set_position
ROUTE Pedestrian-TIMER.fraction_changed TO Pedestrian-ROT-
INTERP.set_fraction
ROUTE Pedestrian-ROT-INTERP.value_changed TO Pedestrian.set_orientation
DEF Architect Viewpoint {
  position 7.5 21 7.5
  orientation 1 0 0 -1.571
  fieldOfView 0.8859
  description \"Architect\"
}
\n");


    fprintf (vrmlfile, "NavigationInfo { type \"EXAMINE\" }\n\n");

    fprintf (vrmlfile, "Background { skyColor [ 1 1 1 ] }\n\n");

    fprintf (vrmlfile, "Transform { translation 7.5 -0.6 7.5 children [
Shape { appearance Appearance { material Material { diffuseColor 0.8 0.8 0.8
emissiveColor 0.8 0.8 0.8} } geometry Box { size 16 0.2 16 } } ] }\n\n");

    fprintf (vrmlfile, "# Shape definitions\n\n");

    fprintf (vrmlfile, "PROTO circ [] { Shape { appearance Appearance{
material Material { diffuseColor 1 0.4 0 transparency 0.2 } } geometry
IndexedFaceSet { ccw TRUE solid TRUE convex FALSE\n");

    fprintf (vrmlfile, "coord Coordinate { point [ -0.425 -0.425 0.425, -
0.425 0.425 0.425, -0.425 -0.425 0.5, -0.425 0.425 0.5, 0.425 -0.425 0.425,
0.425 -0.425 0.5, 0.425 0.425 0.425, 0.425 0.425 0.5, 0.5 -0.425 0.425, 0.5
0.425 0.425, -0.425 0.5 0.425, 0.425 0.5 0.425, -0.5 0.425 0.425, -0.5 -
0.425 0.425, 0.425 -0.5 0.425, -0.425 -0.5 0.425, -0.5 -0.425 -0.425, -0.425
-0.425 -0.425, 0.425 -0.5 -0.425, 0.425 -0.425 -0.425, -0.425 -0.5 -0.425,
0.425 0.5 -0.425, 0.425 0.425 -0.425, 0.5 -0.425 -0.425, 0.5 0.425 -0.425,
0.425 -0.425 -0.5, -0.425 -0.425 -0.5, 0.425 0.425 -0.5, -0.5 0.425 -0.425,
-0.425 0.425 -0.425, -0.425 0.5 -0.425, -0.425 0.425 -0.5, -0.5 0.5 0.5, -
0.5 0.5 -0.5, 0.5 0.5 0.5, 0.5 0.5 -0.5, -0.5 -0.5 0.5, 0.5 -0.5 0.5, 0.5 -
0.5 -0.5, -0.5 -0.5 -0.5 ] }\n");
```

```
    fprintf (vrmlfile, "coordIndex [ 0, 1, 3, 2, -1, 4, 0, 2, 5, -1, 1, 6,
7, 3, -1, 6, 4, 5, 7, -1, 4, 6, 9, 8, -1, 1, 10, 11, 6, -1, 1, 0, 13, 12, -
1, 4, 14, 15, 0, -1, 13, 0, 17, 16, -1, 14, 4, 19, 18, -1, 18, 19, 17, 20, -
1, 21, 22, 6, 11, -1, 23, 24, 22, 19, -1, 23, 19, 4, 8, -1, 25, 26, 17, 19,
-1, 27, 25, 19, 22, -1, 28, 16, 17, 29, -1, 28, 29, 1, 12, -1, 20, 17, 0,
15, -1, 10, 1, 29, 30, -1, 9, 6, 22, 24, -1, 30, 29, 22, 21, -1, 31, 27, 22,
29, -1, 26, 31, 29, 17, -1, 30, 32, 34, 10, -1, 32, 30, 21, 33, -1, 11, 34,
35, 21, -1, 11, 10, 34, -1, 35, 33, 21, -1, 15, 36, 39, 20, -1, 15, 14, 37,
36, -1, 20, 38, 37, 18, -1, 18, 37, 14, -1, 39, 38, 20, -1, 2, 36, 37, 5, -
1, 2, 3, 32, 36, -1, 7, 34, 32, 3, -1, 7, 5, 37, 34, -1, 13, 16, 39, 36, -1,
13, 36, 32, 12, -1, 12, 32, 33, 28, -1, 16, 28, 33, 39, -1, 31, 33, 35, 27,
-1, 31, 26, 39, 33, -1, 25, 38, 39, 26, -1, 25, 27, 35, 38, -1, 23, 8, 37,
38, -1, 23, 38, 35, 24, -1, 24, 35, 34, 9, -1, 8, 9, 34, 37, -1] } }
}\n\n");

    fprintf (vrmlfile, "PROTO flat [] { Shape { appearance Appearance{
material Material { diffuseColor 0.8 0.8 0.8 transparency 0.2 } } geometry
Box { size 1.0 1.0 1.0 } } }\n\n");

    fprintf (vrmlfile, "PROTO face [] { Shape { appearance Appearance{
material Material { diffuseColor 0.4 0.6 0 transparency 0.2 } } geometry Box
{ size 1.0 1.0 1.0 } } }\n\n");

    fprintf (vrmlfile, "# Design follows.\n\n");

        for (k=0;k<=15;k++)
        {
            for (i=0;i<=15;i++)
            {
                for (j=0;j<=15;j++)
                {


                    if (indivArray->get(member,i,j,k)==1) {
                        fprintf (vrmlfile, "Transform { translation %d %d %d
children [ circ {} ] }\n\n", i, k, j);
                    }


                    if (indivArray->get(member,i,j,k)==2) {
                        fprintf (vrmlfile, "Transform { translation %d %d %d
children [ face {} ] }\n\n", i, k, j);
                    }


                    if ((indivArray->get(member,i,j,k)==3)||(indivArray-
>get(member,i,j,k)==4)||(indivArray->get(member,i,j,k)==5)) {
                        fprintf (vrmlfile, "Transform { translation %d %d %d
children [ flat {} ] }\n\n", i, k, j);
                    }


                }
            }
        }

    fclose(vrmlfile);

    return 1;

}



int vrmlchamp (int champmember) {

    FILE* vrmlfile = NULL;
```

```
    char filename[] = "foochamp.wrl";

    sprintf(filename, "champ%d.wrl", champmember+1);


    vrmlfile = fopen(filename,"w");
    if (!vrmlfile) return -1;

    fprintf (vrmlfile, "#VRML V2.0 utf8\n\n");

