FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Visibility-Based Obstacle Placing

Automated Obstacle Placing based on Circularity

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Carina Schwab
Matrikelnummer 0726458

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Mitwirkung: Univ.Ass. Mag.arch. Richard Schaffranek

Wien, 09.01.2016

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Visibility-Based Obstacle Placing

Automated Obstacle Placing based on Circularity

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Computational Intelligence

by

Carina Schwab
Registration Number 0726458

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.-Prof. Dipl.-Ing. Dr.techn. Günther Raidl
Assistance: Univ.Ass. Mag.arch. Richard Schaffranek

Vienna, 09.01.2016

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Carina Schwab
Höritzergasse 2/13, 1140 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Wien, 09.01.2016)

(Unterschrift Verfasserin)

Danksagung

Ich bedanke mich vielmals bei meinem Betreuer Prof. Günter Raidl für die Ermöglichung dieser interessanten Diplomarbeit, die hervorragende Betreuung, richtungsweisenden Vorschläge sowie seine Geduld. Mein Dank gilt auch Richard Schaffranek für die Bereitstellung des Diplomarbeitsthemas, seine Begeisterung für dieses Thema und seine anwendungsorientierten Ideen.

Vielen Dank an Etienne Cordonnier für die Unterstützung, die interessanten Diskussionen über die englische Sprache und insbesondere für die zur Verfügungsstellung seiner Rechnerressourcen für meine Berechnungen.

Mein herzlicher Dank gilt auch meiner Familie, die in meinem Ausbildungsweg immer hinter mir standen und mich in jeder Lebenslage unterstützt haben.

Acknowledgements

I would like to express my gratitude to my supervisor Prof. Günter Raidl for the opportunity to work on this master thesis, the great mentoring, his helpful advises and his patience. I would also like to thank Richard Schaffranek for providing the topic for this thesis, his involvement in this topic and his application-oriented ideas.

I am also very grateful to Etienne Cordonnier for his support, for the interesting discussions about the english language and especially for providing me with his computer's calculation power.

My warm thanks to my family who supported me during my education and are always there for me.

Abstract

In this thesis the problem of placing multiple obstacles in a discrete 2D environment is considered. An example real life application for this problem is the placing of exhibition objects in museums. Previous studies by various authors showed that people's behaviour in space is linked to the circularity measure – also called compactness – of their field of vision. This measure describes the shape of the space that can be seen from a specific point. An evolutionary algorithm (EA) is developed that places the given obstacles at the remaining unblocked space according to the compactness at the remaining unoccupied space. The EA uses different target functions to either minimise or maximise the compactness measure globally or locally. Several test series are executed in order to assess influence of the algorithm's parameters on its performance. The solutions of the EA show emerging patterns for different target functions. From those results a faster constructive heuristic is developed that can provide several different solutions to simple placing problems within short time. In the EA a version of the Shadow Casting algorithm is used to calculate the field of vision for a point. The heuristic extends this algorithm by placing obstacles while iterating through the visible areas. The objects are placed in a way such that preset points show a high compactness value and that the architect can be inspired by the presented diverse solutions.

Kurzfassung

Diese Masterarbeit befasst sich mit dem Problem eine Mehrzahl an Objekten in einem diskreten 2D Raum zu platzieren. Ein Beispiel eines realen Problems dieser Art ist das Platzieren von Ausstellungsstücken in einem Museum. Vergangene Studien diverser Autoren zeigten, dass das Verhalten von Menschen in Räumen mit der Zirkularität – auch Kompaktheit – von ihrem Sichtfeld zusammen hängt. Dieses Maß beschreibt die Form des Raumes der von einem bestimmten Punkt aus gesehen werden kann. Im Rahmen dieser Arbeit wurde ein evolutionärer Algorithmus (EA) entwickelt, der definierte Objekte abhängig von der Kompaktheit des freibleibenden Raumes platziert. Der EA verwendet verschiedene Zielfunktionen um die Kompaktheit global oder lokal entweder zu minimieren oder zu maximieren. Mehrere Testreihen wurden ausgeführt um den Einfluss der Parameter des Algorithmus auf seine Effizienz zu untersuchen. Die Lösungen die der EA produziert zeigen dass sich bestimmte Muster für unterschiedliche Zielfunktionen entwickeln. Basierend auf diesen Ergebnissen wurde eine schnellere konstruierende Heuristik entwickelt, die mehrere unterschiedliche Lösungen für einfache Platzierungsprobleme in kurzer Zeit berechnen kann. Im EA wurde eine Version des Shadow Casting Algorithmus verwendet um das Sichtfeld für einen Punkt zu berechnen. Die Heuristik erweitert diesen Algorithmus um das Platzieren von Objekten während dem Durchlaufen des Sichtfeldes. Die Objekte werden so platziert, dass voreingestellte Punkte eine hohe Kompaktheit aufweisen und die präsentierten Lösungen dem Architekt als Inspiration dienen.

Contents

1 Introduction **1**
1.1 Methodology . . . 2

2 Isovist and Compactness Background **3**
2.1 Isovist . . . 3
2.2 Compactness . . . 4
2.3 Compactness in discrete space . . . 5
2.4 Recursive Shadow Casting . . . 6
2.5 Compactness calculation . . . 9

3 Related Work **11**
3.1 Introduction . . . 11
3.2 Isovist properties and human behaviour . . . 11
3.3 Generative approaches in architecture . . . 18
3.4 Conclusion . . . 22

4 Evolutionary Approach: Introduction **25**
4.1 Problem definition . . . 25
4.2 Background to evolutionary algorithms . . . 25
4.3 Evolutionary algorithm variants . . . 28
4.4 Evolutionary algorithms in architecture . . . 32

5 Evolutionary Approach: Details **35**
5.1 Overview . . . 35
5.2 Evolutionary algorithm for object placement . . . 36
5.3 Shadow Casting algorithm . . . 42
5.4 Typical solutions . . . 46

6 Evolutionary Approach: Evaluation **51**
6.1 Test setting . . . 51
6.2 Number of generations . . . 52
6.3 Mutation and crossover . . . 52
6.4 Children . . . 53
6.5 Population size . . . 54
6.6 Crossover points . . . 56
6.7 Tournament group size . . . 56
6.8 Removal of duplicates . . . 57
6.9 Counting extrema points . . . 57
6.10 Influence of instance size scaling on run time . . . 59
6.11 Scaling of complexity . . . 60
6.12 Build footprint and cross floor area . . . 63
6.13 Performance check against a local improvement . . . 66

7 Heuristic Approach **69**
7.1 Introduction 69
7.2 Analysis of evolutionary results 69
7.3 Basic deterministic heuristic 70
7.4 Introducing diversity: a random factor 74

8 Conclusions and Future Work **77**
8.1 Conclusions 77
8.2 Future Work 78

Bibliography **79**

CHAPTER 1

Introduction

Architectural planning problems evolving around designing space are multilayer problems with a high number of complex relationships influencing the quality of the solution. Parameters relevant in a planning problem are for example technical, functional, aesthetic or legal. Because of their complexity those problems cannot be easily broken down into sub-problems or described formally. Rittel and Webber called this property wicked [1], referring to the fact that these problems cannot be abstracted and described in detail without already specifying the direction in which the solution is considered. An understanding of all possible solutions would therefore be necessary in order to specify the problem domain in detail. It is vital for the planner to gain a better understanding of the problems' nature by exploring possible solutions. Since the problems are wicked it cannot formally be determined when a solution is „right“ or „good enough“. Time is often an important factor which limits the continuous learning and development cycle of a planner [1]. In order to support this exploratory process automatic planning tools can be used to present diverse solutions within a short period of time to the planner. This provides the planner with basic solutions and new ideas he can adapt into his work. Another advantage of computer aided planning is that criteria which would be hard or long to check by people can be effectively integrated into the design process of generative systems [2].

This thesis considers one special type of planning problems – the placing of objects in a space oriented on the compactness values of the unoccupied places. This specific problem occurs for example in museums or exhibitions, where it is relevant how the samples are placed. Batty believes that how far and how much we can see is a key issue for good architecture [3]. When people move through space their experience is determined by the properties which they can perceive through their various senses, in particular their sense of vision. What we can recognise with our eyes is mainly determined by the surface characteristics like materials, textures and colours and the arrangement and size of the environment elements. These attributes are referred to as visuospatial properties. The arrangement of elements in space in particular is termed the spatial configuration. An intuitive way to imagine what people see when they are moving through space is thinking about the field of vision - called isovist - of an individual's vantage point [4]. Conroy-Dalton [5] and Wiener [6] found out that these isovists include information that is used by individuals for decision making in the context of way-finding [7].

There are different measures of space, which relate to the shape of isovist. In this work we are concentrating on the measurement „compactness“ or „circularity“ as defined by Benedikt [8]. This measure describes how circular or complex an isovist is by relating its area with its perimeter. It can be used to describe specific types of fields of vision and relates to how people experience a space. Franz and Wiener 2008 conducted virtual reality experiments in which

the test persons should rate beauty, complexity and spaciousness of the configuration they perceived. They discovered a strong correlation of those ratings to the values of compactness, area and occlusivity [7]. Therefore the compactness measure has a lot of potential to be used to guide the planning process in order to achieve that the effect of the resulting space on people exploring it meets the desired one.

1.1 Methodology

There exist many urban codes and pattern books which recommend dimensions and shapes of roads or open spaces. Designers refer to such guidelines in order to ensure certain visuospatial properties in environments. But since every design problem is unique those principles are of limited use in the planning process. A more useful mechanism would produce several patterns based on the intended effect and enable an intelligent search for appropriate solutions in different contexts. This approach is called „inverse design“ [7].

As method for the generation of solutions evolutionary algorithms (EAs) are used in this thesis. This type of algorithms is especially suited to deal with problems whose solutions should contain properties that are not explicitly included in the problem descriptions. If solutions with such qualities emerge then the design process is considered creative [2]. DeLanda sees the potential of exploratory genetic algorithms as visualisation tools where not all possible configurations can be explored in advance [9]. Schneider and König chose EAs in some of their studies because of their ability to adapt to changing problems and the fact that no patterns are needed to guide the search process [7]. In one of their study they came to the conclusion that isovist properties are well suited as objective criteria for the optimisation of layouts [2]. In this thesis an EA is developed whose objective criteria considers different aspects of the isovists compactness measure.

The EA is experimentally evaluated and its results analysed. Emerging patterns are identified and used to construct a simple heuristic. This second algorithm provides different reasonable solutions for a problem instance within short time.

CHAPTER 2

Isovist and Compactness Background

2.1 Isovist

While the originator of the term „isovist“ is Tandy (1967) [10] [11] [4], Benedikt was the first to use it in syntax research [11]. He developed Tandys idea further by introducing a set of analytical measurements of isovist properties [8], of which one – the compactness – will be in the focus of this work.

Let us consider a vantage point x in a simply connected region D in three-dimensional space. An isovist can be three-dimensional or two-dimensional. For simplicity we will here only look at the horizontal two-dimensional plane section through x. An example of such a region is shown in 2.1, the region boundary is denoted as δD. If D contains any opaque, visible objects they block the view so that from x not all points of D are visible. All of those objects together form an environment E: the collection of all visible, view blocking objects and their positions in D. 2.1 shows one possible environment in region D.

Figure 2.1: Left: Region D with region boundary δD. Right: Environment E in region D (figures from [8]).

The points of D which are visible from x form a single polygon without holes [4]. This polygon, the set of all points visible from x, is called „isovist“ V_x at point x. More formally:

$$V_x = \{v \in D : v \text{ is visible from } x\} \tag{2.1}$$

x is also called “generating location” [4]. Figure 2.2 shows examples of isovists in D for environment E.

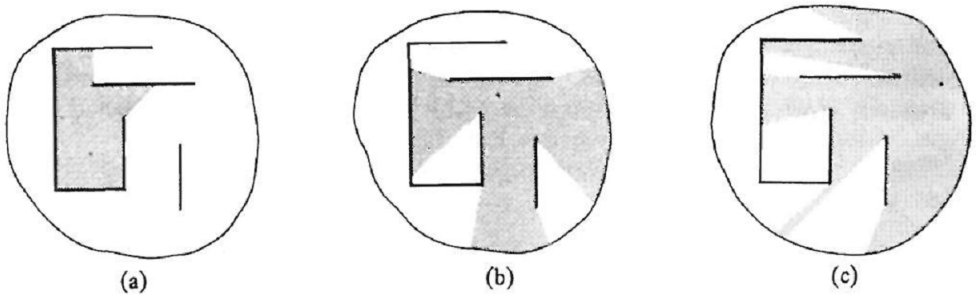


Figure 2.2: Three different isovists of three points in D (figure from [8]).

For any point x in a solid object it holds that $V_x = \emptyset$. It is possible that two or more vantage points in D have the same visible set V. For two different points x and y it then holds that $V_x = V$ and $V_y = V$. The isovist at x describes what an observer at vantage point x can see. The properties of an isovist, like area (notated as A_x) or perimeter (notated as δV_x), therefore relate to some aspects of how the observer experiences the space.

An isovist can contain different types of boundaries. First the field of vision of point x can obviously be bordered by visible surfaces. Lets call these perimeter sections S_x. Secondly when looking at the 2D plan of an isovist it can be seen that the isovist can also have borders which are the result of a sight line from x joining the border of a visible object. These perimeter sections are named R_x and the sum of all their lengths is called the occlusivity $Q_x = |R_x|$ of the isovist. Q_x is small at locations where few or no glances into another part of the configuration are possible. A completely closed, convex space for example has an occlusivity of zero. The isovists perimeter can also contain region border sections δD_x.

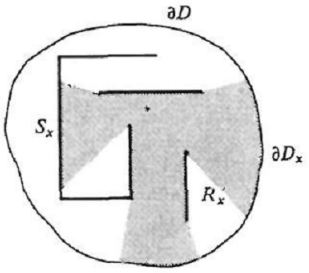


Figure 2.3: The perimeter parts of an isovist: S_x, R_x and δD_x (figure from [8]).

Benedikt gave an alternative definition of an isovist: V_x is the set of line segments from vantage point x to points v' on the boundary surfaces S_x and δD_x. He calls those lines radials, since they „radiate“ out from x to the boundary. The length of a radial is the euclidean distance between x and v' and computed by the following formula [8]:

$$l_{x,\theta} = d(x, v') = ||v' - x|| = \left[(v'^2_1 - x^2_1) + (v'^2_2 - x^2_2)\right]^{1/2} \tag{2.2}$$

Where $0 \leq \theta \leq 2\pi$ represents the direction of the radial, as displayed in Figure 2.4 and v'_1, v'_2 and x_1, x_2 are the coordinates of v' and x.

2.2 Compactness

The compactness – or circularity – of an isovist is:

$$N_x = \frac{(\delta V_x)^2}{4\pi A_x} \tag{2.3}$$

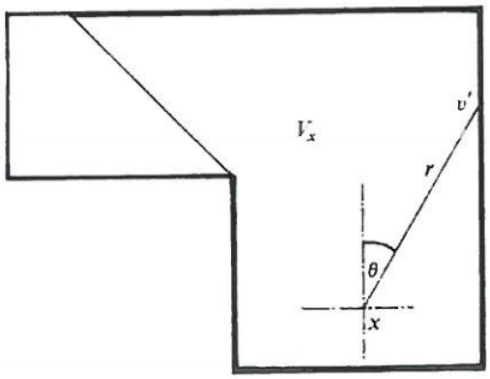


Figure 2.4: Illustration of a radial r of isovist V_x of length $l_{x,\theta}$ (figure from [8]).

The formula is based on the ratio of perimeter to area and built in such a way that it evaluates to one for a circle. When V_x is not a disc, N_x is bigger than one [8].

Batty used a slightly different equation. He defined his measure of convexity "the ratio of the radius of an idealised circle associated with the actual area of the isovist to the radius of an idealised perimeter from the actual perimeter in question" [3]. The formula is:

$$\Psi = \left(\frac{A_x}{\pi}\right)^{1/2} \Big/ \frac{\delta V_x}{2\pi} \tag{2.4}$$

Battys measure Ψ falls between zero for a straight line isovist and one for a circle and Batty states that the measure is sharpened by squaring it [3], which is:

$$\Psi^2 = \frac{4\pi A_x}{(\delta V_x)^2} \tag{2.5}$$

This resembles the inverse of Benedikts compactness formula, which is the one we work with in this thesis. This means that the less circular a shape is, the bigger is its compactness value.

In order to describe a whole environment more than one isovist has to be considered. Benedikt suggested that the interplay of isovists is influencing how we experience a space [4]. He generated one property for all isovists in the environment and assigned the property value to the point which generated the isovist. He then plotted these so called "isovist fields" as a topography of contour lines in order to show how and how quickly those properties vary through the space [8] [4]. In general isovist fields can be plotted by representing the configuration on a grid and assigning every point a colour that relates to its isovist property value. High values are displayed as light points and low values as dark points [7].

2.3 Compactness in discrete space

For simplicity we use a grid of quadratic cells as underlying system for this work. Each cell is considered to be 1x1 in size. Figures 2.5a and 2.6 show different isovist shapes. Shape V_a is a perfect circle with area 10 and radius $r \approx 1.784$. V_c and V_d are shapes in discrete space which could be approximations for that circle. But the compactness values N_c and N_d of those two shapes are more than the double of the compactness N_a of the circle. A better shape for the circle in discrete space is the square. Every square has a compactness of 1.273, which is the lowest possible compactness value in discrete space. This example shows that the discretisation of real environments and objects can have great influence on the compactness of the resulting system and therefore on the results of compactness analysis.