    fprintf (vrmlfile, "DEF Helicopter Viewpoint {
  position 23.5 7.5 7.5
  orientation 0.1354 -0.9815 -0.1354 -1.589
  fieldOfView 0.8859
  description \"Helicopter\"
}
DEF Helicopter-TIMER TimeSensor { loop TRUE cycleInterval 3.333 },
DEF Helicopter-POS-INTERP PositionInterpolator {
  key [0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, 0.24, 0.27, 0.3,
      0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.51, 0.54, 0.57, 0.6, 0.63,
      0.66, 0.69, 0.72, 0.75, 0.78, 0.81, 0.84, 0.87, 0.9, 0.93, 0.96,
      0.99, 1, ]
  keyValue [23.5 7.5 7.5, 23.03 7.5 4.236, 22.16 7.5 1.328, 20.88 7.5 -
1.225,
    19.18 7.5 -3.43, 16.93 7.5 -5.364, 14.29 7.5 -6.934,
    11.5 7.5 -7.992, 8.666 7.5 -8.417, 5.563 7.5 -8.259,
    2.464 7.5 -7.587, -0.3014 7.5 -6.463, -2.666 7.5 -4.838,
    -4.793 7.5 -2.686, -6.514 7.5 -0.2056, -7.663 7.5 2.417,
    -8.251 7.5 5.379, -8.335 7.5 8.572, -7.914 7.5 11.65,
    -6.966 7.5 14.35, -5.416 7.5 16.89, -3.405 7.5 19.17,
    -1.102 7.5 20.99, 1.495 7.5 22.26, 4.52 7.5 23.09,
    7.658 7.5 23.43, 10.58 7.5 23.2, 13.37 7.5 22.35, 16.09 7.5 20.93,
    18.49 7.5 19.1, 20.36 7.5 16.99, 21.78 7.5 14.56, 22.79 7.5 11.77,
    23.39 7.5 8.627, 23.5 7.5 7.5, ] },
DEF Helicopter-ROT-INTERP OrientationInterpolator {
  key [0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, 0.24, 0.27, 0.3,
      0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.51, 0.54, 0.57, 0.6, 0.63,
      0.66, 0.69, 0.72, 0.75, 0.78, 0.81, 0.84, 0.87, 0.9, 0.93, 0.96,
      0.99, 1, ]
  keyValue [0.1354 -0.9815 -0.1354 -1.589, 0.1111 -0.9843 -0.1368 -1.793,
    0.09087 -0.9864 -0.1368 -1.982, 0.07394 -0.9879 -0.1365 -2.159,
    0.05918 -0.9889 -0.1365 -2.331, 0.04481 -0.9896 -0.1369 -2.515,
    0.03062 -0.9901 -0.137 -2.706, 0.01735 -0.9905 -0.1366 -2.891,
    0.005009 -0.9906 -0.137 -3.069, -0.008427 -0.9904 -0.1376 -3.263,
    -0.02233 -0.9903 -0.1374 -3.461, -0.03558 -0.99 -0.1366 -3.646,
    -0.04903 -0.9894 -0.1366 -3.824, -0.06424 -0.9885 -0.1367 -4.012,
    -0.08058 -0.9874 -0.1363 -4.199, -0.09789 -0.9859 -0.1361 -4.375,
    0.1194 0.9834 0.1365 -1.721, 0.1458 0.9799 0.1362 -1.523,
    0.1759 0.9751 0.1348 -1.332, 0.2109 0.9683 0.1336 -1.158,
    0.2596 0.9566 0.1322 -0.9785, 0.3278 0.9358 0.1293 -0.7977,
    0.4242 0.8971 0.1237 -0.6287, 0.5741 0.811 0.1123 -0.4734,
    0.8239 0.5614 0.07804 -0.3343, 0.9993 -0.03576 -0.004955 -0.2755,
    0.815 -0.574 -0.07917 -0.3353, 0.5834 -0.8045 -0.1112 -0.4652,
    0.4246 -0.8969 -0.1241 -0.6303, 0.3247 -0.9369 -0.1294 -0.8042,
    0.2615 -0.9561 -0.132 -0.9716, 0.2157 -0.9672 -0.134 -1.142,
    0.1785 -0.9746 -0.1355 -1.323, 0.1457 -0.98 -0.1358 -1.52,
    0.1354 -0.9815 -0.1354 -1.589, ] },
ROUTE Helicopter-TIMER.fraction_changed TO Helicopter-POS-
INTERP.set_fraction
ROUTE Helicopter-POS-INTERP.value_changed TO Helicopter.set_position
ROUTE Helicopter-TIMER.fraction_changed TO Helicopter-ROT-
INTERP.set_fraction
ROUTE Helicopter-ROT-INTERP.value_changed TO Helicopter.set_orientation
DEF Sniper Viewpoint {
  position 20 7.5 20
  orientation 0.2874 -0.9504 -0.1191 -0.822
  fieldOfView 0.8859
  description \"Sniper\"
```

```
}
DEF Pedestrian Viewpoint {
  position 21.5 0.5 7.5
  orientation -0.0879 -0.9922 0.0879 -1.579
  fieldOfView 0.8859
  description \"Pedestrian\"
}
DEF Pedestrian-TIMER TimeSensor { loop TRUE cycleInterval 3.333 },
DEF Pedestrian-POS-INTERP PositionInterpolator {
  key [0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, 0.24, 0.27, 0.3,
      0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.51, 0.54, 0.57, 0.6, 0.63,
      0.66, 0.69, 0.72, 0.75, 0.78, 0.81, 0.84, 0.87, 0.9, 0.93, 0.96,
      0.99, 1, ]
  keyValue [21.5 0.5 7.5, 21.09 0.5 4.644, 20.33 0.5 2.099, 19.21 0.5 -
0.1345,
    17.72 0.5 -2.064, 15.75 0.5 -3.756, 13.44 0.5 -5.13,
    11 0.5 -6.055, 8.52 0.5 -6.427, 5.805 0.5 -6.289, 3.093 0.5 -5.701,
    0.6738 0.5 -4.718, -1.395 0.5 -3.296, -3.256 0.5 -1.413,
    -4.763 0.5 0.7576, -5.767 0.5 3.052, -6.282 0.5 5.644,
    -6.356 0.5 8.438, -5.987 0.5 11.13, -5.158 0.5 13.49,
    -3.802 0.5 15.72, -2.042 0.5 17.71, -0.02711 0.5 19.31,
    2.245 0.5 20.42, 4.892 0.5 21.14, 7.638 0.5 21.43,
    10.19 0.5 21.24, 12.64 0.5 20.49, 15.02 0.5 19.25,
    17.12 0.5 17.65, 18.76 0.5 15.81, 20 0.5 13.68, 20.88 0.5 11.24,
    21.4 0.5 8.486, 21.5 0.5 7.5, ] },
DEF Pedestrian-ROT-INTERP OrientationInterpolator {
  key [0, 0.03, 0.06, 0.09, 0.12, 0.15, 0.18, 0.21, 0.24, 0.27, 0.3,
      0.33, 0.36, 0.39, 0.42, 0.45, 0.48, 0.51, 0.54, 0.57, 0.6, 0.63,
      0.66, 0.69, 0.72, 0.75, 0.78, 0.81, 0.84, 0.87, 0.9, 0.93, 0.96,
      0.99, 1, ]
  keyValue [-0.0879 -0.9922 0.0879 -1.579, -0.072 -0.9935 0.0887 -1.784,
    -0.05883 -0.9943 0.0886 -1.975, -0.04783 -0.9949 0.08829 -2.153,
    -0.03825 -0.9954 0.08821 -2.327, -0.02895 -0.9957 0.08847 -2.512,
    -0.01978 -0.9959 0.08848 -2.703, -0.01121 -0.996 0.08823 -2.89,
    -0.003234 -0.9961 0.08846 -3.069, 0.005443 -0.996 0.0889 -3.263,
    0.01442 -0.996 0.08874 -3.462, 0.02298 -0.9958 0.08825 -3.649,
    0.03168 -0.9956 0.08827 -3.828, 0.04153 -0.9952 0.08835 -4.017,
    0.05213 -0.9947 0.08816 -4.205, 0.06339 -0.9941 0.0881 -4.383,
    0.07743 -0.9931 0.08855 -4.572, -0.09474 0.9916 -0.08854 -1.512,
    -0.1146 0.9895 -0.08786 -1.318, -0.138 0.9866 -0.08738 -1.141,
    -0.171 0.9814 -0.0871 -0.9574, -0.2187 0.972 -0.08624 -0.7709,
    -0.2896 0.9534 -0.08445 -0.5937, -0.4125 0.9074 -0.0807 -0.4247,
    -0.6845 0.7261 -0.06483 -0.2594, -0.9984 -0.05563 0.00495 -0.1778,
    -0.6723 -0.7374 0.06531 -0.262, -0.4208 -0.9036 0.08021 -0.4158,
    -0.2898 -0.9533 0.08475 -0.5952, -0.2164 -0.9725 0.08625 -0.7777,
    -0.1723 -0.9812 0.08699 -0.9504, -0.1412 -0.9861 0.08773 -1.124,
    -0.1164 -0.9893 0.08832 -1.309, -0.0947 -0.9916 0.08822 -1.508,
    -0.0879 -0.9922 0.0879 -1.579, ] },
ROUTE Pedestrian-TIMER.fraction_changed TO Pedestrian-POS-
INTERP.set_fraction
ROUTE Pedestrian-POS-INTERP.value_changed TO Pedestrian.set_position
ROUTE Pedestrian-TIMER.fraction_changed TO Pedestrian-ROT-
INTERP.set_fraction
ROUTE Pedestrian-ROT-INTERP.value_changed TO Pedestrian.set_orientation
DEF Architect Viewpoint {
  position 7.5 21 7.5
  orientation 1 0 0 -1.571
  fieldOfView 0.8859
  description \"Architect\"
}
\n");


    fprintf (vrmlfile, "NavigationInfo { type \"EXAMINE\" }\n\n");

    fprintf (vrmlfile, "Background { skyColor [ 1 1 1 ] }\n\n");
```

```
    fprintf (vrmlfile, "Transform { translation 7.5 -0.6 7.5 children [
Shape { appearance Appearance { material Material { diffuseColor 0.8 0.8 0.8
emissiveColor 0.4 0.4 0.4 } } geometry Box { size 16 0.2 16 } } ] }\n\n");

    fprintf (vrmlfile, "# Shape definitions\n\n");

    fprintf (vrmlfile, "PROTO circ [] { Shape { appearance Appearance{
material Material { diffuseColor 1 0.4 0 transparency 0.2 } } geometry
IndexedFaceSet { ccw TRUE solid TRUE convex FALSE\n");

    fprintf (vrmlfile, "coord Coordinate { point [ -0.425 -0.425 0.425, -
0.425 0.425 0.425, -0.425 -0.425 0.5, -0.425 0.425 0.5, 0.425 -0.425 0.425,
0.425 -0.425 0.5, 0.425 0.425 0.425, 0.425 0.425 0.5, 0.5 -0.425 0.425, 0.5
0.425 0.425, -0.425 0.5 0.425, 0.425 0.5 0.425, -0.5 0.425 0.425, -0.5 -
0.425 0.425, 0.425 -0.5 0.425, -0.425 -0.5 0.425, -0.5 -0.425 -0.425, -0.425
-0.425 -0.425, 0.425 -0.5 -0.425, 0.425 -0.425 -0.425, -0.425 -0.5 -0.425,
0.425 0.5 -0.425, 0.425 0.425 -0.425, 0.5 -0.425 -0.425, 0.5 0.425 -0.425,
0.425 -0.425 -0.5, -0.425 -0.425 -0.5, 0.425 0.425 -0.5, -0.5 0.425 -0.425,
-0.425 0.425 -0.425, -0.425 0.5 -0.425, -0.425 0.425 -0.5, -0.5 0.5 0.5, -
0.5 0.5 -0.5, 0.5 0.5 0.5, 0.5 0.5 -0.5, -0.5 -0.5 0.5, 0.5 -0.5 0.5, 0.5 -
0.5 -0.5, -0.5 -0.5 -0.5 ] }\n");

    fprintf (vrmlfile, "coordIndex [ 0, 1, 3, 2, -1, 4, 0, 2, 5, -1, 1, 6,
7, 3, -1, 6, 4, 5, 7, -1, 4, 6, 9, 8, -1, 1, 10, 11, 6, -1, 1, 0, 13, 12, -
1, 4, 14, 15, 0, -1, 13, 0, 17, 16, -1, 14, 4, 19, 18, -1, 18, 19, 17, 20, -
1, 21, 22, 6, 11, -1, 23, 24, 22, 19, -1, 23, 19, 4, 8, -1, 25, 26, 17, 19,
-1, 27, 25, 19, 22, -1, 28, 16, 17, 29, -1, 28, 29, 1, 12, -1, 20, 17, 0,
15, -1, 10, 1, 29, 30, -1, 9, 6, 22, 24, -1, 30, 29, 22, 21, -1, 31, 27, 22,
29, -1, 26, 31, 29, 17, -1, 30, 32, 34, 10, -1, 32, 30, 21, 33, -1, 11, 34,
35, 21, -1, 11, 10, 34, -1, 35, 33, 21, -1, 15, 36, 39, 20, -1, 15, 14, 37,
36, -1, 20, 38, 37, 18, -1, 18, 37, 14, -1, 39, 38, 20, -1, 2, 36, 37, 5, -
1, 2, 3, 32, 36, -1, 7, 34, 32, 3, -1, 7, 5, 37, 34, -1, 13, 16, 39, 36, -1,
13, 36, 32, 12, -1, 12, 32, 33, 28, -1, 16, 28, 33, 39, -1, 31, 33, 35, 27,
-1, 31, 26, 39, 33, -1, 25, 38, 39, 26, -1, 25, 27, 35, 38, -1, 23, 8, 37,
38, -1, 23, 38, 35, 24, -1, 24, 35, 34, 9, -1, 8, 9, 34, 37, -1] } }
}\n\n");

    fprintf (vrmlfile, "PROTO flat [] { Shape { appearance Appearance{
material Material { diffuseColor 0.8 0.8 0.8 transparency 0.2 } } geometry
Box { size 1.0 1.0 1.0 } } }\n\n");

    fprintf (vrmlfile, "PROTO face [] { Shape { appearance Appearance{
material Material { diffuseColor 0.4 0.6 0 transparency 0.2 } } geometry Box
{ size 1.0 1.0 1.0 } } }\n\n");

    fprintf (vrmlfile, "# Design follows.\n\n");

        for (k=0;k<=15;k++)
        {
            for (i=0;i<=15;i++)
            {
                for (j=0;j<=15;j++)
                {


                    if (champArray->get(champmember,i,j,k)==1) {
                        fprintf (vrmlfile, "Transform { translation %d %d %d
children [ circ {} ] }\n\n", i, k, j);
                    }


                    if (champArray->get(champmember,i,j,k)==2) {
                        fprintf (vrmlfile, "Transform { translation %d %d %d
children [ face {} ] }\n\n", i, k, j);
                    }
```

```
                    if ((champArray-
>get(champmember,i,j,k)==3)||(champArray-
>get(champmember,i,j,k)==4)||(champArray->get(champmember,i,j,k)==5)) {
                        fprintf (vrmlfile, "Transform { translation %d %d %d
children [ flat {} ] }\n\n", i, k, j);
                    }


                }
            }
        }

    fclose(vrmlfile);

    return 1;