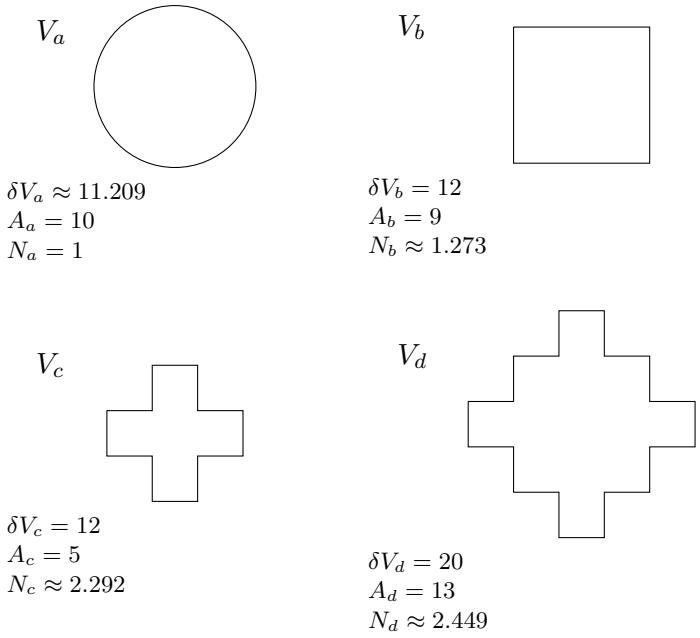


Figure 2.5: Demonstration of different area A_x, perimeter δV_x and compactness N_x values for shapes that approximate a circle in discrete space.

Figure 2.6 shows different example shapes which all have an area of 10 and therefore can be compared with the circle of Figure 2.5a. Shape V_e is close to a circle and has a small compactness of 1.56, while shape V_g is a straight line and shape V_f resembles a curled line. Both, V_f and V_g have a perimeter of size $|\delta V_c| = |\delta V_d| = 22$. Since their area is the same as well, although they have different forms, their compactness is the same: $N_g = N_f \approx 3.852$. This demonstrates that, although the compactness value indicates how "round" or "circle-like" an isovist is, there are several different shapes which could incorporate that property. As the perimeter of a shape increases while its area stays unchanged, the compactness of this shape increases too. While there is a fixed smallest compactness for any environments, the maximum possible compactness depends on the size and shape of the environment. But the maximum compactness for a specific area is found in shapes like straight or very curled lines.

There is no absolute maximum compactness value that holds for any environments. The bigger the environment the bigger the possible maximum compactness value. If we consider a straight line isovist then its perimeter value $\delta V_s = 2 * A_s + 2$ since every area field is bordered by two perimeter lines and the shape is closed by one perimeter line on each end. The limes for such shapes can be calculated for a growing area (the bigger the environment the longer the maximum straight line isovist) and shows that compactness values can grow infinitely:

$$\lim_{A_x \to \infty} \frac{(2A_x + 2)^2}{4\pi A_x} = \infty \tag{2.6}$$

2.4 Recursive Shadow Casting

In order to calculate which cells are visible from a vantage point the shadow casting algorithm divides the area into eight equal parts. Those octants are processed one by one. The algorithm follows a specific order for every octant, going row by row, column by column. It begins at the vantages points and moves outwards by iterating through the columns/rows, always starting

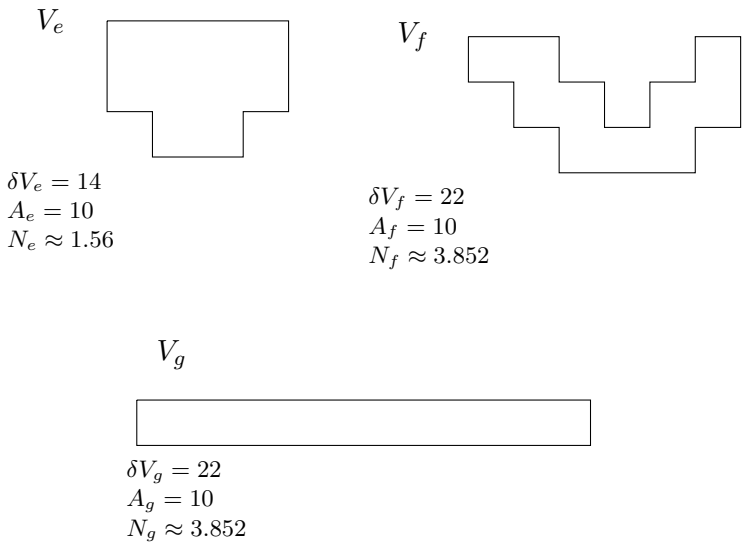


Figure 2.6: Examples of the compactness values N_x for different shapes with varying perimeter δV_x and area $A_x = 10$.

at the diagonal boundary. The advantage of this algorithm compared to ray casting is that less cells are visited more than once [12].

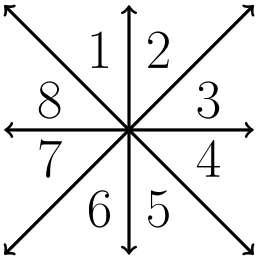


Figure 2.7: The octants as processed for the vantage point in the middle.

The octants are processed in the following order:

Octant 1 and 6: row by row, from the leftmost cell of the row to the rightmost cell.
Octant 2 and 5: row by row, from the rightmost cell of the row to the leftmost cell.
Octant 3 and 8: column by column, from the topmost cell of the column to the bottommost cell.
Octant 4 and 7: column by column, from the bottommost cell of the column to the topmost cell.

A scan always has a start slope and an end slope, which resemble the left and the right edge of the scanned area. The edges between the different octants are always shared and therefore visited by the scans of both octants they border. The slope is calculated by two points (x_1, y_1) and (x_2, y_2) through which the slope passes, one of the points is the vantage point:

$$\begin{aligned} slope &= (x_1 - x_2) \ / \ (y_1 - y_2) \\ inverse\ slope &= (y_1 - y_2) \ / \ (x_1 - x_2) \end{aligned}$$

Depending of the octant, the slope or inverse slope is used. For octant 1 the start slope is one for example, expressing that if y is increased by one, x has to be increased by one too

in order to stay on the slope. The end slope for octant 1 is zero. So for any y value, x stays the same.

As mentioned every octant is scanned in its order. As soon as an obstacle is found a new scan is started in the next row/column. The start slope of the new scan is the same as for the original scan. The end slope is calculated using the center of the vantage point and the corner of the first blocking cell which is closest to the start slope. The corner is simply calculated by adding/subtracting 0.5 from the coordinates of the cell. The scan in the original row/column is continued. When the algorithm finds the first non-blocking cell after a sequence of blocking cells, a new start slope for the scan will be calculated using the center of the vantage point and the corner of the last blocking cell which is closest to the original end slope.

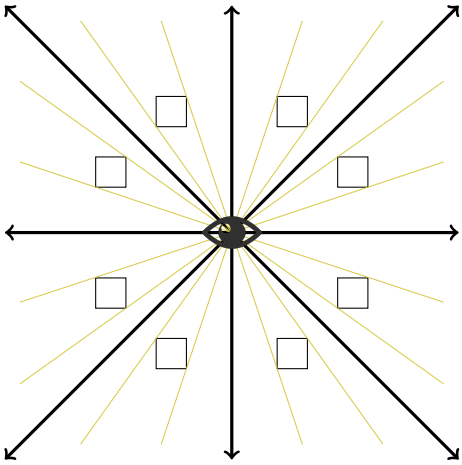


Figure 2.8: Slopes for example blocking cells in every octant.

The two scans are operating individually on the different sides of the blocking sequence. The cells which are behind the blocking cells are not visible and using the end/start slope of the two scans they are simply skipped. For every new sequence of blocking cells a new scan is started and the procedure is repeated. Figure 2.9 shows an example of a calculation with two scans.

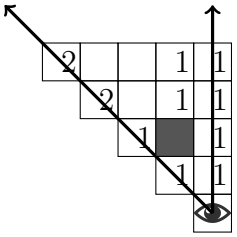


Figure 2.9: Octant one with one blocking cell. The cells scanned by the original scan are labeled „1“, the cells scanned by the second scan are labeled „2“.

The operating mode of the algorithm induces that it is possible to see through the two touching corners of diagonally adjacent blocking cells. Only the cells that were scanned before the current one are considered for the line of sight. A cell which is located in the same row (octants 1,2,5,6) / column (octants 3,4,7,8) but evaluated afterwards therefore cannot influence the evaluation result of a cell. An example is shown in Figure 2.10. The cell marked with * is evaluated without considering the blocking cell to it's right. The blocking cell in the row before the * cell does not block the view to the * cell and therefore the * cell i s considered visible.

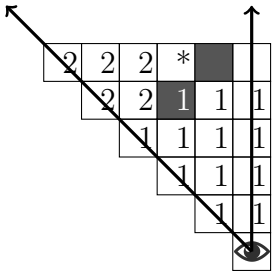


Figure 2.10: The cell marked with *, that can be seen between the two corners, is evaluated after the first and before the second blocking cell by scan 1.

2.5 Compactness calculation

To calculate the compactness value of an isovist, the area and perimeter have to be evaluated first. For the area calculation every visible non-blocking cell is counted. A cell can be seen as a square with length and width of one and therefore evaluates to an area of one. The perimeter is calculated by summing up the length of the outer borders of the isovist. Figure 2.11 shows an example of a small region with two blocking cells on the right and left of the bottom row. The area of the isovist consists of all white cells. The perimeter is marked by the thicker black line around them.

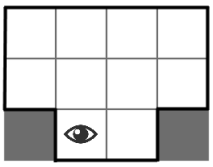


Figure 2.11: An isovist in concrete space with area=10 and perimeter=14

When working with a discrete space consisting of cells, forming a perfect circle is not possible. The lowest possible compactness value in this setting can be reached in any square. The size of the square V_s does not matter, $N_s \approx 1.273$ for all squares. Since the compactness describes the shape of an isovist by relating its perimeter to its area, the scale of the shape does not change the compactness value.

CHAPTER 3

Related Work

3.1 Introduction

When looking at related work there are two different aspects to consider. On one hand work about the isovist properties and their connection to human behaviour is important for the relevance of this work and on the other hand generative approaches in architecture provide information about already used algorithms.

In the first part of this chapter experiments and analysis that set isovist properties in a relation to human behaviour are described. The concentration is on general isovist meanings for human observers and on properties that can be set in a relation to compactness. It starts with the introduction to Benedikt's [8] model that links information and observer exposure to isovist properties. He surmises that with their movement behaviour people try to control the isovist properties of their visual field and through that they influence how much information they have to process at a time and to how many possible other persons they are exposed. This is related to the use of space as public or private spaces. Turner et al [4] analyse the usage of actually spaces and found correlations with a visibility graph measure that has a relation to compactness. Other measures of isovists are investigated in relation to the stopping behaviour of people who are performing wayfinding tasks by Conroy and Dalton [5]. Several papers also identify which distinct types of isovists exist.

In the second part different generative systems for layouts are described. Schneider and König [2] [7] tried different approaches which includes algorithms that generate layouts with regards to isovist properties as well as algorithms that use isovist properties only after the general layout was created in order to finish and refine it. The commonly created pictures of an algorithm that places buildings in environments which they included in their paper can be compared to common patterns that were created with our approach.

3.2 Isovist properties and human behaviour

At first it is described how Benedikt [8] connected isovist properties with the information available at the viewpoint and the possible exposure of the observer to other persons. Here isovists are also related to private and public space. Afterwards Batty's analysis of isovist fields in different environments is summarised [3]. He found that there are two main types of isovists and he also points out that the placement of objects in the field of vision is more relevant than their concrete shape, which is an important statement since placing objects is the goal of this work. The section continues with a description of Conroy's and Dalton's experiments on the stopping behaviour of people who are solving wayfinding tasks and its correlation with isovist properties [5]. Turner et al [4] created visibility graphs according

to isovist properties and calculated the cluster coefficient values for two buildings. Their analysis show a correlation between the measure and the use of spaces as private or public places. Since the typical shapes of isovists for high and low cluster coefficients match with the shapes for high and low compactness values their results are also relevant for the meaning of compactness values.

Benedikt's information model

With regards to behaviour in space and perception of space Benedikt [8] points out that for analytical questions one could assume that in the unblocked space of a region equally information-giving events (or objects) are uniformly distributed. In this case the area A_x of the isovist at vision point x is proportional to the amount of available information at x. Similarly in an alternative scenario the information could be uniformly distributed on real surfaces rather than in unblocked space, in which the real surface perimeter would be the relevant measure. With an example of a region in which a long narrow space joins a large open space Benedikt demonstrates that the path which persons walking from the narrow space in the open space are following, determines how sudden they will experience an increase in available information. Figure 3.1 displays Benedikt's example which shows three different paths for walking around the corner and the speed of increase in visible area for each path. A person taking path $\Pi_{1,2}$ would experience a sudden information rush while taking path $\Pi_{3,4}$ or $\Pi_{1,4}$ would result an a more gradual information increase. Pedestrians who are rounding a street corner might choose a path similar to $\Pi_{1,4}$ because of a desire to control the speed of informational increase in their vision field.

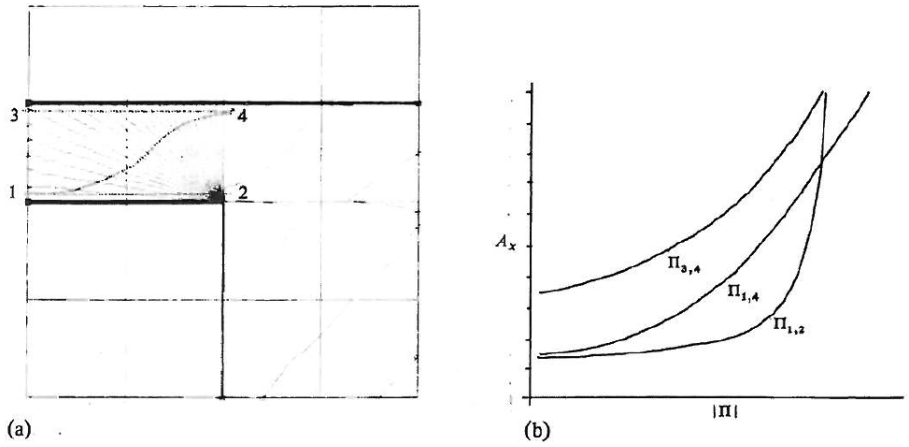
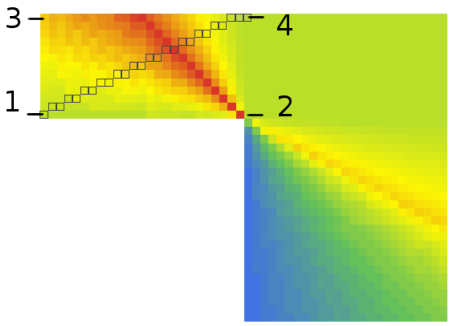


Figure 3.1: Example of a narrow space joining an open space and the information / area that can be seen over time when following different paths to the joining point (figure from [8]).

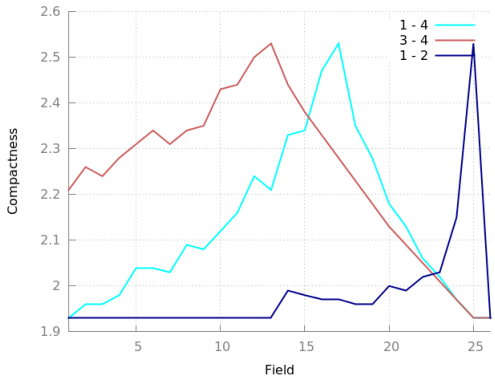
These considerations about available information are connected to the classification of space into private and public areas. Expanding the concept of equally distributed information, isovist measures not only describe the amount of information available at a vantage point, but also the potential amount of people to whom the observer at that point is exposed to. In several situations people typically wish to have access to a maximum amount of information while being minimally exposed themselves. In order to analyse the exposure extent it is necessary to consider not only the shape of the isovist, but also the position of the view point in the isovist. For this purpose Benedikt refers to the skewness of the distribution of radial length, which tends to be positive for view points which are close to real surfaces or situated in corners. Therefore high area and skewness indicate locations that offer a good view but low angular exposure. As examples or real-life scenarios in which the analysis of isovist values could be relevant Benedikt mentions the selection of a restaurant table against

a wall or prevention of crime incidences, since they seem to happen in places with low area and occlusivity [8].

The compactness does not relate to the position of the vision point inside the isovist or its distance to surrounding walls. Also, the perimeter that is used to calculate the compactness does not only consist of the real surface perimeter but also the occlusivity border. However, as mentioned by Turner et al, moving from a road junction means that the visibility of many previously seen areas will be lost [4]. So the compactness measure relates to how much the field of vision and therefore also the visible area will change if the observer moves from the vision point and therefore can also relate to how much information is lost or gained by that movement. Conroy and Dalton [5] observed that people pause after entering a larger open space as well as at road junctions. This could mean that people tend to pause after a significant change of compactness of their field of view, which might be connected to people rounding street corners. They might try to control the speed of informational change in order to avoid a sudden overload with new information which would result in the desire to stop in order to process it. Figure 3.2b shows the compactness map for a similar environment like in Benedikt's example, points 1, 2, 3 and 4 as well as a marked path from 1 to 4. Figure 3.2a shows the development of compactness on path 1-4, 1-2 and 3-4. Please note that path 1-4 is not exactly the path $\prod_{1,4}$ in Benedikt's example but a similar one. The graph looks different than for the area since the compactness not only grows but varies through the paths. It still shows that the change of compactness is very abrupt on path 1-2 and more gradual on path 1-4.



a) Compactness Map of an example that relates to Benedikt's area information flow example. The path 1-4 is marked by light borders around the cells used for the calculation of the compactness development graph in b).



b) Compactness development graphs for the straight paths from 1-2 and 3-4 as well as the path 1-4 that is marked in a).

Figure 3.2: Comparison of the area / perimeter ratio map and the compactness map.

Batty's exploration of isovist fields

In his work about the exploration of isovist fields Batty [3] uses two measures of shape. One is the compactness Γ_i which is defined as the ratio of the average to the maximum radial length of the vantage point i. Γ_i is one if the isovist is a circle and i situated at its centre and tends to zero if the isovist is a straight line and i located on one of its ends. The other measure Ψ is also one for a circle and zero for a straight line isovist and if squared it is the complement of Benedikts and our compactness measure. Batty refers to Ψ as shape index, convexity index or cluster index.

Batty computed the isovist measures for different shapes and layouts and analysed their distribution and correlations. It might seem confusing that the convexity index Ψ is called cluster index, like the cluster index γ_i which he also mentions earlier in his paper. But as Batty writes that he concentrates on Γ_i and Ψ it seems save to assume that with "cluster index" he is referring to Ψ in his following analysis. He states that isovist fields can be separated into categories that are based on key elements of urban and architectural morphology. He suggests two types: long narrow elements like streets and smaller compact ones like rooms. He also observed that the convexity index of long streets is low (which corresponds to a high compactness value for our measure) while squares have a greater convexity index (which corresponds to a low compactness value for our measure). These types match our observation of emerging configurations for maximum or minimum compactness placements.