}




// txtview show the design as text on the screen

int txtview (int member)
{

    cout << "\n\n   j+ -->\n\n";
    cout << "i\n+\n\n|\n|\nv\n";

        for (k=0;k<=15;k++)
        {
            cout << "\nLevel " << k << "\n";

            for (i=0;i<=15;i++)
            {
                for (j=0;j<=15;j++)
                {
                    cout << indivArray->get(member,i,j,k);
                }
                cout << "\n";
            }
            cout << "\n";
        }

    return 1;

}

// basic evaluation criteria calculation

int evalbasic(int member) {

    circno=2;
    flatno=0;

    for (i=0;i<=iterations-1;i++) {
        if ( (indivShapeCode->get(member, i)<=16) && (indivShapeCode-
>get(member, i)>=1) ) { flatno++; }
        if ( indivShapeCode->get(member, i)>=17 ) { circno++; }
    }

    return 0;

}

// evalextents figures out the furthest extremities in all directions
int evalextents(int member) {
```

```
    kmaxextent=-1;
    iminextent=-1;
    imaxextent=-1;
    jminextent=-1;
    jmaxextent=-1;


    for (k=15;k>=1;k--)
        {
        for (i=0;i<=15;i++)
            {
            for (j=0;j<=15;j++)
                {
                    if ( (indivArray->get(member,i,j,k)!=0) &&
(kmaxextent==-1) ) {
                        kmaxextent = k;
                        }
                }
            }
        }


    for (i=0;i<=15;i++)
        {
        for (j=0;j<=15;j++)
            {
            for (k=0;k<=15;k++)
                {
                        if ( (indivArray->get(member,i,j,k)!=0) &&
(iminextent==-1) ) { iminextent = i; }
                        if (indivArray->get(member,i,j,k)!=0) { imaxextent =
i; }
                }
            }
        }


    for (j=0;j<=15;j++)
        {
        for (i=0;i<=15;i++)
            {
            for (k=0;k<=15;k++)
                {
                        if ( (indivArray->get(member,i,j,k)!=0) &&
(jminextent==-1) ) { jminextent = j; }
                        if (indivArray->get(member,i,j,k)!=0) { jmaxextent =
j; }
                }
            }
        }

return 0;

}


/// Like get(), but returns zero iff any of i1, i2, i3, or i4
/// are out of range of the array.

inline int getInf (const int i1,
           const int i2,
           const int i3,
           const int i4) {

    //fprintf(stderr,"getInf(%d,%d,%d,%d)\n", i1,i2,i3,i4);

    if ((i1 < 0) || (i1 >= (signed int) indivArray->s1) ||
```

```
        (i2 < 0) || (i2 >= (signed int) indivArray->s2) ||
        (i3 < 0) || (i3 >= (signed int) indivArray->s3) ||
        (i4 < 0) || (i4 >= (signed int) indivArray->s4)) return 0;

    return indivArray->get (i1, i2, i3, i4);
    }


int evalbalconies(int member) {

    balcno = 0;

    for (i=0;i<=15;i++) {
        for (j=0;j<=15;j++) {
            for (k=2;k<=15;k++) {
                if ( (getInf(member,i,j,k)==2)
                 && ( ( (getInf(member,i+1,j,k)==0)
                    && (getInf(member,i+1,j,k-1)!=0) ) ||
                      ( (getInf(member,i-1,j,k)==0 &&
                     getInf(member,i-1,j,k-1)!=0) ) ||
                      ( (getInf(member,i,j+1,k)==0) &&
                    (getInf(member,i,j+1,k-1)!=0) )
                      || ( (getInf(member,i,j-1,k)==0)
                       && (getInf(member,i,j-1,k-1)!=0) ) ) ) {
                    balcno++; }
            }
            }
    }

return 0;

}


int evalviews(int member) {

    int counterip, counterim, counterjp, counterjm;

    viewsip=0;
    viewsim=0;
    viewsjp=0;
    viewsjm=0;
    noviews=0;

    for (i=0;i<=15;i++)
        {
        for (j=0;j<=15;j++)
            {
            for (k=1;k<=15;k++)
                {
                    if (indivArray->get(member,i,j,k)==2) {

                        counterip=0;
                        for (foo=(i+1);foo<=15;foo++) {
counterip+=indivArray->get(member,foo,j,k); }

                        counterim=0;
                        for (foo=(i-1);foo>=0;foo--) {
counterim+=indivArray->get(member,foo,j,k); }

                        counterjp=0;
                        for (foo=(j+1);foo<=15;foo++) {
counterjp+=indivArray->get(member,i,foo,k); }

                        counterjm=0;
                        for (foo=(j-1);foo>=0;foo--) {
counterjm+=indivArray->get(member,i,foo,k); }
```

```
                        if (counterip==0) { viewsip++; }
                        if (counterim==0) { viewsim++; }
                        if (counterjp==0) { viewsjp++; }
                        if (counterjm==0) { viewsjm++; }
                        if ( counterip && counterim && counterjp &&
counterjm ) { noviews++; }

                    }
                }
            }
        }

return 0;

}


int viewshapecode(int member) {

    cout << "\n\nThe shape code for design No. " << member+1 << " is:\n";
    for (i=0;i<=iterations-1;i++) cout << indivShapeCode->get(member, i) <<
" ";
    cout << "\n";

    return 0;

}

int viewshapecodes() {

    int n,q;

    cout << "\n";

    for (n=1;n<=population;n++)
        {
            cout << "Design No. " << n << ":\n";
            for (q=0;q<=iterations-1;q++) { cout << indivShapeCode->get(n-1,
q) << " "; }
            cout << "\n";

        }

    return 0;

}


int generatepop() {

    int aa,bb;

    srand( time ( NULL ) );

    for (aa=1;aa<=population;aa++)
        {


        cleararray();
        initstack();
        generate();

        //cout << "\n" << (time (NULL) ) << "\n";


        // this condition ensures that every design has iterations no. of
rules
        if (shapeCode[iterations-1]!=0)
```

```
            {

            for (bb=1;bb<=iterations;bb++) { indivShapeCode->set(aa-1, bb-1,
shapeCode[bb-1]); }

            for (i=0;i<=15;i++)
                {
                for (j=0;j<=15;j++)
                    {
                    for (k=0;k<=15;k++)
                        { indivArray->set(aa-1,i,j,k,array[i][j][k]); }
                    }
                }
            }

        else { aa--; }

        }

    return 0;

}


int scorepop() {

    int sccnt;

    maxScore=0;

    if (wtflatno>0) maxScore+=wtflatno;
    if (wtbalcno>0) maxScore+=wtbalcno;
    if (wtheight>0) maxScore+=wtheight;
    if (wtfooti>0) maxScore+=wtfooti;
    if (wtfootj>0) maxScore+=wtfootj;
    if (wtviewsip>0) maxScore+=wtviewsip;
    if (wtviewsim>0) maxScore+=wtviewsim;
    if (wtviewsjp>0) maxScore+=wtviewsjp;
    if (wtviewsjm>0) maxScore+=wtviewsjm;
    if (wtnoviews>0) maxScore+=wtnoviews;

    // The line below is wrong as it does not take into account possible
negative weightings.
    //
maxScore=(wtflatno+wtbalcno+wtheight+wtfooti+wtfootj+wtviewsip+wtviewsim+wtv
iewsjp+wtviewsjm+wtnoviews);

    for (sccnt=1;sccnt<=population;sccnt++) {

        evalbasic(sccnt-1);
        evalextents(sccnt-1);
        evalbalconies(sccnt-1);
        evalviews(sccnt-1);

//        indivScore[sccnt-1] = ( ( abs(circno-goalcircno) * wtcircno ) + (
abs(flatno-goalflatno) * wtflatno ) + ( abs(balcno-goalbalcno) * wtbalcno )
+ ( abs(kmaxextent+1-goalheight) * wtheight ) + ( abs(imaxextent-
iminextent+1-goalfooti) * wtfooti ) + ( abs(jmaxextent-jminextent+1-
goalfootj) * wtfootj ) + ( abs(viewsip-goalviewsip) * wtviewsip ) + (
abs(viewsim-goalviewsim) * wtviewsim ) + ( abs(viewsjp-goalviewsjp) *
wtviewsjp ) + ( abs(viewsjm-goalviewsjm) * wtviewsjm ) );

        indivScore[sccnt-1] = (float) ( maxScore - ( ( ( fabs ( round (
goalflatno / 100. * float(iterations) ) - float(flatno) )  / ( floor ( 0.8 *
float(iterations) ) ) * wtflatno )
                        + ( fabs ( round ( goalbalcno / 100. *
float(flatno) ) - float(balcno) ) / float(flatno) * wtbalcno )
```

```
                            + ( fabs ( float(goalheight) / 4. - (
float(kmaxextent) + 1. ) ) / 14. * wtheight )
                            + ( fabs ( float(goalfooti) / 4. - (
float(imaxextent) - float(iminextent) + 1. ) ) / 15. * wtfooti )
                            + ( fabs ( float(goalfootj) / 4. - (
float(jmaxextent) - float(jminextent) + 1. ) ) / 15. * wtfootj )
                            + ( fabs ( round ( goalviewsip / 100. *
float(flatno) ) - float(viewsip) ) / float(flatno) * wtviewsip )
                            + ( fabs ( round ( goalviewsim / 100. *
float(flatno) ) - float(viewsim) ) / float(flatno) * wtviewsim )
                            + ( fabs ( round ( goalviewsjp / 100. *
float(flatno) ) - float(viewsjp) ) / float(flatno) * wtviewsjp )
                            + ( fabs ( round ( goalviewsjm / 100. *
float(flatno) ) - float(viewsjm) ) / float(flatno) * wtviewsjm )
                            + ( fabs ( round ( goalnoviews / 100. *
float(flatno) ) - float(noviews) ) / float(flatno) * wtnoviews ) ) );

        }

    return 0;

}



int htmlexport() {


    int designno;

    for (designno=1;designno<=population;designno++)
vrmlexport(sorted[designno-1]);

    FILE* htmlfile = NULL;

    htmlfile = fopen("shapeevolution.html","w");
    if (!htmlfile) return -1;

    fprintf (htmlfile, "<html>\n\n<head>\n\n<title>Shape Evolution
Output</title>\n\n<link rel=\"stylesheet\" href=\"shapeevolution.css\"
type=\"text/css\">\n\n</head>\n\n<body>\n\n");

    fprintf (htmlfile, "<table><tr><td><img src=\"seicon.gif\"
align=\"left\"></td>\n\n<td><h2>Shape Evolution Output</h2>\n\n<p>Population
size: <em>%d</em><br />Iterations: <em>%d</em><br />Generations:
<em>%d</em><br />Average score: <em>%f</em><br />Maximum score:
<em>%f</em></p><p>Set mutation rate: <em>%f</em><br />Actual mutation rate:
<em>%f</em><br />Time elapsed: <em>%d s</em></p></p></td></tr></table>\n\n",
population, iterations, generations, avScore, maxScore, mutationrate, (
float(mutacounter)/( float(population) * float (iterations) * float
(generations) ) ), deltatee);

    fprintf (htmlfile, "<table
class=\"border\"><tr><th>Criterion</th><th>Target</th><th>Weight</th></tr>\n
");
    fprintf (htmlfile, "<tr><td>Apartments</td><td>%f
%%</td><td>%f</td></tr>\n", goalflatno, wtflatno);
    fprintf (htmlfile, "<tr><td>Balconies</td><td>%f
%%</td><td>%f</td></tr>\n", goalbalcno, wtbalcno);
    fprintf (htmlfile, "<tr><td>Height</td><td>%d m</td><td>%f</td></tr>\n",
goalheight, wtheight);
    fprintf (htmlfile, "<tr><td>Footprint i</td><td>%d
m</td><td>%f</td></tr>\n", goalfooti, wtfooti);
    fprintf (htmlfile, "<tr><td>Footprint j</td><td>%d
m</td><td>%f</td></tr>\n", goalfootj, wtfootj);
    fprintf (htmlfile, "<tr><td>Views in i+</td><td>%f
%%</td><td>%f</td></tr>\n", goalviewsip, wtviewsip);
```

```
    fprintf (htmlfile, "<tr><td>Views in i-</td><td>%f
%%</td><td>%f</td></tr>\n", goalviewsim, wtviewsim);
    fprintf (htmlfile, "<tr><td>Views in j+</td><td>%f
%%</td><td>%f</td></tr>\n", goalviewsjp, wtviewsjp);
    fprintf (htmlfile, "<tr><td>Views in j-</td><td>%f
%%</td><td>%f</td></tr>\n", goalviewsjm, wtviewsjm);
    fprintf (htmlfile, "<tr><td>No views</td><td>%f
%%</td><td>%f</td></tr></table>\n\n", goalnoviews, wtnoviews);


    fprintf (htmlfile, "<p>Average and best scores per
generation:</p>\n<table class=\"border\">\n");

    int htmlgen;

    for (htmlgen=0;htmlgen<=generations-1;htmlgen++) {

    fprintf (htmlfile, "<tr><td><span
class=\"small\">%f</span></td><td><span class=\"small\">%f</td></tr>\n",
avScoreMonitor[htmlgen], bestScoreMonitor[htmlgen]);

    }

    fprintf (htmlfile, "</table>\n\n");


    for (designno=1;designno<=population;designno++) {

        fprintf (htmlfile, "<hr>\n\n<h4>Design N<sup>o</sup> %d</h4>\n\n",
sorted[designno-1]+1);

        fprintf (htmlfile,"<p>Score: <em>%f</em><br />Fitness:
<em>%f</em></p>\n\n<p>Shape code:<em>", indivScore[sorted[designno-1]],
fitness[sorted[designno-1]]);
            for (i=0;i<=iterations-1;i++) fprintf (htmlfile," %d",
indivShapeCode->get(sorted[designno-1],i) );
        fprintf (htmlfile,"</em></p>\n\n");

        fprintf (htmlfile, "<p><a href=\"design%d.wrl\"
target=\"_new\"> View VRML model. </a></p>", sorted[designno-
1]+1);


        evalbasic(sorted[designno-1]);

        fprintf (htmlfile,"<p>N<sup>o</sup> of flats: <em>%d</em><br
/>\nN<sup>o</sup> of circulation blocks: <em>%d</em><br />\nVolume: <em>%d
m<sup>3</sup></em><br />\nTotal area: <em>%d m<sup>2</sup></em><br
/>\nApartment area: <em>%d m<sup>2</sup></em><br />\nCirculation area:
<em>%d m<sup>2</sup></em></p>\n\n", flatno, circno, ( (flatno*256) +
(circno*64) ), ( (flatno*64) + (circno*16) ), (flatno*64), (circno*16) );

        evalextents(sorted[designno-1]);

        fprintf (htmlfile,"<p>Height: <em>%d m</em><br />\nFootprint: <em>%d
m &times; %d m</em></p>\n\n", ( (kmaxextent+1) *4 ), ( (imaxextent-
iminextent+1) *4 ), ( (jmaxextent-jminextent+1) *4 ) );

        evalbalconies(sorted[designno-1]);

        fprintf (htmlfile,"<p>N<sup>o</sup> of flats with balconies:
<em>%d</em><br />\nPercentage of flats with balconies: <em>%f
%%</em></p>\n\n", balcno, ( (float) ( (float) balcno / (float) flatno ) *100
) );

        evalviews(sorted[designno-1]);
```

```
        fprintf (htmlfile,"<p>Flats with views to +i: <em>%d</em><br
/>\nFlats with views to -i: <em>%d</em><br />\nFlats with views to +j:
<em>%d</em><br />\nFlats with views to -j: <em>%d</em><br />\nFlats with no
views: <em>%d</em><br />\n</p>\n\n", viewsip, viewsim, viewsjp, viewsjm,
noviews );