His examples consist of two simple geometrical environments and three worlds that are based on real world layouts. His analysis shows a high correlation between area, perimeter and either average or maximum radial length. In most of his worlds area, perimeter are bimodal[1] or trimodal[2] and average radial length was found to be bimodal in two worlds as well. These indicate that there are two or three different types of isovists. However, in two of his real world examples no bimodalities were found. Batty points out that this fact implies simpler patterns, but also a greater complexity in that the patterning of spatial objects in the visual field takes on more significance than their shape in real world environments. He took notice that „the actual physical morphology of such complex urban building and streetscapes cannot best be measured by the geometry itself but is more likely to be represented by the visual 'objects' or spaces which emerge as a result of this geometry". This is an important statement since the goal of this thesis is to development an algorithm for placing such objects in an environment.

Batty simulated a walk through a street-environment in which the isovist properties were recorded for every pixel. He notices that the clustering index „picks up all the local detail associated with the pixelation of the street map". As we also demonstrated in the background chapter the compactness of shapes is greatly influenced by the shape that was chosen as the representative shape in concrete space.

Conroy's and Dalton's analysis of stopping behaviour

Conroy and Dalton [5] investigated the behaviours and actions of people that navigate through space. They used isovist measures to analyse the points at which people stop. Several measures they used can be linked to the compactness measure used in this work. They designed a series of virtual worlds through which people moved while trying to perform a wayfinding task. In the first part they compared the movements of people in virtual worlds to movements in the real world during which they demonstrated powerful analogies between the two. They then performed analysis on the gaze behaviour that was gathered while people moved through seven virtual worlds. Their investigation included the calculation of peoples pause points and the examination of isovist data of pause points. In their worlds they filled the free space with a grid of points, whose spacing was set to approximately four meters. Every point then was considered as a possible view point for which the isovists were created and their attributes stored. View points on which people stayed for more than a threshold time (0.5 seconds to 5 seconds, depending on the world) was identified as a pause point. They compared the isovist data of the pause points of one world to the mean values of the isovists of all possible view points in that world in order to identify the isovist measures that are relevant for peoples stopping choices.

One measure is the ratio of area / perimeter, which is the highest for a circle and drops as the circle gets deformed and it's perimeter increases. Therefore the area / perimeter ratio

[1] Bimodal means the histogram shows two modes which are recognisable as two distinct peaks.

[2] Trimodal means the histogram shows three modes which are recognisable as three distinct peaks.

can be directly linked to the compactness measure. The more spiky an isovist the higher it's compactness value and the lower it's area / perimeter ratio. In the first world, the tate gallery, people paused at locations with a area / perimeter ratio that is relatively low in the world and high perimeter value, which characterize junctions with longer lines of view. In the second world, world B, there is also a high correlation of pause points and area / perimeter ratio, but in this world people stop in areas where the ratio is higher than on average, which is the case for more rounded and open space. However, like in the tate gallery, the pause points again show a higher than average perimeter value and longer than average maximum perimeter length.

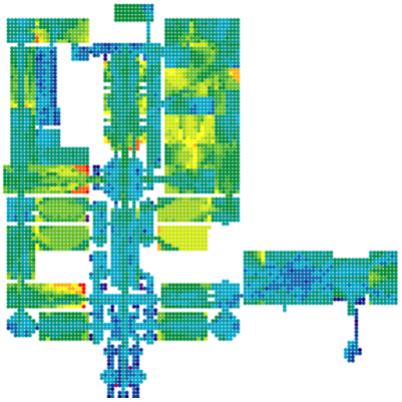
Another measure that indicates how spiky an isovist is the deviation of all radial lengths of an isovist. If the isovist is spiky, which also means it has a higher compactness, the deviation is higher. Conroy and Dalton put an overview about relations of measures into their paper, which shows an r-squared value of 0.702 for deviation and area / perimeter ratio, which indicates their correlation. In several worlds the deviation of radial length was significantly higher at pause points, as well as the maximum radial length of the isovists on the stopping points [5]. These two measures are also highly correlated according to Conroy and Daltons findings (r-squared value of 0.821).

As pointed out by Turner et al in [4] Benedikt and Burham (1985) showed that the variance of isovist radials and the perimeter also influence the perception of the size of a space.

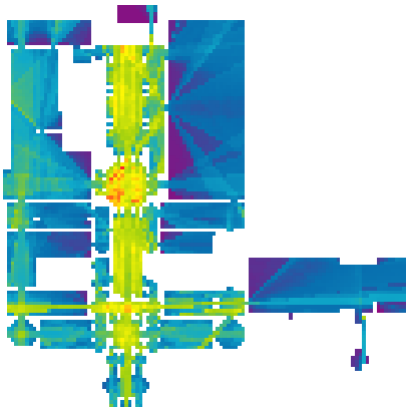
Other properties which characterised pause points are a large isovist area and the distance to occluding surfaces. Conroy and Dalton concluded that people pause in strategically locations which offer maximum visual information like long lines of sight and large isovist area. Since pausing was observed at road junctions as well as after people entered a larger open space, we may deduce that either areas with a larger than the world's average compactness as well as areas with smaller than the world's average compactness or a significant change of the compactness influence the stopping behaviour of people who perform way-finding tasks.

In order to demonstrate the connection between area / perimeter ratio and compactness visually, two example worlds were picked from Conroy and Daltons paper. Figure 3.3a shows the heat map of the isovist area / perimeter ratio of the tate gallery in which the lowest values are marked by blue colours and highest values by red colours, the white areas are blocking obstacles. In comparison the compactness heat map is displayed to the right in Figure 3.3b. Its colour key was chosen according to the colours displayed by Conroy and Daltons picture, but the scaling of values and hues is different. The lighter colours of right picture in the middle axis resemble the high compactness because of the spiky isovists in that area. On the left side the colours are darker because the spiky isovists have a high perimeter and low area value which results in a low area / perimeter ratio. Some interesting details show the strong relationship of the two measures. Note for example the red corners of high area / perimeter ratios in two rooms on the left side of the gallery. Those corners are notably darker than the surroundings in the compactness map.

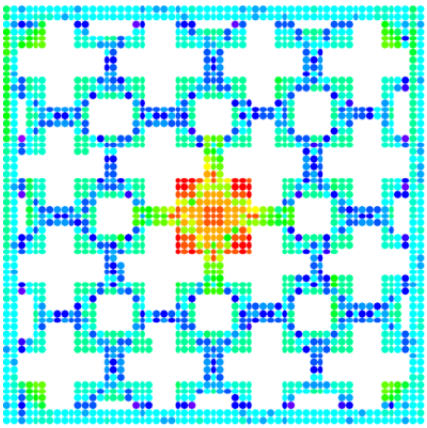
Figure 3.3c and 3.3d show Conroy and Daltons area / perimeter ratio map of their world B and in comparison the compactness heat map. Both maps show the same patterns. The inner part of the corners show a lower compactness and respective higher area / perimeter ratio. On the contrary, the occurring pattern that seems to run along the shortest pathway between the openings has a lower area / perimeter ratio and a higher compactness. Although the two measures are strongly connected, they are not proportional. That means that the point with the lowest compactness value is not necessarily the point with the highest area / perimeter ratio or vice versa.



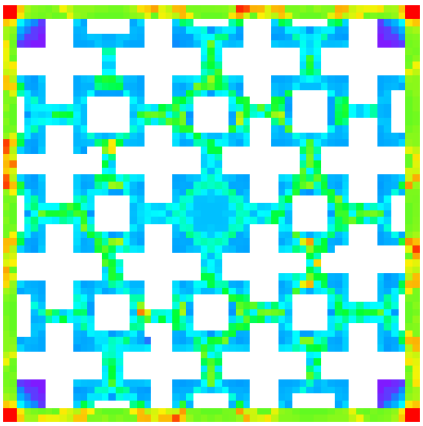
a) Conroy and Daltons heat map of the isovist area / perimeter ratio of the tate gallery [5].



b) Heat map of the isovist compactness of the tate gallery.



c) Conroy and Daltons heat map of the isovist area / perimeter ratio of world B [5].



d) Heat map of the isovist compactness of world B.

Figure 3.3: Comparison of the area / perimeter ratio map and the compactness map.

Turner et al on properties of isovist visibility graphs

Turner et al [4] point out that isovists have only been used in a limited number of studies for architectural analysis. They see one reason for this matter in the fact that isovists measure only local properties of space. Another issue would be that Benedikt did not state any useful instructions on how to interpret the results of his isovist measures. In order to counteract these barriers, they introduce broader methodology, which is sensible to the relationship of visual characteristics between locations and can be interpreted with regards to social relevance.

By using visibility graphs derived from isovists, local and global spatial measures can be obtained. A visibility graph is a graph that is based on the information which locations are mutually visible. For the construction of such a graph an appropriate set of isovists has to be selected. Since practicality has to be considered, some compromises have to be made. Try to select a set of view points which describe the space almost completely. One possibility, as the authors state: “the most obvious approach”, is to choose view points throughout the environment at a regularly spaced interval. They chose a “human-scale” grid with spaces that measure about one metre.

Two relationships between view points were used for the generation of the isovist graph:

- First-order relationship: this relationship exists between all view points which are mutually visible. A and B have a first-order relationship if point A is included in the isovist of point B and vice versa.

- Second-order relationship: A and B have a second-order relationship if there exists a point C that is visible from point A and from point B.

The graph of the first-order relationships is the visibility graph of the environment, while the second-order relationships form an isovist intersection graph. Since the second can be generated from the first, the authors chose to concentrate on the visibility graph.

For the analysis of the graph they used the following three graph structural measures:

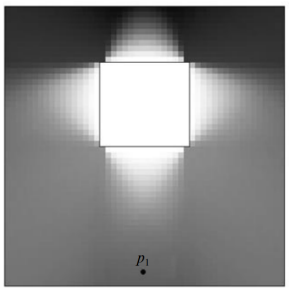
- Neighbourhood size: a local property which measures the amount of vertices directly connected to a vertex. So all vertices in the isovist which are visible from a vertex make up it's neighbourhood.

- Clustering coefficient: a local property which is calculated by dividing the actual number of edges between the vertices of the neighbourhood by the number of possible edges.

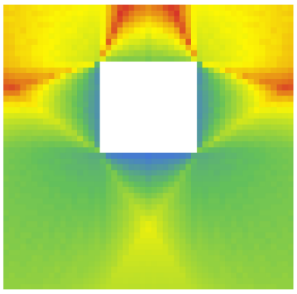
- Mean shortest path length: a global property which is the average of the lengths of shortest paths from a vertex to every other vertex in the graph, where the shortest path length for two vertices is the the least number of edges that have to be traversed to move from one vertex to the other.

Since we are working with the visibility graph, an edge between two vertices of a neighbourhood means that they are mutually visible. A high number of edges between the vertices of a neighbourhood therefore means that many points of the neighbourhood are mutually visible. This is the case for isovists whose shape is almost a convex polygon, which is circle-like. The clustering coefficient is tending to one in this case and tending to zero if the shape is not convex at all. Turner et al note that if the clustering coefficient is tending to zero the generating location of the isovist is some kind of junction. Moving from a location like that will result in a loss of visibility of parts of the currently visible area. It can be concluded that the clustering coefficient also indicates how much of the currently visible area an observer at that point will loose or keep if he moves away from it. The paper suggests that therefore the clustering coefficient is „related to the decisionmaking process in way-finding and navigation and certainly marks out key decision points within complex configurations“ [4]. Since they mention that convex shapes have a high cluster coefficient and shapes that are not convex have a low cluster coefficient and resemble junctions, we may conclude that this measure is somehow connected to the compactness measure. High compactness values should be present were the cluster coefficient is low and the convex shapes with high cluster coefficient probably have a low compactness value. However, unlike the compactness, the cluster coefficient of the points within an isovist is also influenced by the space that lies outside of the isovist. The authors point out that this behaviour resembles that people traversing a space can still perceive the space fully if they move, even if there is a pillar in the middle of the room. Figure 3.4a, which contains one big obstacle, and Figure 3.4c, which contains one small obstacle, show the influence of the object size on the cluster coefficient. High values are displayed whiter than low values. In 3.4c the arrow is showing that points from one side are visible from the other side. If the space in the shadow of the obstacle would be occupied by an object the view lines from one side to the other would be blocked, which would noticeably influence the cluster coefficient. Figure 3.4b and 3.4d indicate that the compactness heat map shows similar patterns like the cluster coefficient for the points P1 and P2.

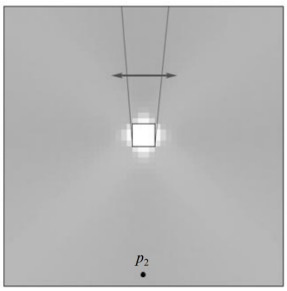
In order to analyse the clustering coefficient values Turner et al calculated a grid of 1.0m on eye level for two buildings and produced one visibility graph per house (by linking two



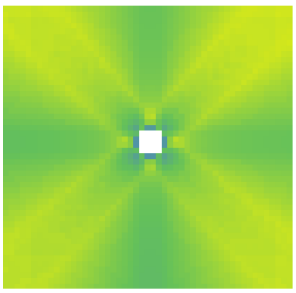
a) Cluster coefficient for point P1 in environment 1 [4].



b) Compactness heat map for environment 1.



c) Cluster coefficient for point P2 in environment 2 [4].



d) Compactness heat map for environment 2.

Figure 3.4: Cluster coefficient for the isovist of a point in an environment with a big and a small object in comparison to the compactness map of both of the environments.

levels of one house via the stairwells). They discovered that private spaces like bedrooms or study rooms are highly clustered and that social spaces, like living rooms, offer more spiky fields of view which hint at the range of private spaces without intruding them. Their clustering coefficient is therefore tending to 0 [4]. Private spaces, which are rectangular and circular rooms, are characterised by a low compactness value and since social spaces offer spiky fields of view, they are indicated by a high compactness value. This finding matches Benedikts theory that isovist properties describe the potential amount of people to whom the observer is exposed.

3.3 Generative approaches in architecture

This section describes systems that automatically generate layouts. Schneider and König used several different generative approaches. One is to generate room layouts with a dense packing algorithm which follows functional criteria and afterwards adds doors to the generated rooms with regards to isovist properties [2]. They also developed an algorithm that uses isovist properties as target functions from the start and creates floor plans by activating and deactivating predefined grid lines [2]. And their work includes an algorithm that places buildings in an environment to generate urban layouts automatically [7]. They used different fitness functions that consider isovist properties, which also included the compactness. Their work therefore is very similar to ours. But they placed buildings, which are quite big objects compared to our smaller but more plentiful objects.

Schneider and König setting doors by isovist properties

Schneider and König [2] investigated some existing systems for layout generations that use different underlying mechanisms. None of the systems they analysed were currently used by practising architects and with their system they tried to solve some problems that prevent architects from using those systems. They saw one problem in the fact that after the architect formulated his target function most programs searched until they find an acceptable solution or the search process is cancelled. In that case the architect cannot influence the search process. They therefore wanted to design a system that is better integrated into the design process. They argue that designing is an iterative process and therefore a design system shall be able to react to changing problem definitions. Also, the presented solutions should be distributed evenly on the pareto front, which contains the solutions with the best compromises, in order to show all relevant solutions to the user. The user shall then decide on his own which one he wants to work with. For the architect it is of utter importance to see the impacts of different parameters immediately. This provides an instant feedback on effects of specific criteria. The systems needs to react fast.

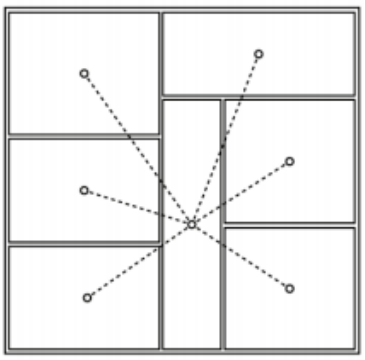


Figure 3.5: A floor plan generated by the dense packing algorithm (figure from [2]).

One method to generate floor plans with regards to isovist properties that Scheider and König explored refines a configuration that was developed with dense packing following functional criteria. The input of the dense packing algorithm is a buildings outline, the sizes of the rooms which shall be placed (in m^2) and minimum and maximum room dimensions. The sizes of the rooms have to add up to the buildings size. Overlaps or spaces between the rooms should be avoided. The dense packing algorithm minimises the sum of all overlapping areas and of the occupied space outside the buildings outline. The rooms are positioned and their length and height is adjusted. The output of the dense packing algorithm consists of the positions and dimensions of rooms in the building, an example is shown in Figure 3.5. In the refining step the isovist criteria came into effect. The average value of the area of the isovist fields (Mean Isovist Area) was maximised by positioning doors on the rectangles. Figure 3.6 shows the heat maps of some solutions using the isovist area values on the map cells.

A graphical user interface enables the user to rate the solutions and configure the layout problem. Different visualisation methods were used to display the floor plan as well as it's properties and the configuration attributes. Since an iterative system was used changes of the problem definition can have an immediate effect. One possibility is to change the floor plan directly. The system also offers the possibility of indirect manipulation by adjusting the generative rules or evaluation criteria. Although this approach offers an effective generation of layouts for form and function of rooms, the possibilities for the isovist properties to influence the form of the layouts are restricted.

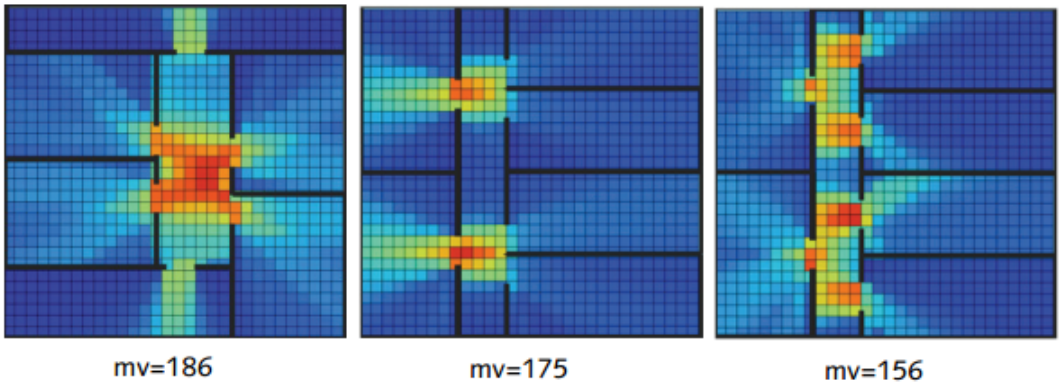


Figure 3.6: Heat maps of isovist area values of solutions generated by placing the doors on results of the dense packing algorithm (figure from [2]).