    }

    fprintf (htmlfile,"</body>\n\n</html>");

    fclose(htmlfile);

    return 1;

}




int htmlreport() {

    int champno;

    FILE* htmlfile = NULL;

    htmlfile = fopen("shapeevolution.html","w");
    if (!htmlfile) return -1;

    fprintf (htmlfile, "<html>\n\n<head>\n\n<title>Shape Evolution
Output</title>\n\n<link rel=\"stylesheet\" href=\"shapeevolution.css\"
type=\"text/css\">\n\n</head>\n\n<body>\n\n");

    fprintf (htmlfile, "<table><tr><td rowspan=\"2\"><img src=\"seicon.gif\"
align=\"left\"></td>\n\n<td colspan=\"2\"><h2>Shape Evolution
Output</h2></td></tr>\n\n<tr><td>\n\n");

    fprintf (htmlfile,
"<table><tr><th> </th><th> </th></tr><tr><td>Population
size</td><td><em>%d</em></td></tr><tr><td>Iterations</td><td><em>%d</em></td
></tr><tr><td>Generations</td><td><em>%d</em></td></tr><tr><td>Set mutation
rate</td><td><em>%8.4f</em></td></tr><tr><td>Actual mutation
rate</td><td><em>%8.4f</em></td></tr><tr><td>Final average
score</td><td><em>%8.4f</em></td></tr><tr><td>Maximum
score</td><td><em>%8.4f</em></td></tr><tr><td>Champion
score</td><td><em>%8.4f</em></td></tr><tr><td>Champion
generation</td><td><em>%d</em></td></tr><tr><td>Time elapsed</td><td><em>%d
s</em></td></tr></table></td>\n\n", population, iterations, generations,
mutationrate, ( float(mutacounter)/( float(population) * float (iterations)
* float (generations) ) ), avScore, maxScore, champScore[0], champGen[0],
deltatee);

    fprintf (htmlfile,
"<td><table><tr><th>Criterion</th><th>Target</th><th>Weight</th></tr>\n");
    fprintf (htmlfile, "<tr><td>Apartments</td><td><em>%8.2f
%%</em></td><td><em>%8.2f</em></td></tr>\n", goalflatno, wtflatno);
    fprintf (htmlfile, "<tr><td>Balconies</td><td><em>%8.2f
%%</em></td><td><em>%8.2f</em></td></tr>\n", goalbalcno, wtbalcno);
    fprintf (htmlfile, "<tr><td>Height</td><td><em>%d
m</em></td><td><em>%8.2f</em></td></tr>\n", goalheight, wtheight);
```

```
    fprintf (htmlfile, "<tr><td>Footprint i</td><td><em>%d
m</em></td><td><em>%8.2f</em></td></tr>\n", goalfooti, wtfooti);
    fprintf (htmlfile, "<tr><td>Footprint j</td><td><em>%d
m</em></td><td><em>%8.2f</em></td></tr>\n", goalfootj, wtfootj);
    fprintf (htmlfile, "<tr><td>Views in i+</td><td><em>%8.2f
%%</em></td><td><em>%8.2f</em></td></tr>\n", goalviewsip, wtviewsip);
    fprintf (htmlfile, "<tr><td>Views in i-</td><td><em>%8.2f
%%</em></td><td><em>%8.2f</em></td></tr>\n", goalviewsim, wtviewsim);
    fprintf (htmlfile, "<tr><td>Views in j+</td><td><em>%8.2f
%%</em></td><td><em>%8.2f</em></td></tr>\n", goalviewsjp, wtviewsjp);
    fprintf (htmlfile, "<tr><td>Views in j-</td><td><em>%8.2f
%%</em></td><td><em>%8.2f</em></td></tr>\n", goalviewsjm, wtviewsjm);
    fprintf (htmlfile, "<tr><td>No views</td><td><em>%8.2f
%%</em></td><td><em>%8.2f</em></td></tr></table>\n\n</td></tr></table>\n\n",
goalnoviews, wtnoviews);


    fprintf (htmlfile, "<!-- Average and best scores per generation:\n");

    int htmlgen;

    for (htmlgen=0;htmlgen<=generations-1;htmlgen++) {

    fprintf (htmlfile, "%f, %f\n", avScoreMonitor[htmlgen],
bestScoreMonitor[htmlgen]);

    }

    fprintf (htmlfile, "--!>\n\n");

    fprintf (htmlfile, "<p> </p>\n\n<table width=\"%d\"
cellpadding=\"0\" cellspacing=\"0\" class=\"chart\"><tr><td height=\"%d\"
valign=\"bottom\" class=\"best\">", (2*generations)+4, int (maxScore*100));


    int chartcol;


    for (chartcol=0;chartcol<=generations-1;chartcol++) {
    fprintf (htmlfile, "<img src=\"lime.png\" width=\"1\" height=\"%d\"
alt=\"%f\">", int (bestScoreMonitor[chartcol]*100),
bestScoreMonitor[chartcol]);
    fprintf (htmlfile, "<img class=\"average\" src=\"orange.png\"
width=\"1\" height=\"%d\" alt=\"%f\">", int (avScoreMonitor[chartcol]*100),
avScoreMonitor[chartcol]);
    }

    fprintf (htmlfile, "</td></tr></table>\n\n" );


    // copy the contents of all the champ registers to the main registers,
so that the evaluation algorithms can find them.

        int copychamps;

        for (copychamps=0;copychamps<=champPool-1;copychamps++) {

            for (i=0;i<=iterations-1;i++) indivShapeCode-
>set(copychamps,i,champCode->get( copychamps, i ) );

            for (i=0;i<=15;i++) {
            for (j=0;j<=15;j++) {
                for (k=0;k<=15;k++) {
                indivArray->set(copychamps,i,j,k,champArray->get(copychamps,
i, j, k));
                }
            }
            }
```