Schneider's and König's grid acitivation system

Another method which Schneider and König used to generate floor plans used a model of a uniform grid in which horizontal and vertical lines can be activated. The evaluation considered only one fixed isovist point which has to be set before the optimisation process. The fitness function minimises the sum of the deviation of area, perimeter and occlusivity of the isovist at that fixed point from the set of target values tA (target area), tV (target perimeter), tQ (target occlusivity). The three deviation values in the fitness function are also normalised and can be weighted by individual weights wA, wV and wQ:

$$f1(x) = norm(|Ax - tA|) * wA + norm(|Vx - tV|) * wV + norm(|Qx - tQ|) * wQ \quad (3.1)$$

Using mutation and recombination individual lines are activated and deactivated until a solution is found that satisfies the fitness criteria sufficiently. Examples of solutions created for different target value combinations are shown in Figure 3.7. The floor plans that evolved show the big influence that the isovist attributes have on the layout. The authors extended the system and added the overlapping area of isovists as another fitness criteria in order to analyse if topological relations evolve between the different areas of the floor plan. They defined three view points for which the program should develop suitable isovists. They set the overlapping area for isovist $I1$ and $I3$ as well as $I2$ and $I3$ to 300, while the isovists for $I1$ and $I2$ should not overlap at all. However, the results showed that this criteria is not enough to describe the relations between rooms. The developed rooms where not closed, therefore rooms which where defined with non-overlapping isovists, where still accessible to each other [2].

Schneider and König on the generative potential of isovist fields

In their in 2012 presented paper [7] Schneider and König used an inverse design process to automatically generate urban layouts. In their evolutionary algorithm individuals represented urban layouts that consist of a fixed number of rectangular buildings placed on a plot of land. The algorithm places and scales those rectangles while ensuring that they do not overlap and stay within the given boundary. To this end solutions which do not meet these constraints after the random placing, are corrected by special movements. Unreasonable solutions shall be avoided by setting minimum and maximum widths and surface areas for the buildings and a minimum and maximum coverage of the plot. Because of the computationally intensive and time-consuming calculation of isovist fields they made use of the graphical processing unit

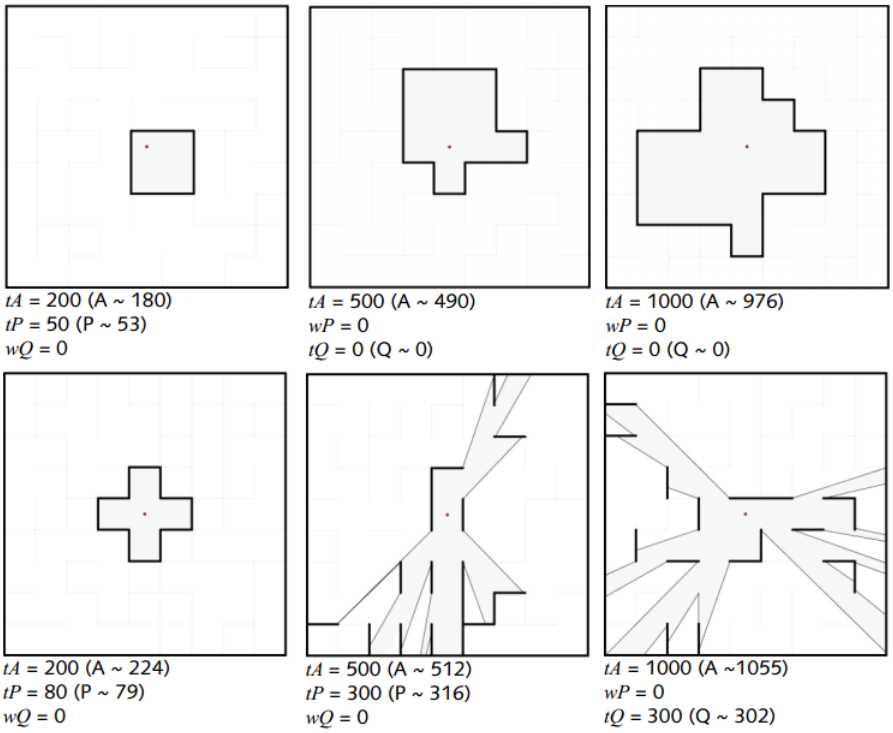


Figure 3.7: Solutions created by activating (/ deactivating) grid lines. Notice that some configurations set a target value of 0 for one of the sizes, minimising it, while others set its weight to 0, which leads to the target function ignoring that size (figure from [2]).

(GPU) for those calculations. The evaluation of an individual in the used test scenario with the GPU usage takes approximately 0.15s. The objective function minimises or maximises the average, minimum, maximum and standard deviation of the properties area or compactness. They generated configurations for the test instance for all 16 objective functions. They proved that it is possible to reproduce generated patterns based on specific visuospatial properties.

Figure 3.8 shows examples of the configurations their program created when the objective criteria was the compactness. Since they use the compactness measure between 0 and 1 like Batty [3] the results need to read with the conversion to our compactness measure in mind. In the following analysis references to their compactness measure will noted as Ψ and our compactness measure will be called N.

The minimisation of the average Ψ corresponds to the maximisation of the average N. The result for this case shows a configuration in which the buildings seem equally distributed throughout the environment and many points offer long views. On the other hand the maximisation of the average Ψ (and minimisation of the average N) results in one open space which is almost quadratic.

The minimisation of the minimum Ψ and therefore maximisation of the maximum N produces configurations with at least one point that offers a very non-compact field of view. The isovist with the minimum Ψ is marked in red in their image. The maximisation of the minimum Ψ corresponds to the minimisation of the maximum N and generates layouts with one or more open spaces. These results are similar to the ones from maximising the average Ψ.

The minimisation of the maximum Ψ, which means the maximisation of the minimum N, results in the buildings being placed a short distance from the outer wall and creates configurations which do not offer compact fields of view. The maximisation of the maximum Ψ on the other hand leads to configurations with at least one enclosed square space. As can be seen in 3.8 this square space is quite big. So far all of their generated examples match

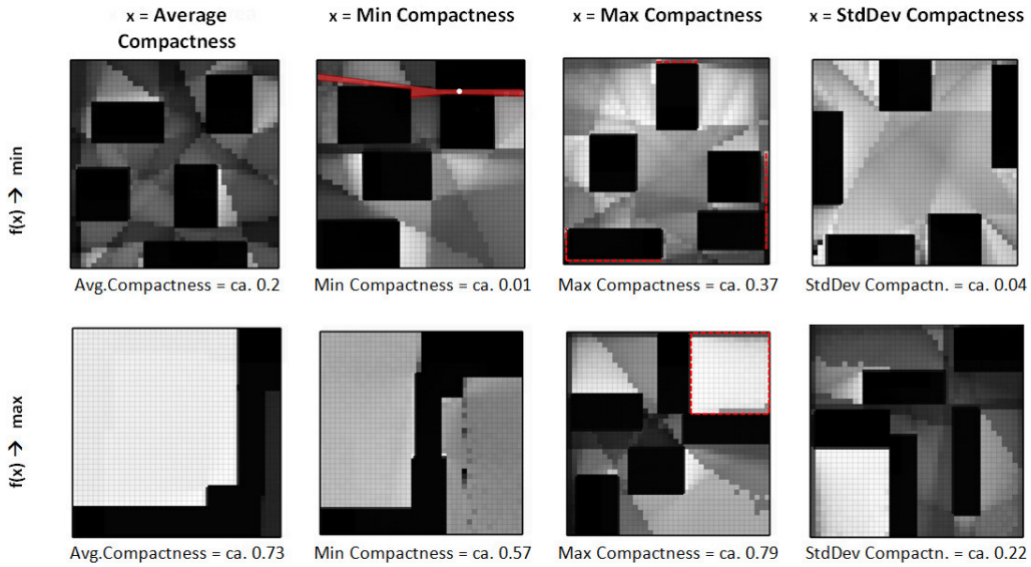


Figure 3.8: Examples of generated urban layouts for every compactness target function that Schneider and König analysed (figure from [7]).

with the general patterns that emerged in our experiments. However, the formed squares when minimising the minimum N are often of size 1x1 and therefore a lot smaller than the square in the Ψ example. This is probably due to their buildings being a lot bigger than the objects we place.

The minimisation of the deviation of Ψ, which also minimises the deviation of N, generates layouts which either have a low compactness at all view points or a high compactness at all view points. The maximisation of the deviation of Ψ and therefore also the maximisation of the deviation of N results in configurations with points with high compactness values as well as points with low compactness values. Again the size of the area with the high Ψ is different to the size of the squares that were produced for N when smaller objects were used.

3.4 Conclusion

The introduced experiments show that isovist properties indeed have an influence on peoples behaviour. There are several different isovist properties and ways to represent and evaluate them and many were found to be strongly connected. They can indicate how much or in which form information is available at their view point and relate to the exposure of the observer to other people (see *Benedikt's information model*). This leads to the fact that they influence at which places people stop or which paths they take through an environment. Interestingly not only the absolute value of compactness seems to matter but also a significant change in compactness compared to previously traversed places can issue people to stop (see *Conroy's and Dalton's analysis of stopping behaviour*). Isovist properties can characterize private or public space and indicate the usage form of a space. There seem to be two types of isovists. Places with high compactness are narrow streets or view points with spiky isovists and indicate public space (see *Turner et al on properties of isovist visibility graphs*). Places with low compactness are more compact and rounded places which do not offer many long view lines and they indicate private space. Batty specifically points out that the patterning of spatial objects in the visual field is more important than the shape of the objects (see *Batty's exploration of isovist fields*). He also noted that the measure of clustering index, which relates to compactness, is influenced by the discretisation of shapes, which we demonstrated in Section 2.3.

The generative approaches that were introduced show that isovist properties can be used to automatically generate layouts depending on isovist properties. But it seems to be difficult to provide the algorithm with enough constraints to generate reasonable solutions that are practically useful without restricting it too much to allow a wide range of possible layouts (see *Schneider and König setting doors by isovist properties* and *Schneider's and König's grid acitivation system*). One of the most related works is probably the work of Schneider and König who developed an algorithm that automatically generates urban layouts by placing buildings in specified environments (see *Schneider and König on the generative potential of isovist fields*). This is similar to our goal of placing objects in environments, although our objects are significantly smaller and more objects are placed. They also provided pictures of solutions for every compactness target function which can be compared to our results. When comparing the areas with smallest compactness in their results with the area with smallest compactness in our results it is noticeable that their areas are bigger. This is probably because of the difference in object size and suggests that an awareness of this size issue is important to keep in mind in order to guide the generation to the desired results.

CHAPTER 4

Evolutionary Approach: Introduction

4.1 Problem definition

The specific problem which is considered in this thesis is the placing of objects in space. This relates to real life problems like the placing of exhibition objects in museums. One of the main goals is to develop an algorithm which places predefined obstacles in the input environment. The resulting placement shall provide vision points that offer particularly high or low compactness values. The algorithm needs to solve an optimisation problem whose target function considers the compactness and therefore the area and perimeter of the isovists in the environment. The solution space for this problem contains every possible combination of obstacle placings in this environment. The complete exploration of the solution space is not feasible because of its size and the corresponding computational complexity. Therefore a heuristic approach is necessary. As DeLanda already pointed out, EAs are useful where not all possible configurations can be explored in advance [9]. As mentioned in Kremlas [2] EAs are also suitable for the explorations of problems whose solution should contain properties which are not explicitly stated as input criteria. It is more important that planners are provided with different creative solutions than to find an optimal solution in a strongly mathematical sense. Evolutionary algorithms use populations consisting of multiple current candidate solutions and incorporate a random component in variation operators for deriving new solutions. They can find several different good solutions and are therefore well suited for the task.

The next section offers a general introduction to evolutionary algorithms and their most common properties. Afterwards the specific approach for the introduced problem is described.

4.2 Background to evolutionary algorithms

Evolutionary Algorithms use the concept of Charles Darwin's theory of biological evolution to find solutions to optimisation problems [13]. To simulate the development of different generations such an algorithm runs through several iterations. The current generation of an iteration is formed by a pool of individuals, called population. Every individual is represented by its genes, usually a string or vector of fixed length in which every gene has a specific location. This representation is called genotype and can be translated into a corresponding solution to the problem, called phenotype [14]. The translation is called mapping [2]. For each phenotype the quality of the represented solution is evaluated and assigned a fitness value. Individuals with higher fitness values denote better solutions. To simulate the Darwinian „survival-of-the-fittest“ this value is used in a selection process in which individuals

of the current generation are chosen as parents for the next generation. Simple copying, crossover and mutation are applied to create new individuals [13]. Crossover creates genes for a child individual by combining some genes of one parent with some genes of another parent. Mutation refers to the random change of genes, like a bit flip or permutation [15].

Algorithm 4.1: Structure of a general evolutionary algorithm [15] [16]

```
init population;
repeat
    recombine;
    mutate;
    evaluate;
    select new population;
until stopping criteria met;
return best individual as the problem solution;
```

The code shown in Algorithm 4.1 presents the general template of any evolutionary algorithm. But for the concrete implementation there are a lot of possibilities to choose from. The representation of genotypes and phenotypes is only one decision which has to be taken [15]. In order to provide an overview of techniques, some possibilities are introduced here.

Population

The population holds the solutions of the current generation. While in a natural system the size μ of the population can vary, in most evolutionary algorithms the population size is fixed and the same in all generations. The number λ of children generated in each iteration may deviate from μ. The individuals that shall be part of the next generations population can be either chosen from λ - then the notation (μ, λ) is used, where $\lambda > \mu$ - or from $\mu \cup \lambda$ - then the notation $(\mu + \lambda)$ is used. In (μ, λ) algorithms it is possible that the best individual of one generation is worse than the best individual from the last generation. In $(\mu + \lambda)$ algorithms the selection pressure is higher [13]. The initial population is typically generated as random individuals [17].

Selection

An evolutionary algorithm provides two opportunities where a selection mechanism can be applied. First parents for recombination need to be chosen, this is called parent selection. After the recombination individuals for the next generations population are selected, this is called survival selection [17]. Different mechanisms can be applied for both. An important property of a mechanism is the selection pressure which expresses the probability of the fittest individual being chosen relative to an individual of average fitness. If this selection pressure is too high, the algorithm will converge towards a solution very fast and might get trapped with a suboptimal solution in a local optimum. On the opposite if selection pressure is too low, it might take very long to converge. Applying a selection mechanism for both selection steps increases the selection pressure and might lead to too much greediness. Therefore most evolutionary algorithms choose a fitness related selection mechanism for only one of the two opportunities [13] [15].

Some popular selection mechanisms are [15] [14]:

- *Truncation selection*: This is one of the simplest selection mechanisms. The individuals are ordered by their fitness values, the fittest ones are selected and the rest gets

truncated. Since only the best individuals get chosen, the selection pressure is very high.

- *Tournament selection (K)*: K individuals are chosen at random and only the best one is selected. This is repeated until μ individuals got chosen. The selection pressure can vary with the group size K.

- *Proportional selection*: The selection probability of each individual is its relative fitness.

- *Rank selection*: The individuals are ordered by their fitness values and their index in the ordering is used to calculate their selection probability.

Each approach has strength and weaknesses. One problem with the proportional selection is that if the population contains a small number of individuals that are much fitter than the rest of the population they get chosen often and reproduce quickly and prevent the algorithm from further exploring the search space. This situation often occurs during the first generations. The opposite problem emerges in the last generations: most individuals are similar and have a similar fitness value. Therefore their probability of getting chosen is very similar too. In order to counter those two effects scaling methods were developed. One of these methods is sigmascaling. It normalizes the fitness values relative to the standard deviation of all fitness values in the population. The scaled values are:

$$f'(x_i) = \begin{cases} 1 + \frac{f(x_i) - \overline{f}}{2\sigma} & \text{if } \sigma \neq 0 \\ 1 & \text{if } \sigma = 0 \end{cases}$$

where $f(x_i)$ is the fitness value of individual i, $\overline{f}$ is the average compactness and σ is the standard deviation [18] [19].

Reproduction

When a new generation is created, new individuals have to be derived from existing ones. One possibility is asexual reproduction, which means that an existing individual is cloned and a mutation operator is applied to slightly change the individuals genes. Another possibility is sexual production, which involves more than one individual as parent. As in natural systems, usually two parents are used [15]. The offspring creation by recombination can be classified into two groups: dominant or intermediate recombination.

In dominant recombination the offspring directly inherits traits of a parent individual while disregarding the other parents corresponding trait [16]. Standard algorithms perform a one-point-crossover, which involve only one crossover point. One child is created by concatenating the genes from the beginning of one parent until the crossover point and the genes after the crossover point from the other parent. The genes of the second child consist of the genes which were not used for the first child [14]. Let parent 1 be represented by j genes $g_1, .., g_j$ and parent 2 likewise by j genes $h_1, ..h_j$. If the crossover point is chosen to be between the genes d and $d+1$ where $1 \leq d < j$ then the genes of child 1 will be $g_1, .., g_d, h_{d+1}, .., h_j$ and child 2 will be described by the genes $h_1, .., h_d, g_{d+1}, .., g_j$. Figure 4.1 shows a visualised example.

If more than one crossover point is used, the genes are simply distributed alternatingly. Figure 4.2 shows an example of a two-point-crossover. Another possible approach is uniform crossover [14], in which for each gene it is randomly decided from which parent it shall be copied.

Intermediate recombination takes into account the values of all parents in order to determine a child's characteristic. In the simplest case the mean value is calculated [16].