```
        }

    // include attributes of champion and wrl of champion

    for (champno=1;champno<=champPool;champno++) {

        fprintf (htmlfile, "<hr>\n\n<h4>Champ N<sup>o</sup> %d</h4>\n\n",
champno);

        fprintf (htmlfile,"<p>Score: <em>%f</em><br />Appeared in
generation: <em>%d</em></p>\n\n<p>Shape code:<em>", champScore[champno-1],
champGen[champno-1]);
            for (i=0;i<=iterations-1;i++) fprintf (htmlfile," %d",
champCode->get(champno-1,i) );
        fprintf (htmlfile,"</em></p>\n\n");

    vrmlchamp(champno-1);

        fprintf (htmlfile, "<p><a href=\"champ%d.wrl\"
target=\"_new\"> View VRML model. </a></p>", champno);

    evalbasic(champno-1);

        fprintf (htmlfile,"<p>N<sup>o</sup> of flats: <em>%d</em><br
/>\nN<sup>o</sup> of circulation blocks: <em>%d</em><br />\nVolume: <em>%d
m<sup>3</sup></em><br />\nTotal area: <em>%d m<sup>2</sup></em><br
/>\nApartment area: <em>%d m<sup>2</sup></em><br />\nCirculation area:
<em>%d m<sup>2</sup></em></p>\n\n", flatno, circno, ( (flatno*256) +
(circno*64) ), ( (flatno*64) + (circno*16) ), (flatno*64), (circno*16) );

        evalextents(champno-1);

        fprintf (htmlfile,"<p>Height: <em>%d m</em><br />\nFootprint: <em>%d
m &times; %d m</em></p>\n\n", ( (kmaxextent+1) *4 ), ( (imaxextent-
iminextent+1) *4 ), ( (jmaxextent-jminextent+1) *4 ) );

        evalbalconies(champno-1);

        fprintf (htmlfile,"<p>N<sup>o</sup> of flats with balconies:
<em>%d</em><br />\nPercentage of flats with balconies: <em>%f
%%</em></p>\n\n", balcno, ( (float) ( (float) balcno / (float) flatno ) *100
) );

        evalviews(champno-1);

        fprintf (htmlfile,"<p>Flats with views to +i: <em>%d</em><br
/>\nFlats with views to -i: <em>%d</em><br />\nFlats with views to +j:
<em>%d</em><br />\nFlats with views to -j: <em>%d</em><br />\nFlats with no
views: <em>%d</em><br />\n</p>\n\n", viewsip, viewsim, viewsjp, viewsjm,
noviews );


    }


    fprintf (htmlfile,"</body>\n\n</html>");

    fclose(htmlfile);

    return 1;

}
```

```
int calcfitness() {

    float totScore=0;
    int fitgizmo;
    int selectionStrength=1;

    for (fitgizmo=0;fitgizmo<=population-1;fitgizmo++)
totScore+=indivScore[fitgizmo];

    avScore=totScore/population;

    for (fitgizmo=0;fitgizmo<=population-1;fitgizmo++)
fitness[fitgizmo]=pow( (indivScore[fitgizmo]/avScore), selectionStrength);



    return 1;

}



int tournament() {

    // randomly pick two individuals, compare their fitnesses, and select
the best one
    // put markers for the intermediate population in intermediate[P]

    int tourneygizmo;

    for (tourneygizmo=0;tourneygizmo<=population-1;tourneygizmo++) {

    int redcorner = ( rand () >> 8 ) % population;
    int bluecorner = ( rand () >> 8 ) % population;

    //cout << "\n" << redcorner << "(" << indivScore[redcorner] << ") vs. "
<< bluecorner << "(" << indivScore[bluecorner] << ")\n";

    if (indivScore[redcorner]>=indivScore[bluecorner]) {
        intermediate[tourneygizmo]=redcorner;
        // cout << redcorner << " wins!\n";
    }
    else {
        intermediate[tourneygizmo]=bluecorner;
        //cout << bluecorner << " wins!\n";
    }

    }

    return 1;

}


int initxoverstack() {

    //initialise a xoverStack[iterations-1] of crossover positions with a
random order, then start a loop
    //that is "iterations" long that picks a rule, and the below ensues.


    int jen, jentemp;
```

```
    for(jen=0;jen<=(iterations-2);jen++) xoverStack[jen]=jen;

    for(jen=0;jen<=(iterations-2);jen++) {
        int jenpos = ( rand() >> 8) % (iterations-1);


        jentemp = xoverStack[jen];
        xoverStack[jen] = xoverStack[jenpos];
        xoverStack[jenpos] = jentemp;
    }

    return 0;

}


int xshuffler() {

    //initialise a xoverStack[iterations-1] of crossover positions with a
random order, then start a loop
    //that is "iterations" long that picks a rule, and the below ensues.


    int fiona, fionatemp;

    for(fiona=0;fiona<=(population-1);fiona++) xshuffle[fiona]=fiona;

    for(fiona=0;fiona<=(population-1);fiona++) {
        int fionapos = ( rand() >> 8) % (population);


        fionatemp = xshuffle[fiona];
        xshuffle[fiona] = xshuffle[fionapos];
        xshuffle[fionapos] = fionatemp;
    }

    return 0;

}


int crossover() {

    // Crossover probabilistically.
    // make sure crossover does not create monsters
    // - try different crossover points until offspring is valid
    // - do embryogenesis and check phenotypes are valid
    // - if no crossover point yields valid offspring, just copy parents
over

    int xgizmo1;
    int xgizmo2;
    int xgizmo3;
    int xgizmo4;

    for(xgizmo1=0;xgizmo1<=(population-2);xgizmo1+=2) {

    initxoverstack();
    xshuffler();

    for (xgizmo2=0;xgizmo2<=(iterations-2);xgizmo2++) {

        for (xgizmo3=0;xgizmo3<=xoverStack[xgizmo2];xgizmo3++) {

        indivShapeCode->set(population+xgizmo1, xgizmo3, indivShapeCode-
>get(xshuffle[xgizmo1],xgizmo3));
        indivShapeCode->set(population+xgizmo1+1, xgizmo3, indivShapeCode-
>get(xshuffle[xgizmo1+1],xgizmo3));
```

```
        }

        for (xgizmo3=xoverStack[xgizmo2]+1;xgizmo3<=(iterations-
1);xgizmo3++) {

        indivShapeCode->set(population+xgizmo1, xgizmo3, indivShapeCode-
>get(xshuffle[xgizmo1+1],xgizmo3));
        indivShapeCode->set(population+xgizmo1+1, xgizmo3, indivShapeCode-
>get(xshuffle[xgizmo1],xgizmo3));

        }

        // now must test if resulting two xoverShapeCodes generate valid
designs...

        //if yes, break loop 2, continue loop 1.

        if
((embryogenesis(population+xgizmo1)==1)&&(embryogenesis(population+xgizmo1+1
)==1)) {
        // cout << "."; //debugging
        break;
        }

        // cout << "X"; // debugging

        // if not, continue loop 2.

        // if the xoverStack is exhausted, just copy the individuals over as
they are.

        if (xgizmo2==(iterations-2)) {

        for (xgizmo4=0;xgizmo4<=iterations-1;xgizmo4++) {

            indivShapeCode->set(population+xgizmo1, xgizmo4, indivShapeCode-
>get(xshuffle[xgizmo1],xgizmo4));
            indivShapeCode->set(population+xgizmo1+1, xgizmo4,
indivShapeCode->get(xshuffle[xgizmo1+1],xgizmo4));
        }

        // cout << "X"; // debugging

        break;
        }

    }

    }

    return 1;

}


int copydown() {

// Copies the top half of the indivShapeCode array to the bottom half

    int copypop;
    int copyiter;

    for (copypop=0;copypop<=population-1;copypop++) {

    for (copyiter=0;copyiter<=iterations-1;copyiter++) {
```

```
        indivShapeCode->set(copypop, copyiter, indivShapeCode-
>get(population+copypop, copyiter));

    }

    }

    return 1;