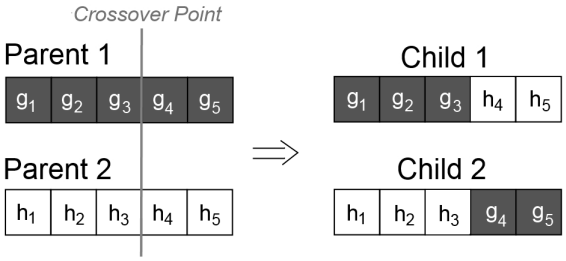


Figure 4.1: Example children generation by one-point-crossover.

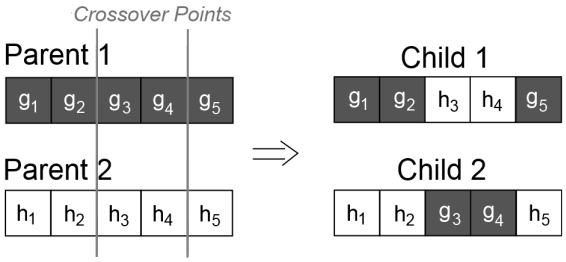


Figure 4.2: Example children generation by two-point-crossover.

Since asexual reproduction produces individuals more similar to their parents it is more a local search operator. By contrast, sexual reproduction tends to produce individuals which vary more from their parents and therefore it is a global search operator [15].

4.3 Evolutionary algorithm variants

Different types of EAs were developed. The categorisation of an EA in a specific variant group got harder as the EA variants were expanded and the characteristics of some variants started to overlap. In this section the main variants that were developed are introduced in order to provide an overview about EA techniques, their development and diversity.

Genetic algorithm

Genetic algorithms (GAs) were mainly developed by Holland and his students from the 1960s to the 1980s. Further improvements were made by De Jong and Goldberg [15] [13]. In the very early GAs the emphasis was on recombination rather than mutation. Today it is mostly on the combined interactions of recombination, mutation and selection.

The parent selection process is emphasised in GAs. It is seen as a process with two stages. At first the parent individuals are chosen and form an intermediate generation. Recombination and mutation are then applied to this intermediate generation in order to create the children which form the new generation. The selection usually takes into account the fitness of individuals. Commonly used selection mechanisms are for example tournament selection or fitness proportionate selection with scaling [15]. The new generation may replace the population of the old generation completely or just some individuals are replaced by new ones [13].

The used data structures often require a mapping from genotype to phenotype. In early GA applications the individuals were represented by bit strings since they represent the problem with the largest number of smallest possible components. We will discuss some characteristics of bit representations here, although today any suiting data structure may be used.

The most common mutation form for bit strings is the flip of one bit. When real values should be represented by bit strings, one has to decide which precision shall be used. The number of bits per parameter can have a big impact on the performance of the algorithm [15]. When deciding for a data structure for the individuals some desirable characteristics should be kept in mind. One important property that representations should possess is that similar phenotypes should be represented by similar genotypes [20]. Numbers encoded by standard binary representation lead to the problem that numbers which are close to each other in the real domain may have bit encodings with a large Hamming distance. Consider for example the four bit standard binary representation of 7, which is 0111 and of 8, which is 1000. Although the phenotypes 7 and 8 are very similar their genotypes have a Hamming distance of 4, which is the maximum in this four bit representation. In order to solve this problem Gray codes were introduced as an alternative to standard bit representation. There exist different Gray codes but all fulfil the requirement that adjacent numbers differ in only one bit in their bit represenation [20].

Evolution strategies

Evolution strategies (ES) were invented by Rechenberg and Schwefel for shape optimisation in the early 1960s. ES are usually used for continuous parameter optimisation, such that a function $\mathbb{R}^n \to \mathbb{R}$ needs to be minimised. Each object variable is represented by a floating-point variable, which means that the genotype space is identical to the phenotype space $\mathbb{R}^n$ [17]. Today the original ES version by Rechenberg and Schwefel is called (1 + 1)-ES [15]. It uses only one individual as population and only one child is produced. Algorithm 4.2 outlines this specialised algorithm. The vector $\overline{x}^{(t)}$ of size n represents the current solution at generation t. The fitness values are calculated by fitness function f and the fitness of the solution $\overline{x}^{(t)}$ is compared to the fitness of its mutated version $\overline{y}^{(t)}$. If $\overline{y}^{(t)}$ reaches a higher fitness than $\overline{x}^{(t)}$ it is used as the individual for the next generations population. Otherwise $\overline{x}^{(t)}$ will be utilised again in the next generation.

Algorithm 4.2: (1 + 1)-ES after example from [17]

input : fitness function f
output: best individual that was found
$t \leftarrow 0$;
create random starting individual $\overline{x}^{(t)} \in \mathbb{R}^n$;
repeat
  $\overline{y}^{(t)} \leftarrow \text{mutate}(\overline{x}^{(t)})$;
  if $f(\overline{x}^{(t)}) \leq f(\overline{y}^{(t)})$ **then**
    $\overline{x}^{(t+1)} \leftarrow \overline{x}^{(t)}$;
  else
    $\overline{x}^{(t+1)} \leftarrow \overline{y}^{(t)}$;
  $t \leftarrow t + 1$;
until *stopping criteria met*;

The population in this ES consists of only one individual so no recombination is used, mutation is the only variation operator. In order to mutate $\overline{x}^t$ a random number is added to each of its n components. Those numbers are drawn from a normal distribution with mean zero and standard deviation σ. Therefore small changes are more likely than big ones. σ is also called mutation step size since it determines to which extend the values of $\overline{x}^t$ are modified. Rechenberg came up with the 1/5 success rule. It states that the ratio of successful mutations to all mutations should be 1/5. If it is smaller than 1/5 then the search is not concentrated enough and σ has to be decreased. If it is greater than 1/5 then the search is

too concentrated and σ needs to be increased. Adjustments are usually executed periodically, for example after m iterations [17].

ES were further improved and extended. Rudolph lists the following characteristics that specify an EA as an instance of an ES today [15]:

1. Parent selection is unbiased: parent individuals are drawn randomly with a uniform distribution.

2. Survival selection is deterministic: only the best individuals survive.

3. Mutation is parameterised and therefore the operator changes its properties during the optimisation process.

4. Individuals consist of solution characteristics as well as strategy parameters.

As mentioned in [16] for the (1 + 1)-ES only one set of strategy parameters is needed, therefore it may be added to the algorithm itself rather than the individual. Point 3. and 4. refer to the ability to adapt the parameters for mutation during runtime, this characteristic is called self-adaptation.

A generic ES is denoted as $(\mu/\rho, \kappa, \lambda)$-ES where μ describes the population size and λ the number of children that are generated each generation. ρ is the number of individuals that are used as parents for the creation of one offspring by recombination. The additional notation κ refers to the maximum live time, or age, of each individual. It replaces the „plus“ (survivors are chosen from $\mu \cup \lambda$) or „comma“ (survivors are chosen from λ only) notation where „plus“ corresponds to $\kappa = \infty$ and „comma“ to $\kappa = 1$. The age attribute was established 1995 and is part of an individual.

The generic framework for a $(\mu/\rho, \kappa, \lambda)$-ES is demonstrated in Algorithm 4.3. The population at generation $t \geq 0$ is denoted by $P^{(t)}$. An individual $p \in P^{(t)}$ contains the object variables $\overline{x}$ which represent a potential solution and a finite set of strategy parameters Ψ. The *variate* step represents the recombination as well as mutation steps. Although early theoretical publications put more emphases on mutation, like the (1 + 1)-ES, recombination is an important part of an ES as well [15]. For ES global recombination is common, where new parents are drawn for each gene. Different recombination strategies may be chosen for the object variable and the strategy parameter part. For the object variables discrete recombination is recommended while for the strategy parameters intermediary is recommended [17].

Evolutionary programming

When evolutionary programming (EP) was originally developed it's goal was to create an artificial intelligence through the learning process of simulated evolution. In the classic form of EP finite state machines (FSM) were evolved which could be used as predictors [17]. FSM possess a finite number of internal states, process input symbols from a finite set and produce output symbols from a finite set. The set of state transitions and the input and output symbols specify the behaviour of the FSM [15]. Example tasks that were used for experiments are the prediction of the next input or the prediction if the next input number will be a prime number or not. The prediction accuracy of a FSM was used as its fitness value. Each FSM in the population was mutated once in order to create offspring. Recombination was not used. The best half of the union of the old population and the children was used as population for the next generation [17].

Another example of application of EP is the development of globally optimal strategies in two-player, zero-sum games in 1969. Later this application was extended to nonzero-sum games like pursuit evasion for aerial combat in which the application outperformed human subjects. EP were extended and improved and a variety of mechanisms is used today. Since the 1990s real value representations became more common and adaptive parameters for

Algorithm 4.3: $(\mu/\rho, \kappa, \lambda)$-ES based on [15]

input :
output: best individual that was found
initialize population $P^{(0)}$ with μ individuals;
foreach $p \in P^{(0)}$ **do**
  $p.\Psi.age \leftarrow 1$;
$t \leftarrow 0$;
repeat
  $Q^{(t)} \leftarrow \{\}$;
  for $i \leftarrow 1$ *to* λ **do**
    select ρ parents $p_1, ..., p_\rho \in P^{(t)}$ uniformly at random;
    $q \leftarrow variate(p_1, ..., p_\rho)$;
    $q.\Psi.age \leftarrow 0$;
    $Q^{(t)} \leftarrow Q^{(t)} \cup \{q\}$;
  $p^{(t+1)} \leftarrow$ selection of μ best individuals from $Q^{(t)} \cup \left\{p \in P^{(t)} : p.\Psi.age \leq \kappa\right\}$;
  foreach $p \in P^{(t+1)}$ **do**
    $p.\Psi.age \leftarrow p.\Psi.age + 1$;
  $t \leftarrow t + 1$;
until *stopping criteria met*;

mutation were used. Today the representation and mutation possibilities are chosen in a way that fits the problem that needs to be solved best [17]. Tournament or proportional selection were used in later experiments. In the canonical EP each parent generated λ offspring through mutation operators. In continuous EP new individuals are added to the population without iterative generations. Another expansion is the implementation of random extinction events [15].

Genetic programming

GAs use a fixed length string representation for individuals. For many applications this representation is too constraining and unnatural. This is also the case for the evolution of computer programs. Genetic programming (GP) solves this problem with individuals, which represent programs, of hierarchical structure and variable size. The most common and also oldest representation is the tree-based representation, which will be discussed further in this section [15]. But other representations were introduced as well. Examples of employed structures are linear genomes, graph-based genomes or grammar based representations.

A GP basically works like any GA, but the special representation form requires suiting recombination and mutation mechanics. The possible structures a GP can generate are defined by the trees that are constructable from a set of function symbols $\mathbb{F} = \{f_1, f_2, ..., f_n\}$, which are used at tree nodes, and terminal symbols $\mathbb{T} = \{t_1, t_2, ..., t_m\}$, which are used at the leaves. Each function operates on a fixed number of arguments, referred to as its *arity*. Each terminal symbol can be either a variable or a constant. An example of a tree is displayed in Figure 4.3.

It is good practice to make sure that the applied combination of $\mathbb{F}$ and $\mathbb{T}$ fulfils two specific properties. One is called *closure* and requires that every function symbol in $\mathbb{F}$ can take as any of its arguments any data type and value that may result from the evaluation of any terminal symbol or function. In many applications fulfilling the closure property is not straightforward. But in order to calculate the fitness value of a program it needs to be executed without abortion. In strongly typed GP each primitive contains information about its type and the types it can call. Functions are forced to cast their arguments into

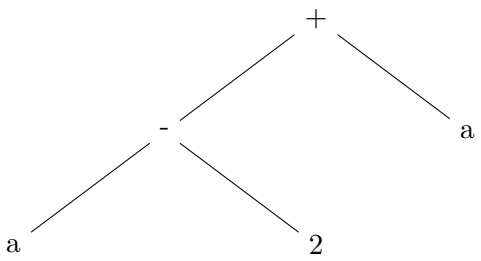


Figure 4.3: Example of a program tree that can be constructed with $\mathbb{F} = \{+, -\}$ and $\mathbb{T} = \{a, 2\}$.

appropriate types which enforces closure. The second property is *sufficiency*, which requires that the function and terminal symbols are able to formulate a solution to the problem.

When the initial population is generated, difference practices can be used to create a program tree. Trees are usually constructed starting at the root. In the most common methods a depth parameter d defines the maximum depth any tree within the GP can reach. In the grow method a tree is built applying the following steps:

1. The root symbol f is drawn with uniform probability from $\mathbb{F}$.
2. Let n be the arity of f. If f is at level $l < d - 1$, n nodes are selected with uniform probability from $\mathbb{F} \cup \mathbb{T}$. If f is at level $d - 1$ the nodes are drawn from $\mathbb{T}$ only. These n symbols are the children of f.
3. For each function symbol f_i within the n children the grow method is invoked recursively starting at 2. with f_i in the place of f.

With this method trees of different shapes and sizes are created. The full method chooses only symbols from $\mathbb{F}$ in depth smaller than d and only symbols from $\mathbb{T}$ in depth d. This approach results in full trees that reach the maximum depth in each branch. In order to reach greater diversity in the initially generated trees, the ramped half-and-half method was formulated. For each depths m from 1 to d a fraction of $\frac{1}{d}$ of the population is created with depth m. Half of each depth group is created by the grow method and the other half with the full method.

Crossover is usually handled by swapping subtrees. Standard GP crossover independently selects a random node in each parent. Usually internal nodes are chosen with a probability of 0.9 while any point is chosen with a probability 0.1. The chosen subtree in parent one is deleted and the chosen subtree of parent 2 inserted at the deletion node. Some implementations create only one tree while others create a second child by replacing the chosen subtree of parent 2 with the subtree of parent 1. When the leaf of a tree is replaced by a node that was the root of another tree, offspring can be bigger and deeper than their parents. This phenomenon can cause a progressive growth of individual sizes in the population and is called bloating. In order to counter it a size limit may be set for offspring. If a child exceeds this limit it is either discarded and the crossover procedure started again or it is assigned a very low fitness value. Different variants of crossover exist. Common ones include the assignment of selection probabilities to tree nodes based on depth or the selection of crossover nodes at the same position in both parents.

4.4 Evolutionary algorithms in architecture

One of the first who experimented with evolutionary algorithms for architectural applications was Frazer [21] [22]. Since then EAs were used to search for solutions for different architectural problems.

When no better fitness measure can be found than the one in the human mind, the user is often directly involved in the evaluation process during runtime. Interactive genetic algorithms (IGA) integrate human judgement into the fitness values of individuals and let the user influence the algorithmic exploration process. In architecture often subjective aesthetic criteria or functional criteria which are hard or impossible to describe formally are important. Therefore there exist many applications of IGA for architecture.

In order to create biologically inspired architectural forms Hemberg et al used Lindenmayer systems, which model the growth of plants, in combination with an EA [23]. This specific EA variant is called „Grammatical Evolution“ and contains elements from GP as well as GA. The developed system, named Genr8, creates three dimensional digital forms or surfaces by growing and evolving them. The user can control the process via the fitness function. In order to evaluate an individual and assign a fitness value, the individual is grown by applying its production rules. The EA uses a variable-length binary string which determines which grammatical production rules in Backus-Naur form are used to grow the individual [24]. In Genr8 the growth process is influenced by an environment, which has the effect that changing the environment can also change the fitness value of individuals [23].

Quiroz et al used an IGA to explore floor plan designs. They employed a special GA variant called NSGA-II: non-dominated sorted multi-objective genetic algorithm. The multi-objective optimisation combines the objective and subjective criteria for the fitness function. The algorithm creates fronts in which no individual is worse than any other. Two individuals are chosen for tournament selection. In case both individuals are part of the same front the one which is situated in the less crowded region is chosen, which is the more diverse one. The algorithm should be used in a collaborative work by several designers. Each designer can see a subset of each other's solutions. The best individuals per user are saved over generations and are part of the solution subset view. The rest of the displayed individuals are chosen randomly from the collaborative population. Figure 4.4 shows a screenshot of the collaborative design tool. The user can decide when and which individuals should be injected into his own population. The worst ten percent of his population are then replaced by copies of the individuals that should be injected. Via fitness biasing it is ensured that those individuals survive long enough to influence the search process [25].

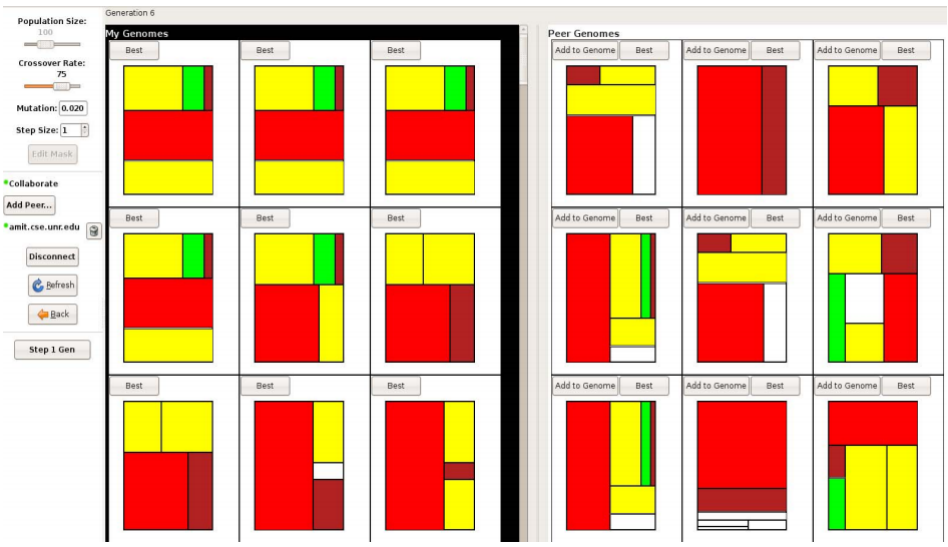


Figure 4.4: Screenshot of the collaborative design tool which enables the designer to inject solutions from peers into his population (from [25]).

These examples demonstrate that EAs are highly adaptable and extendible. Different variants were developed and mechanisms exchanged and combined so that the categorisation

of an evolutionary algorithm in a specific variant group is not clear in every case.