}


int mutate() {

    // Mutates every bit of the population according to mutationrate
    // Mutation must be selective lest it invalidates designs too often
    // Only mutate to circulation rule

    copydown();

    int mutapop;
    int mutaiter;

    int mutation;

    for (mutapop=0;mutapop<=population-1;mutapop++) {

    for (mutaiter=0;mutaiter<=iterations-1;mutaiter++) {

        // If mutationrate allows, change a bit to a new one (within top
population), selected randomly

        if ( ( ( rand () >> 8 ) % 1000000) < ( int (
(mutationrate*1000000.0) ) ) ) {

        mutation = ( ( ( rand () >> 8 ) % 22 ) + 1 );

        indivShapeCode->set( population+mutapop, mutaiter, mutation );

        // check validity of mutant

        if ( (embryogenesis(population+mutapop)==1) && ( ( indivShapeCode-
>get(mutapop, mutaiter) ) != mutation ) ) {

            // if valid, copy down to bottom population

            indivShapeCode->set(mutapop, mutaiter, mutation );

            //cout << "\n Mutated gene " << mutaiter << " of individual " <<
mutapop << "\n";

            mutacounter++;
            // cout <<"m" << mutapop << " "; // debugging

        }

        else {

            indivShapeCode->set( population+mutapop, mutaiter,
indivShapeCode->get(mutapop, mutaiter) );

        //cout << "M"; // debugging

        }

        }

    }
```

```
    }

    return 1;

}


int comparecodes( int champid, int individ) {

    int comparador;

    int sameornot=1;

    for (comparador=0;comparador<=iterations-1;comparador++) {

    if (champCode->get(champid,comparador)!=indivShapeCode-
>get(individ,comparador)) {

        sameornot=0;
        break;

    }

    }

    return sameornot;

}



// main does nothing but call generator and vrmlexport.

int main() {

    fileinput();

    generatepop();

    int maingizmo; //looping variable
    int darwin; //looping variable

    int champgizmo; //looping variable
    int billboard; //looping variable
    int shiftdown; //looping variable

    // this variable should be changed to 1 if
    // this generation managed to produce a champion,
    // i.e. one of its top performers placed in the "billboard"
    int placed=0;

    cout << "\nProgress:\n";

    deltatee = time( NULL );

    for (maingizmo=0;maingizmo<=generations-2;maingizmo++) {

    scorepop();

    sortpop();

    calcfitness();

    avScoreMonitor[maingizmo]=avScore;
    bestScoreMonitor[maingizmo]=indivScore[ sorted[0] ];
```

```
        // start routine for picking champions

    placed=0;

    for (champgizmo=0;champgizmo<=champPool-1;champgizmo++) {

        for (billboard=0;billboard<=champPool-1;billboard++) {

        // if the new design scores higher than an existing champ, or
        // if the new design has the same score with an existing champ and
it's not identical it should be placed on the champ list

        if ( (indivScore[ sorted[champgizmo] ] > champScore[billboard]) || (
(indivScore[ sorted[champgizmo] ] == champScore[billboard]) &&
(comparecodes(billboard, sorted[champgizmo])!=1) ) )  {

            placed=1;

            //routine for shifting down lower positions...

            for (shiftdown=champPool-1;shiftdown>=billboard+1;shiftdown--) {

            champScore[shiftdown]=champScore[shiftdown-1];
            champGen[shiftdown]=champGen[shiftdown-1];

            for (i=0;i<=iterations-1;i++) champCode-
>set(shiftdown,i,champCode->get(shiftdown-1, i) );

            for (i=0;i<=15;i++) {
                for (j=0;j<=15;j++) {
                for (k=0;k<=15;k++) {
                    champArray->set(shiftdown,i,j,k,champArray-
>get(shiftdown-1, i, j, k) );
                }
                }
            }

            }

            //routine for copying new champion into the current position
(billboard)

            champScore[billboard]=indivScore[ sorted[champgizmo] ];
            champGen[billboard]=maingizmo+2;

            for (i=0;i<=iterations-1;i++) champCode-
>set(billboard,i,indivShapeCode->get( sorted[champgizmo], i ) );

            for (i=0;i<=15;i++) {
            for (j=0;j<=15;j++) {
                for (k=0;k<=15;k++) {
                champArray->set(billboard,i,j,k,indivArray-
>get(sorted[champgizmo], i, j, k));
                }
            }
            }

            // once this individual from the current generation has found
its place, stop searching for it.
            break;

        }

        // if the design is identical it should be ignored, but set placed=1

        if (comparecodes(billboard, sorted[champgizmo])==1) {

            placed=1;
```

```
            break;
            }

        }

        if (placed=0) break;


//    if (indivScore[ sorted[0] ] >= champScore) {
//
//            champScore=indivScore[ sorted[0] ];
//            champGen=maingizmo+1;
//
//            for (i=0;i<=iterations-1;i++) champCode[i]=indivShapeCode-
>get( sorted[0], i );
//
//            for (i=0;i<=15;i++) {
//                for (j=0;j<=15;j++) {
//                    for (k=0;k<=15;k++) {
//                        champArray[i][j][k]=indivArray->get(sorted[0], i,
j, k);
//                    }
//                }
//            }

        }

    tournament();

    crossover();

    mutate();

    for (darwin=0;darwin<=population-1;darwin++) {
        embryogenesis(darwin);
        if (embryogenesis(darwin)!=1) {

        cout << "\nHoly Cow! Individual " << darwin << " is a hideous
monster!\n"; // break programme if an invalid individual somehow infiltrates
the population

        return 0;
        }
    }


    cout << "*";

    }

    scorepop();

    sortpop();

    calcfitness();

    avScoreMonitor[generations-1]=avScore;
    bestScoreMonitor[generations-1]=indivScore[ sorted[0] ];

    deltatee = time( NULL ) - deltatee;

    htmlreport();

    cout << "\n--- Shape Evolution completed.\n";


    //cout << "\nTotal mutations: " << mutacounter << "\n";
```

```
    //txtview();
    //txtexport();
    //vrmlexport(0);

    //viewshapecode(0);


    //Be careful with the order the evaluation functions are called.
    //Some are referencing variables produced by earlier evaluation
    //functions. This is in order to avoid evaluating too many loops.

    //evalbasic(0);

    //cout << "No of flats: " << flatno << "\n";
    //cout << "No of circulation blocks: " << circno << "\n";
    //cout << "Volume: " << ( (flatno*256) + (circno*64) ) << " m^3\n";
    //cout << "Total area: " << ( (flatno*64) + (circno*16) ) << " m^2\n";
    //cout << "Apartment area: " << (flatno*64) << " m^2\n";
    //cout << "Circulation area: " << (circno*16) << " m^2\n";

    //evalextents(0);

    //cout << "Height: " << ( (kmaxextent+1) *4 ) << " m\n";
    //cout << "Footprint: " << ( (imaxextent-iminextent+1) *4 ) << " m x "
<< ( (jmaxextent-jminextent+1) *4 ) << " m\n";

    //evalbalconies(0);

    //cout << "No of flats with balconies: " << balcno << "\n";
    //cout << "Percentage of flats with balconies: " << ( (float) ( (float)
balcno / (float) flatno ) *100 ) << " %\n";

    //evalviews(0);

    //cout << "Flats with views to +i: " << viewsip << "\n";
    //cout << "Flats with views to -i: " << viewsim << "\n";
    //cout << "Flats with views to +j: " << viewsjp << "\n";
    //cout << "Flats with views to -j: " << viewsjm << "\n";
    //cout << "Flats with no views: " << noviews << "\n";

    //viewshapecodes();



    return 1;
}
```