Coates and Hazarika used GP to generate spatial compositions with solid forms or surfaces. They used basic forms as terminal symbols and operations on forms as function symbols. Although they used artificial selection by people they also experimented with natural selection with a fitness function that takes into account the number of elements the form consists of and its surface area [26]. Although the use of IGA can be found in many EAs for architectural application, our EA will not depend on human judgement for fitness assignments. The EA developed for this thesis will use the compactness value of the isovists in the developed placing configurations as basis for their fitness values. The algorithm will work with 2D environments. But like in Coates and Hazarikas fitness function, the surface area, in our case the visible surface perimeter and the visible unblocked floor area, will also be integrated in our fitness values since they are used to calculate the compactness.

Schneider and König used several different generative approaches which are based on ES to generate room layouts or floor plans. One approach combined an ES that optimises room sizes in layouts with a GA that optimises their neighbourhood relationships [2]. Some of their efforts and results are explained in more detail in Chapter 3. One of their approach is similar to ours since they also use isovist properties in the fitness function in order to place objects in an environment [7]. They placed less but bigger objects than we intend to, but they proved that it is possible to reproduce generated patterns based on specific visuospatial properties. An individual in their EA consists of buildings, represented as rectangles, which are placed on the plot of land, represented as a rectangle as well. The algorithm could position and scale the buildings. The buildings were not allowed to overlap or to stretch out of the given boundary. These constraints were checked after the placing of a building and if one was violated special movements were applied in order to adjust the coordinates. The objects we place cannot be scaled, only placed. But we will also check the no-overlap and the boundary criteria for the objects and adjust their positions if necessary. They used a 2D, discrete environment, which we will employ as well.

Elezkurtaj used evolutionary and genetic algorithms to create architectural layouts and enable the user to interact at any time in order to direct the process. He divided the problem into a form finding and a placing problem which operate in a coevolution process. The rooms should be formed and placed in a 2D coordinate system. Since the rooms edges should all be orthogonal to their neighbouring edges, they can only be positioned horizontal or vertical. The orientation of the edges of the polygons in such an environment are alternating. Therefore when running through such a polygon, if one knows the orientation of the first edge, the orientation of the following edges can be derived. Elezkurtaj used this fact to code a room with the sequence of the alternating horizontal and vertical lengths of its edges [27]. We will use this compact encoding method for the representation of the form of the objects that should be placed.

CHAPTER 5

Evolutionary Approach: Details

5.1 Overview

Our evolutionary algorithm will take a description of an environment and of objects as input. For simplicity it will work on a discrete 2D grid. The objects will be described in shape and amount and the algorithm shall place all of them at empty places in the environment. If an overlap option is set then the objects may overlap with each other but not with the environmental blocking spaces. In the beginning as many random individuals are generated as are needed for the configured population size. Afterwards the population is sorted by fitness values. Since the fitness function is dependent on the compactness values of the individual, this step includes the calculation of isovists for each vision point. Different fitness functions can be chosen. We want to be able to minimise or maximise each of the following:

- the compactness of the point with the maximum compactness value in the environment
- the compactness of the point with the minimum compactness value in the environment
- the average compactness of all vision points in the environment
- the deviation of compactness values of all vision points in the environment

The calculation of the compactness for a vision point includes the calculation of the isovist at that point. For this task a version of the shadow casting algorithm is used. Afterwards the isovist's area and perimeter can be calculated and used for the compactness computation.

The selection mechanism is used to select individuals as parents for the children generation from the sorted population. One random value determines if crossover will be applied to the parents. If the random value is above a threshold the two children are created as copies of their parents. Afterwards it is decided individually for each child if its genes will experience a random mutation. If the adults shall die then the set of children is sorted. If $(\mu + \lambda)$ selection is used then the adults and children are all sorted by their fitness. The worst individuals are simply discarded until the population size is reached again and the resulting set will become the population of the next generation.

The creation of random individuals as well as the crossover and mutation processes require that object placings are added to an individual. An objects is always connected with its shape description. This description together with an origin coordinate can describe which points in the environment will be occupied by this object. Whenever an object needs to be placed it is checked if the object and point form a legal placing. Several criteria need to be satisfied. At first it is necessary to calculate which points the object with that origin point would occupy. Then for each of those points it needs to be checked if it is within the environment boundaries

and if it is free of environment obstacles. If overlaps are not allowed it also has to be free of other obstacles. If the checking routine found a rule violation the algorithm tries to move the obstacle a bit. If a valid point was found then the placing is added to the individual. The crossover adds obstacles of both parents to one child. If overlaps are not allowed this can lead to illegal placings. Any placing that is not possible and could not be moved, is discarded and a random placing added instead.

The algorithm stops after a prespecified number of generations and returns the best individual of the last generation.

5.2 Evolutionary algorithm for object placement

The algorithm builds on the usual structure of an evolutionary algorithm as outlined in Algorithm 4.1. Let the population at generation $t \geq 0$ be denoted as $P^{(t)}$. At first the initial population $P^{(0)}$is generated. In this step μ individuals are created by placing the required b number of objects randomly for every individual. The population is then sorted from best to worst fitness by a quicksort algorithm. This step involves the calculation of a field of vision for every non-occupied point in every individual and its compactness value in order to evaluate the fitness relevant properties.

The generation of children is one of the key components of any evolutionary algorithm. In each generation λ children are created. If elitism is activated the first step is to copy the configured number ϵ of best individuals from $P^{(t)}$ as the first children into the generations children set $C^{(t)}$. Afterwards as long as $|C^{(t)}| + 2 \leq \lambda$ two different parents are chosen from $P^{(t)}$ by the selection mechanism. A random variable $rand1$ is drawn from $[0, 1)$ and compared to the crossover rate r_{xover}. If $rand1 < r_{xover}$ crossover is applied to the parents in order to generate two children, otherwise the parents are simply copied. An additional random variable per child likewise determines whether the child's genes will experience mutation by comparing $rand2$ and $rand3$ to the mutation rate r_{mut}. Depending on the value of λ and ϵ, $|C^{(t)}| + 1 = \lambda$ might hold true at one point. This last missing child is created as a copy of a random individual from $P^{(t)}$.

The selection protocol is applied to $P^{(t)} \cup C^{(t)}$ to pick those groups which are valid candidates for the next generation. In this step the individuals from $P^{(t)}$ are discarded if the configuration determines that they shall die at the end of their generation. The remaining candidates are then sorted and if the number of candidates is larger than the population size, the worst candidates are simply discarded. The surviving μ individuals form $P^{(t+1)}$. The algorithm continues with the creation of children for generation $t+1$ until the desired number of generations, denoted by η, is reached. The pseudocode is listed in Algorithm 5.1.

Representation

The environment in which the obstacles shall be placed is given in a two dimensional matrix E which represents the w_e columns and h_e rows of the environment. Each cell is marked as either "empty" (value 0) or "blocked" (value 1). Blocked cells do not allow obstacle placement, they are occupied by environment obstacles that block the view. Empty cells do not block the view. But it might be forbidden to place obstacles in specific empty cells, these cells are called "irreclaimable". Let $E_{x,y} \in [0, 1]$ refer to the environment value of the cell in column x and row y, $x = 1, ..., w_e$ and $y = 1, ..., h_e$. In the following the term point (x, y) will also be used to refer to the cell in column x and row y. Set I contains the points that were marked irreclaimable.

The objects that have to be placed might have different shapes which need to be specified. As pointed out by Elezkurtaj [27], in an orthogonal grid movements in only two directions are possible: the vertical direction and the horizontal direction. Following the border of a polygon the directions are alternating, which makes it possible to describe the shape by

Algorithm 5.1: Used evolutionary algorithm

```
input  : η, μ, λ, r_xover, r_mut
output: best found individual
t = 0;
generate P^(0);
sort P^(0) by fitness;
while t < η do
    copy ϵ best individuals to C^(t);
    while |C^(t)| + 2 ≤ λ do
        choose two different parents from P^(t);
        if rand1 < r_xover then
            crossover;
        if rand2 < r_mut then
            mutateChild1;
        if rand3 < r_mut then
            mutateChild2;
    if |C^(t)| < λ then
        add copy of random individual from P^(t) to C^(t);
    select candidates for P^(t+1) from P^(t) ∪ C^(t) using selection protocol;
    sort P^(t+1);
    if |P^(t+1)| > μ then
        reduce to μ by cutting off the worst individuals;
return best individual from P^(t)
```

an array of integer numbers which represent alternating horizontal and vertical movements. Elezkurtaj's movement array describes the perimeter of the polygon that represent a room. Since we do not use rooms but mostly smaller and slimmer object shapes, our movement array will describe the shape directly. If the starting direction is fixed, the following directions can be derived.

Our descriptions will always start with a vertical movement. Positive y values move down, positive x values move to the right. For example the shape description $\{-1, 3\}$ means that the object is constructed by starting at origin point (x, y), then moving one cell up to $(x, y-1)$ and then three cells to the right, occupying $(x+1, y-1)$, $(x+2, y-1)$ and $(x+3, y-1)$. resulting in the object shape displayed in Figure 5.1.

Figure 5.1: Example object shape for the movement sequence $\{-1, 3\}$.

For one placement problem objects with different shapes may be defined. Each of those k object categories cat_i, $i = 1, ..., k$, is given by a movement array as shape description and the amount o_i how many objects of that category shall be placed. A placing problem is defined by an environment E, irreclaimable points set I and the object definitions cat_i, $i = 1, ..., k$ with their shape descriptions and o_i. An individual *ind* within the EA represents a solution to such a placing problem. It contains a 2-dimensional array p_{ind} which holds its obstacle placements. It contains an array $p_{ind}[i]$ of o_i points for each category cat_i which define the points at which the objects of category i are placed. p_{ind} can be processed to calculate which cells of the environment will be occupied by the placed obstacles. Afterwards the isovists of the cells that were left empty and their compactness values can be calculated in order to

assess the fitness of *ind*.

Placing an obstacle at a random position

Obstacles need to be placed at random positions during different stages of the algorithm. In the beginning the initial population is created by placing all obstacles randomly. The placement of obstacles needs to follow some restrictions which the procedure in Algorithm 5.2 ensures. At first a random point within the environment is chosen. Afterwards in the *isValidPlacing* method it is calculated which points will be occupied by an obstacle of the desired category if it is placed at that point. For each of those occupied points (x, y) it is checked if they are:

- an irreclaimable or blocked environment cell, that is if $(x, y) \in I$ or $E_{x,y} = 1$
- located outside of the environment, if $x < 1$ or $x > w_e$ or $y < 1$ or $x > h_e$
- overlapping with another object's cell if the no-overlap option is set

If any of the occupied points fulfils any of these properties, then the placing is considered to be invalid and *solveOverlap* is called. This repair mechanism shown in Algorithm 5.3 will try to move the object slightly in order to make it valid. It will try to move the object within θ steps to each side. If the root point would leave the environment border the mover continues on the opposite border, thus treating the environment like a closed circle. The starting direction is chosen at random and each direction is evaluated completely until the next one is tried. If no valid point was found within θ steps into each direction, *null* is returned.

If the overlap could not be solved by *solveOverlap*, another random point is chosen and the evaluation is started again. If within z random points no valid point was found by *addRnd* the algorithm aborts with an error. Otherwise the obstacle is placed at the first valid point that was found.

If it was configured that no overlap of obstacles is allowed then a set is maintained for each individual which contains the points that are occupied by the current object placings. If a new point shall be added every point of the obstacle to be placed can be checked against the occupied points set. If an object is deleted, the points that were occupied by this object are removed from the set.

Algorithm 5.2: addRndObj

```
input  : individual ind, category cat, number of objects to add a, step size θ, max.
         tries z
output: updated individual
count ← 0;
for i ← 0 to a do
    if count > z then
        error: no allowed point found
    p ← random cell in environment;
    if not isValidPlacing(ind, cat, p) then
        p ← solveOverlap(ind, cat, p, θ);
    if p not null then
        addPlacing(ind, cat, p);
        count ← 0;
    else
        count + +;
        i − −;
```

Algorithm 5.3: solveOverlap

input : individual *ind*, category *cat*, point to add p, steps θ
output: valid placement point, *null* if none was found
mover ← new PlacingMover(p, θ);
while *not all directions visited* **do**
  $p \leftarrow$ *mover.getNextPoint*();
  if *isValidPlacing*(*ind*, *cat*, p) **then**
    return p;
return *null*;

In the worst case the *solveOverlap* routine would evaluate θ points in each of the four directions. Its time complexity is therefore $O(\theta)$. If all of the a objects that should be added with *addRndObj* cause an overlap, *solveOverlap* is called at maximum z times for each object. *addRndObj* therefore has a time complexity of $O(a \cdot z \cdot \theta)$.

Mutation

When mutation is applied to an individual, a random object from a random category is chosen and moved to a new random location. For very large object amounts the mutation of just one object's position has little chance of influencing the compactness of a minimum or maximum point and also hardly changes the average or deviation of the whole population. Therefore the mutation steps to be executed are calculated as the rth fraction of the overall amount of cells that need to be placed. The overall amount of cells is calculated as the sum of the number of cells each obstacle category occupies multiplied by how often it shall be placed. At least one mutation step is executed. More formally, $\tau = \lfloor sum_{cell} \cdot r \rfloor$ obstacles are moved if $\lfloor sum_{cell} \cdot r \rfloor > 0$, where sum_{cell} is:

$$sum_{cell} = \sum_{i=1}^{k} o_i \cdot cell\_count(cat_i) \tag{5.1}$$

and $cell\_count(cat_i)$ symbolises the amount of cells the shape for category cat_i occupies. Otherwise, if $\lfloor sum_{cell} \cdot r \rfloor = 0$ then $\tau = 1$ obstacle is moved.

This does not influence the mutation probability. But in case the mutation is executed, more than one object's position is changed.

For every mutation movement a random obstacle of a random object category is chosen and deleted from the individuals placement list. Afterwards a new obstacle of the same category is inserted at a random position. The mutation is demonstrated in Algorithm 5.4. For each mutation step *addRndObj* is called for adding one object. The time complexity of mutation is therefore $O(\tau \cdot z \cdot \theta)$.

Algorithm 5.4: Mutation on one individual

input : individual *ind*
output: mutated *ind*
for $m \leftarrow 0$ **to** τ **do**
  cat ← random category;
  obj ← random obstacle in p_{ind} [*cat*];
  removeObject(*ind*, *obj*);
  addRndObj(*ind*, *cat*, 1);

Crossover

Crossover is applied to two individuals, called $parent_1$ and $parent_2$ in the following, by using n_{xover} crossover points. For every crossover point a random object category and subsequently a random object in that category is chosen. The algorithm sequentially processes object for object in the parents placement lists and creates two children, $child_1$ and $child_2$. Every processed object of $parent_1$ is copied to $child_1$ and every processed object of $parent_2$ is copied to $child_2$.

Algorithm 5.5: Crossover of two individuals

```
input  : Individuals parent_1 and parent_2, n_xover, category amount k
output: Individuals child_1, child_2
Create new individuals: child_1, child_2;
beginObjectCat ← 1;
for h ← 1 to n_xover do
    endObjectCat ← random between beginObjectCat and k;
    endObject ← random between 1 and o_endObjectCat;
    for curCat ← beginObjectCat to endObjectCat do
        fill ← addGetFill(child_1, curCat, p_parent_1[curCat]);
        fillChild_1[curCat] = fill;
        fill ← addGetFill(child_2, curCat, p_parent_2[curCat]);
        fillChild_2[curCat] = fill;
    fill_1 ← 0;
    fill_2 ← 0;
    for i ← 1 to o_endObjectCat do
        curP1 ← p_parent_1[endObjectCat][i];
        curP2 ← p_parent_2[endObjectCat][i];
        fill_1 += addGetFill(child_1, endObjectCat, curP1);
        fill_2 += addGetFill(child_2, endObjectCat, curP2);
        if i = endObject then
            swap contents of child_1 and child_2;
            swap contents of fill_1 and fill_2;
            swap contents of fillChild_1 and fillChild_2;
    fillChild_1[endObjectCat] = fill_1;
    fillChild_2[endObjectCat] = fill_2;
    beginObjectCat = cat_endObjectCat+1;
for curCat ← beginObjectCat to k do
    fill ← addGetFill(child_1, curCat, p_parent_1[curCat]);
    fillChild_1[curCat] = fill;
    fill ← addGetFill(child_2, curCat, p_parent_2[curCat]);
    fillChild_2[curCat] = fill;
for cat ← 1 to k do
    addRndObj(child_1, cat, fillChild_1[cat]);
    addRndObj(child_2, cat, fillChild_2[cat]);
return child_1, child_2;
```

When the chosen object for the current crossover point is reached, $child_1$ and $child_2$ exchange places, so that the next objects will be copied from $parent_2$ to $child_1$ and from $parent_1$ to $child_2$. Algorithm 5.5 demonstrates the process. After the chosen object for the last crossover point was processed the children are exchanged once more and the remaining objects copied. When obstacles are added to an individual they might cause an overlap with

other obstacles. If overlapping of objects is not allowed (if the no-overlap option is set), these cases need to be repaired. The *addGetFill* method shown in Algorithm 5.6 adds the obstacles of the given category to the given individual if they cause no overlap or the overlap could be solved by the *solveOverlap* method. It counts how many objects could not be added and returns that number. In the *crossover* algorithm those numbers are saved per category for each child in $fillChild_1$ and $fillChild_2$. After all obstacles that could be added were added to the children, for each $child_j, j \in \{1, 2\}$ and for each category *cat* $fillChild_j[cat]$ random objects of category *cat* are added to $child_j$. Adding the random objects at the end of the process rather than when an overlap could not be resolved, prevents causing more overlaps with randomly placed objects during the copy process.

Both parents as well as both children have the same amount of placed objects per object category. The number of crossover points is naturally limited by the number of overall objects that shall be placed.

Algorithm 5.6: addGetFill

```
input : Individual ind, object category cat, list list of coordinates to add, step size θ
output: number of how many objects could not be added
count ← 0;
for point p in list do
    if isValidPlacing(ind, cat, p) then
        addPlacing(ind, cat, p);
    else
        p ← solveOverlap(ind, cat, p, θ);
        if p not null then
            addPlacing(ind, cat, p);
        else
            count++;
return count;
```

In the worst case the *addGetFill* algorithm calls *solveOverlap* for every point in the given *list*. Therefore its time complexity is $O(|list| \cdot \theta)$. In the crossover procedure two parents are used to create two children. If the no-overlap option is set then the objects of each parent are known not to be overlapping. At most half of the placings per child can cause an overlap. Therefore for both children together *solveOverlap* could be called at maximum $\sum_{i=1}^{k} o_i$ times. Additionally at the end at most $\sum_{i=1}^{k} o_i$ random objects might be added by *addRndObj*. Therefore the time complexity of crossover is $O(\sum_{i=1}^{k} o_i \cdot \theta)$.

Calculation of perimeter, area and compactness

The Shadow Casting algorithm (which is described in general in the „Background“ chapter and in detail in the next section) calculates a lit map L for a point p in an environment F. L contains the information which environment cells are visible from p. If cell (x, y) is visible from p then $L_{x,y} = 1$, otherwise $L_{x,y} = 0$. All cells x, y such that $L_{x,y} = 1$ form the isovist of p. L is passed to an algorithm that calculates the perimeter and area for the isovist. It iterates through every column of every row, evaluating each cell (x, y):

- $F_{x,y} = 0$ and $L_{x,y} = 1$, the cell is empty and visible: add one to area
- $x > 0$ and $F_{x-1,y} = 0$ and $L_{x-1,y} = 1$ and ($F_{x,y} = 1$ or $L_{x,y} = 0$): add one to perimeter
- $y > 0$ and $F_{x,y-1} = 0$ and $L_{x,y-1} = 1$ and ($F_{x,y} = 1$ or $L_{x,y} = 0$): add one to perimeter
- $x > 0$ and ($F_{x-1,y} = 1$ or $L_{x-1,y} = 0$) and $F_{x,y} = 0$ and $L_{x,y} = 1$: add one to perimeter

- $y > 0$ and ($F_{x,y-1} = 1$ or $L_{x,y-1} = 0$) and $F_{x,y} = 0$ and $L_{x,y} = 1$: add one to perimeter

With the perimeter and area values of the isovist at p the compactness value at p can be calculated using Benedikt's formula (see Equation 2.3). In the EA each individual *ind* represents a solution environment F^{ind} such that $F^{ind}_{x,y} = 1$ if either $E_{x,y} = 1$ or if (x, y) is occupied by an object placement from *ind*. To evaluate an individual *ind* the Shadow Casting algorithm is called for each point (x, y) in environment F^{ind} for which $F^{ind}_{x,y} = 0$ holds.

Fitness - target functions

In order to evaluate the solutions and calculate a fitness value for every individual, it is necessary to define a target function. Which characteristic of the solution shall be maximised or minimised? The evolutionary algorithm searches for the individual with the highest fitness value. Therefore if a value shall be minimised, its reciprocal value is used as fitness value. In the following the used measures are listed and shortly described.

Average compactness of the compactness values of all empty fields.

Deviation is the standard deviation of the compactness values of all empty fields.

Maximum compactness is the highest compactness value of all empty fields.

Minimum compactness is the lowest compactness value of all empty fields.

Every measure can be minimised or maximised, resulting in a total of eight different possible target functions:

max avg, min avg, max dev, min dev, max max, min max, max min, min min

5.3 Shadow Casting algorithm

This section contains details on the implementation of the Shadow Casting algorithm. For a general explanation on how the Shadow Casting algorithm works see Chapter 2.

The Shadow Casting algorithm (based on [28]) processes the environment of width w_e and height h_e by octants. The *castLight* algorithm is started twice for every of the four diagonals that originate from the vantage point and processes the octants that join this diagonal. Each octant is processed from the diagonal to the axis. This requires one of the two casts for each diagonal to be processed row by row and the other one to be processed column by column. These are the outer directions. Each row would be processed column by column and respectively each column would be processed row by row. These are the inner directions. The *castLight* method of the Shadow Casting algorithm is recursively started if a blocking cell was found. It takes the following arguments:

- The current **distance** of the outer row or column to the vantage point.
- The **start slope** is one of the boundary slopes for the evaluation of that call. The starting slopes for the eight octant calls are the diagonals which evaluate to a slope of 1.
- The **end slope** is the other boundary slope. The end slopes for the eight octant calls are the axes which evaluate to a slope of 0.
- The **direction** is the current quadrant which is processed. Each direction contains an x and an y delta value which determines into which direction the evaluation progresses.

- The flag **xInner** signifies which half of the quadrant is evaluated. If it is *true* then the part which is processed by using the columns as outer direction. If it is *false* then row by row is the outer approach.

The starting calls for the octants are listed in Algorithm 5.7. Algorithm 5.8 then lists the *castLight* algorithm.

Algorithm 5.7: shadowCastingFoV

input : vision point startx, starty
output: lightMap
lightMap[*starty*] [*startx*] ← true
Directions.add(DOWN_RIGHT($x : -1, y : -1$));
Directions.add(DOWN_LEFT($x : 1, y : -1$));
Directions.add(UP_RIGHT($x : -1, y : 1$));
Directions.add(UP_LEFT($x : 1, y : 1$));
for *d in Directions* **do**
    castLight$(1, 1, 0, d)$;
    castLight$(1, 1, 0, d)$;
return *lightMap*;

At first the *castLight* method checks if the start slope is smaller than the end slope. If this is the case then there is no valid area in between and the method returns immediately. Starting with the distance that was passed as parameter it evaluates every outer entity by iterating though every of its inner elements. The starting inner element is the one with the same distance like the current outer distance (the element on the diagonal at this distance) and the inner distance is reduced at each iteration. After the end slope was reached the outer distance is increased by one and the next elements are evaluated. The current cells coordinates are calculated by using the vision point coordinates and the inner and outer distance. In case the outer coordinate is situated outside of the environment border all viable cells in this octant were evaluated and *castLight* returns. If the inner coordinate lies outside of the environment it means that the distance to the border is smaller in the inner direction than in the outer direction. The inner distance is then adjusted to the first element that lies within the environment for the current outer distance. This procedure is represented in Algorithm 5.9.

Once the current cell coordinates are calculated the slopes for the leftmost and rightmost visible corners of that cell are calculated. The slope closer to the diagonal (and therefore the start slope) is called right slope, the other one is the left slope. If the right slope is at least as steep as the starting slope, the cell is considered outside of the visible area and the evaluation continues with the next visible cell. This requires adjusting the inner distance accordingly in order to jump over all out of slope cells in between. If the left slope is smaller or equal to the end slope, then all visible cells in this outer distance unit were completed and the algorithm continues with one distance further. But if the cell is within the slope boundaries it is recognised as visible and its content and context is evaluated according to the state of this cell and the cell that was evaluated right before this cell.

- This cell **free**, last cell **free**:
  If it is a free cell and last evaluated cell was free as well then the evaluation can continue with the next cell.

- This cell **occupied**, last cell **free**:
  If the last cell was free but this one contains an obstacle then *castLight* is called recursively for the cells between the start slope and the left slope of the current cell for

all further distances. The right slope is saved as the current known end of obstacles in case no other obstacle will follow.

- This cell **occupied**, last cell **occupied**:
  The right slope is saved as the new current known end of obstacles.

- This cell **free**, last cell **occupied**:
  If the last evaluated cell contained an obstacle and this one does not then the end of the obstacles was found and the new starting slope for the following evaluation is set to the last saved right slope. This skips the iteration through cells behind the obstacle series which are not visible.

Algorithm 5.8: Shadow Casting FOV castLight

```
input  : distance, start slope, end slope, direction d, xInner
output: light map
newStart ← 0;
if startSlope < endSlope then
    return;
for distance to max(w_e, h_e) do
    if last cell was blocking then
        return;
    outer ← −distance;
    for inner ← −distance to 0 while end < (deltaInner − 0.5)/(deltaOuter + 0.5)
    do
        currentX ← startx + d.deltaX · (xInner?deltaInner : deltaOuter);
        currentY ← starty + d.deltaY · (xInner?deltaOuter : deltaInner);
        newInner ← checkBoundaries(curY, curX, inner, d, xInner);
        if newInner = 1 then                    // reached end in outer direction
            return;
        if newInner ≠ inner then
            inner ← newInner;
            continue;
        leftSlope ← (inner − 0.5)/(outer + 0.5);
        rightSlope ← (inner + 0.5)/(outer − 0.5);
        if startSlope <= rightSlope then
            if startSlope ≠ rightSlope then
                inner ← (startSlope · (outer − 0.5) − 0.5) − 1;
            continue;
        lightMap[curY] [curX] ← true;
        if last cell was blocking then
            if this cell is blocking then
                newStart ← rightSlope;
            else
                startSlope ← newStart;
                if startSlope < endSlope then
                    return;
        else
            if this cell is blocking then
                castLight(distance + 1, startSlope, leftSlope, d, xInner);
                newStart ← rightSlope;
```

Algorithm 5.9: checkBoundaries

input : currentY, currentX, deltaInner, Direction d, xInner, map height h_e, map width w_e
output: new deltaInner or 1 if out of slope

```
if currentY < 0 then
    if xInner then
        return 1;
    return ((0 - vision point y / d.deltaY) -1;
if currentY > h_e -1 then
    if not xInner then
        return ((h_e -1 - vision point y) / d.deltaY) -1;
    return 1;
if currentX < 0 then
    if not xInner then
        return 1;
    return ((0 - vision point x) / d.deltaX) -1;
if currentX > w_e -1 then
    if xInner then
        return ((w_e - 1 - vision point x) / d.deltaX) -1;
    return 1;
```

The *checkBoundaries* method has a constant time complexity. The worst case for the *castLight* algorithm is an environment in which every cell is visible from the vision point. In this case the algorithm needs to evaluate each cell. Its time complexity therefore is $O(w_e \cdot h_e)$.

5.4 Typical solutions

In this section the effect of the different target functions shall be distinguished. A test series was executed and the results of the optimisation processes are presented in this section in order to provide a representive solution for every target function.

The layout of the environment in which the objects shall be placed and the amount and shape of the objects need to be configured as input of the algorithm. The used environments will be displayed as ascii image and can also be downloaded from `https://www.ac.tuwien.ac.at/research/problem-instances`. They contain the following symbols:

#	wall - blocking cell
.	free cell
'	irreclaimable but non-blocking cell

Each object category that was defined to be placed in the environment will be listed with a number, indicating how often that object shall be placed, followed by a colon and the objects shape.

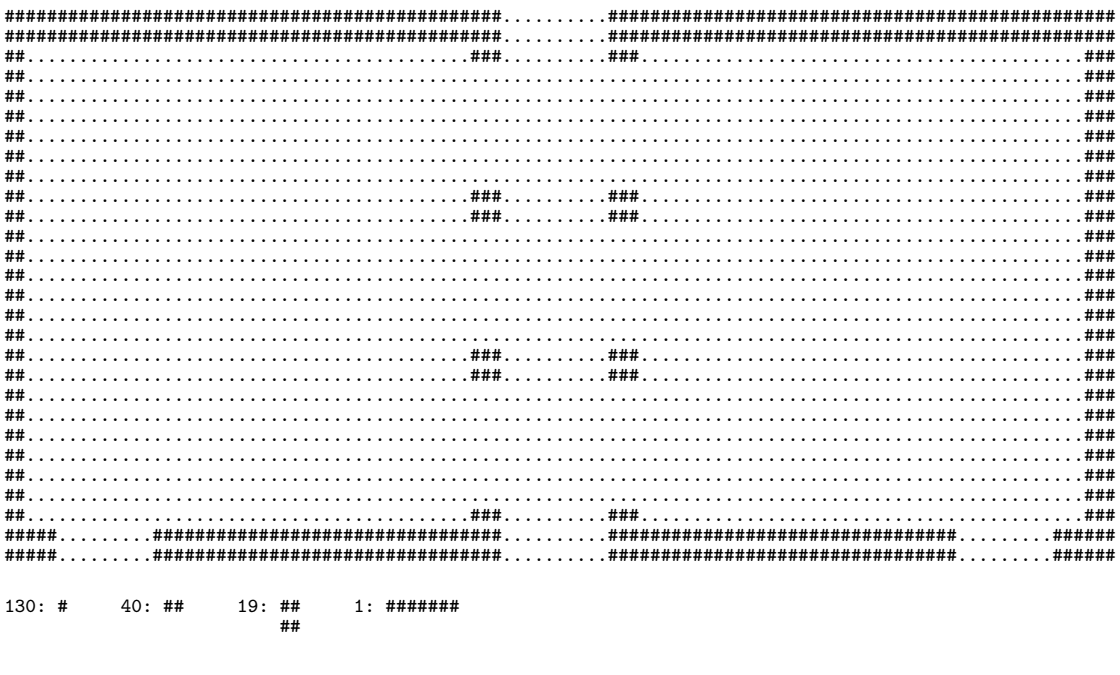
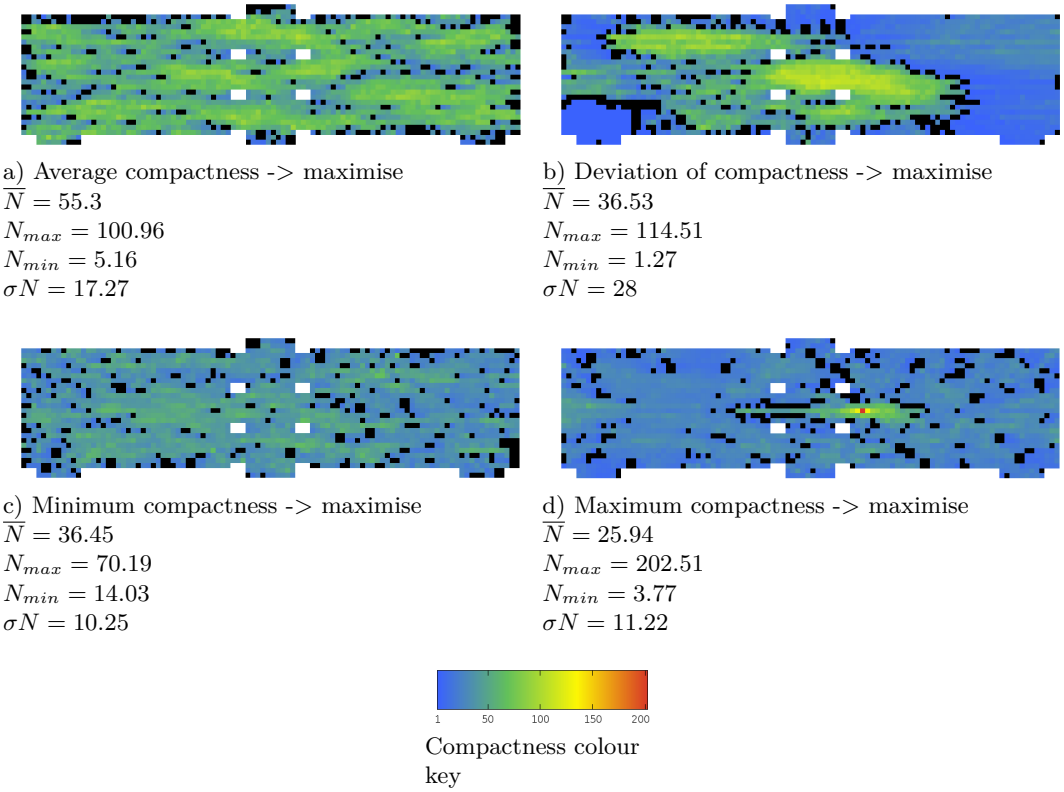


Figure 5.2: The bienale pavilion instance and the objects to be placed in it.

As environment for the runs in this section the Bienale Pavilion instance shown in Figure 5.2 was used. It resembles a room which is divided into two parts by symmetrically placed columns. It contains a path in the middle between the left and right half. Objects of different shapes shall be placed, the objects and their quantity is displayed under the instance.

Several different options can be configured for the EA, including popular parameters like: target function(see Section 5.2 for details), population size, children number, generations, selection mechanism, elitism, number of crossover points, mutation rate, crossover rate and the choices for removal of duplicates and if adults shall die at the end of their generation. Preliminary tests were executed that provided insight in parameter qualities. An overview about parameter impacts on solution qualities can be found in Chapter 6. For the target function investigation in this section a test series using a population of 460 was performed for 600 generations. For every available target function one solution was chosen to represent it in this comparison. The results show patterns for the placed obstacles and the differences in

compactness distribution. The images were divided into one group of maximisation and one group of minimisation. Since the compactness values that occur in the maximisation results are a lot higher than the highest value occurring in the minimisation results, the two groups use a different colour code scaling which is displayed below the group.



a) Average compactness -> maximise
$\overline{N} = 55.3$
$N_{max} = 100.96$
$N_{min} = 5.16$
$\sigma N = 17.27$

b) Deviation of compactness -> maximise
$\overline{N} = 36.53$
$N_{max} = 114.51$
$N_{min} = 1.27$
$\sigma N = 28$

c) Minimum compactness -> maximise
$\overline{N} = 36.45$
$N_{max} = 70.19$
$N_{min} = 14.03$
$\sigma N = 10.25$

d) Maximum compactness -> maximise
$\overline{N} = 25.94$
$N_{max} = 202.51$
$N_{min} = 3.77$
$\sigma N = 11.22$

Compactness colour key

Figure 5.3: Examples of placings reached by maximising the different compactness measures where $\overline{N}$ is the average compactness, N_{max} the maximum compactness, N_{min} the minimum compactness and σN the deviation of the compactness values for the shown example.

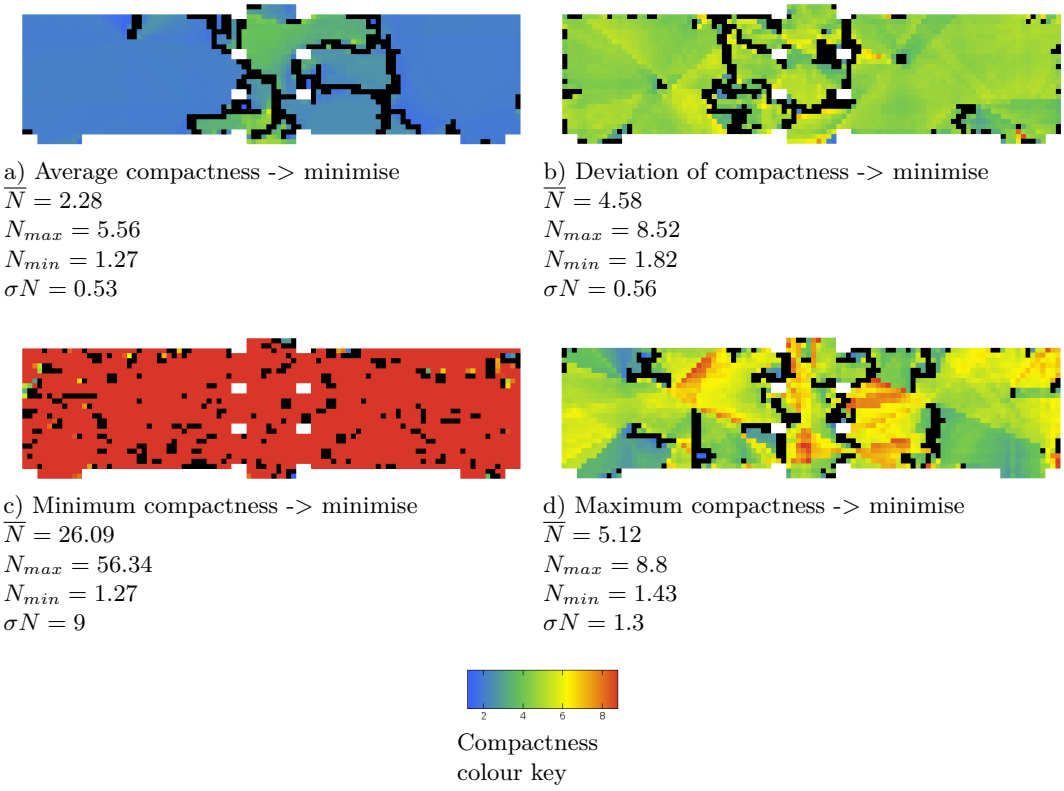
The maximisation of the average compactness yields configurations in which the objects are scattered over the whole map (see example in Figure 5.3a). This means there will be many points with a high compactness value and an isovist with fingers, which show you glimpses of different environment areas.

The maximisation of the standard deviation of the compactness values leads to configurations which contain an area similar to the maximisation of the average compactness as well as an area which contains almost no obstacles and resembles one big free space which you can perceive as a whole (example in Figure 5.3b).

The maximisation of the compactness of the point with the smallest compactness value yields placements similar to the average compactness maximisation. However, since the focus is on raising the compactness of every individual point of the configuration and not on raising the average compactness, the maximum values that occur are in general smaller (example in Figure 5.3c).

The maximisation of the compactness of the point with the highest compactness value means there will be one point in the resulting configuration whose isovist consists of as many fingers as possible. Usually this means that there is one small area which is highlighted by having clearly higher compactness values than the rest of the map (example in Figure 5.3d). As can be seen by comparing the heat maps for the different maximum target functions, the maximisation of the maximum compactness leads to maximum compactness values which

are clearly higher than the values that occur in the results of the other maximisation target functions.



a) Average compactness -> minimise
$\overline{N} = 2.28$
$N_{max} = 5.56$
$N_{min} = 1.27$
$\sigma N = 0.53$

b) Deviation of compactness -> minimise
$\overline{N} = 4.58$
$N_{max} = 8.52$
$N_{min} = 1.82$
$\sigma N = 0.56$

c) Minimum compactness -> minimise
$\overline{N} = 26.09$
$N_{max} = 56.34$
$N_{min} = 1.27$
$\sigma N = 9$

d) Maximum compactness -> minimise
$\overline{N} = 5.12$
$N_{max} = 8.8$
$N_{min} = 1.43$
$\sigma N = 1.3$

Compactness
colour key

Figure 5.4: Examples of placings reached by minimising the different compactness measures where $\overline{N}$ is the average compactness, N_{max} the maximum compactness, N_{min} the minimum compactness and σN the deviation of the compactness values for the shown example.

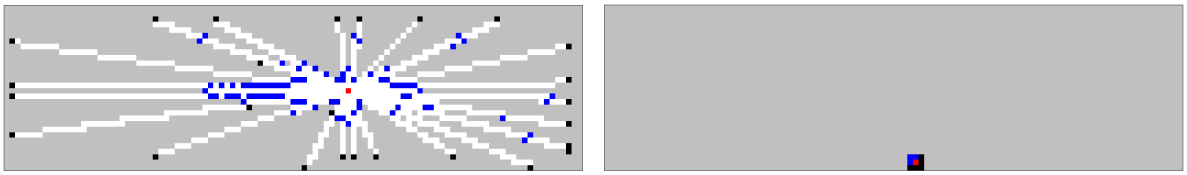
The minimisation of the average compactness results in some big empty chambers and very low compactness values. The obstacles are usually grouped around the already existing columns in the middle of the instance (example in Figure 5.4a).

The minimisation of the standard deviation of the compactness values leads to similar placing like runs which use the average compactness as the target value. However, the compactness values are less extreme (example in Figure 5.4b).

The minimisation of the compactness of the point with the highest compactness value also leads to configurations with bigger rooms. But compared to the average compactness or deviation maps the rooms are more frazzled (example in Figure 5.4d).

The minimisation of the compactness of the point with the smallest compactness value generates more or less random placements (example in Figure 5.4c). This is due to the fact that the minimum compactness value occurs in a square which means just one square isovist on the map and the best possible target value is reached. This is most often one simple point which is surrounded completely. Please note: the compactness values in this map are very high compared to all other values occurring in the minimisation group and therefore above the high end of the colour map used for the minimisation maps.

Figure 5.5a shows the isovist of the point which has the maximum compactness of 202.51 of the solution shown in Figure 5.3d. Figure 5.5b likewise shows the isovist of the minimum compactness of 1.27 of the solution shown in Figure 5.4c, which is also the shape with the smallest compactness possible in this discrete setting. In the figures the red point marks the view point, the white area represents free seen space. The black dots are parts of the environment and the blue ones represent visible placed obstacles.



a) The isovist of the point with the biggest compactness.

b) The isovist of the point with the smallest compactness.

Figure 5.5: The isovists of the biggest and smallest compactness in this test series. The blue blocks are blocking cells, the white area is unblocked space and the red dot represents the vision point.

The minimisation of the maximum compactness in general leads to a bigger maximum compactness than the minimisation of the average compactness. This can be explained by considering that when minimising the maximum compactness the comparison of two solutions only considers the highest compactness value. But for a good solution all compactness values need to be low. Exactly this is done by minimising the average compactness. Therefore it finds solutions with a small highest compactness value clearly faster than the minimisation of that value directly.

On the contrary, the maximisation of the minimum compactness does in general lead to higher minimal compactness values than the maximisation of the average compactness. It seems very easy to find a solution which contains a square isovist and therefore the lowest possible compactness value. But it seems hard to find a solution which contains only high compactness values. The maximum minimal values found are not that big in general (clearly below one fourth of the maximum compactness value found). Therefore the exact value of the minimum might not matter that much for the maximisation of the average value. However it is noteworthy that in this test series those minima were still bigger than the lowest possible compactness value.

CHAPTER 6

Evolutionary Approach: Evaluation

6.1 Test setting

For any evolutionary algorithm an important task to carry out is the calibration of the algorithm parameters. Only a parameter setting that suits the use case can ensure that the discovered solutions belong to the best solutions for this problem. Since the algorithm uses random choices in the solving process, more than one run with specific settings is necessary in order to evaluate a setting set. For the statistics presented in this chapter every setting was executed by at least 15 instances. Every instance results in one best solution found in this run. For the statistics the average value of those 15 or more best solutions was calculated. Depending on the target function and algorithm settings, the actual fitness values can differ from the compactness values. For the creation of the plots the compactness values were used, since they are more interesting and descriptive. Also, compactness values can be compared between different runs, settings or target function. The ranges of the fitness values can differ for different target functions or settings with fitness penalty and therefore they are not easily comparable to each other.

In the following test runs the maximum compactness appearing on any field of the map is minimised. Elitism for one element was set for every run. Duplicates were prohibited where not stated otherwise (see subsection Removal of duplicates) and adults died at the end of the generation so that the survival selection only involved individuals from λ. Additionally to tournament selection sigmascaling combined with proportional selection was used.

The results are presented mostly in boxplot graphics. The box is starting at the first quartile, which marks the boundary such that 1/4 of the points have a value equal or less than it and ends at the third quartile, which marks the boundary such that 3/4 of the points have a value equal or less than it. The median is marked as a horizontal line in the box, it is the point of which 1/2 of the points have a value equal or less than it. The whiskers end at the most distant point that lies within 1.5 times the interquartile range. Any points outside of those ranges are marked with a cross.

The environment that was used is the room instance in Figure 6.1. It is a simple rectangle of 46x21 which is completely empty. Only one type of objects shall be placed into the room, but their shape is a bit more complex than the shapes of the pavilion instance.

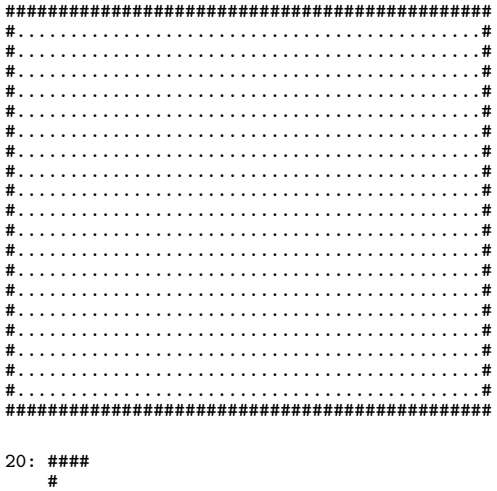


Figure 6.1: The room instance and the objects to be placed in it.

6.2 Number of generations

The number of generations which the algorithm calculates is an important parameter to set. The algorithm should have enough time to converge to a solution. Otherwise if the number of generations is too low, the results might be worse than what could have been achieved. But the number of generations should not be too high either. If the algorithm already converged and does not find any better solutions the runtime is increase without providing benefits to the solution.

Figure 6.2 shows a comparison of runs for the minimum maximum compactness with different numbers of generations. The population size and number of children was 50. The mutation rate was set to 0.6, the crossover rate to 0.8 and two crossover points were used. For the first entry with 0 generations just the initial population was generated and evaluated, no children were generated. It is not surprising that with only 20 iterations the best found placing configuration is much worse than with 50 or more iterations. Of the two tested selection mechanisms tournament selection with a group size of 5 clearly performed better than sigmascaling in this test. It seems that with this configuration after more than 300 generations not a lot of improvements are found any more with the tournament configuration. The sigmascaling runs still have room to improve, so the gap between the two mechanism closes with the increasing number of generations.

6.3 Mutation and crossover

The crossover rates determines the amount of individuals which is created through the crossover of two individuals of the previous generation. The mutation rate determines the chance that individuals experience gene mutation. The results in Figure 6.3 show that in this setting a mutation rate of at least 0.5 in combination with tournament selection seems to be advantageous. These runs used a population of 200 individuals, 200 children and ran for 200 generations. Again, two crossover points were used.

It is interesting to see that with a crossover rate of 0.8 the sigmascaling runs seem to perform worse with increasing mutation rate, while the opposite holds for the tournament selection runs. Tournament selection clearly outperforms sigmascaling with higher mutation rates. The statistics suggest that the best results are to be expected with tournament selection, a mutation rate of 0.6 and a crossover rate between 0.2 and 0.6.

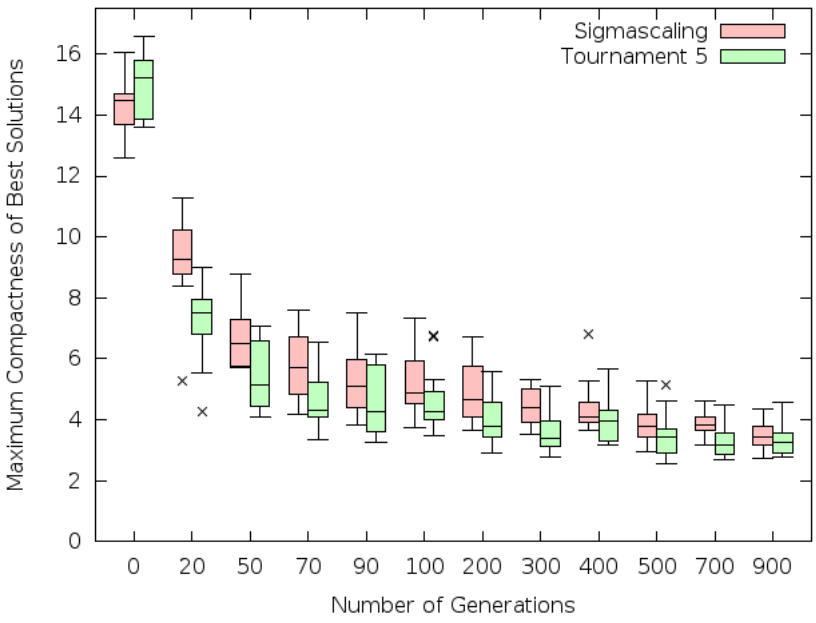


Figure 6.2: Comparison of runs for the minimum maximum compactness with different numbers of generations.

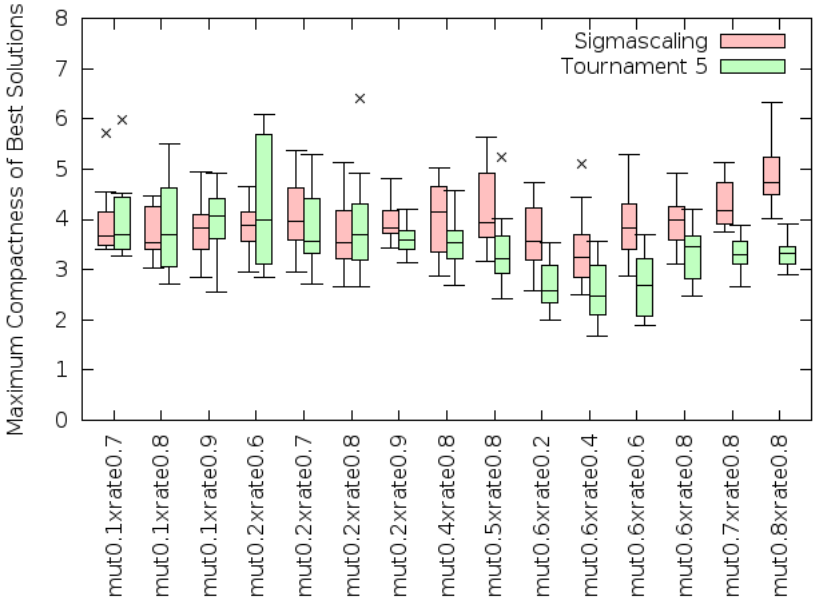


Figure 6.3: Comparison of solutions with different mutation and crossover rates from runs for the minimum maximum compactness.

6.4 Children

The term „overproduction“ refers to the configurations which define the number of children λ bigger than the population size μ. In this case in every generation the number of produced children is bigger than the population pool. Therefore the group of next generation individuals has to be reduced after creation. In this algorithm the worst individuals are simply discarded until only exactly the population size remains. This increases the selection pressure [15].

Figure 6.4 shows an overview of the evaluation of the impact of different sizes of λ. All

runs used the same population size of $\mu = 50$ and ran for 100 generations. With an increasing number of children the statistics show a trend to a decreasing maximum compactness, which means an increasing quality of solution.

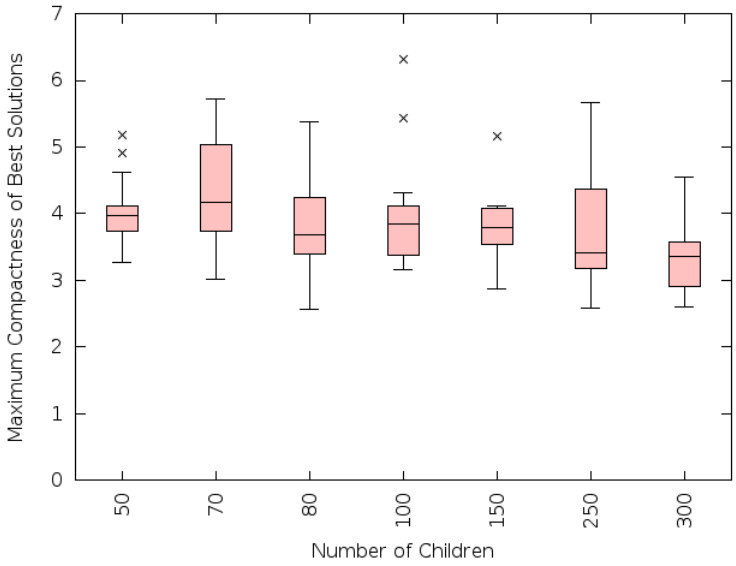


Figure 6.4: Comparison of solutions with different numbers of children from runs for the minimum maximum compactness.

6.5 Population size

Additionally to increasing the children size, the population size could be increased as well. More individuals in the population can cover a wider range of solutions. The population size is therefore determining the degree of parallel search. Complex, multi-peaked search landscapes need a higher degree of parallelism in order to explore the search space thoroughly [15].

For the population size evaluation instances with a population and children size of $\mu = \lambda = 50$ and instances with $\mu = \lambda = 100$ were used. Since we start with the configuration that generates as many children as are needed to make up a new generation, an increase of the population size requires another change if we want to keep the survival selection mechanism (parents die, select the next generation individuals only from λ). Therefore it is necessary to increase the children size as well.

The test instances again minimised the maximum compactness. As can be seen in Figure 6.5 runs with an increased population size and children number perform better. That even holds for the runs with higher generation numbers, where a simple increase of the generation number did not lead to significant improvements any more. The runs with 100 or more generations showed better performances with an increase of the population size and number of children by 50 than the further increase of the generation number by 100.

Figure 6.6 shows the development of the maximum compactness of the best solution for runs with a population size of 50, 100 and 200. The graphs of the runs with the smaller population size show big jumps and stagnation episodes, while the graphs of the runs with the bigger population size seem more continuously and reach better solutions earlier. It also shows clearly that runs with higher population size reach better final solutions while runs with smaller population size tend to stagnate earlier.

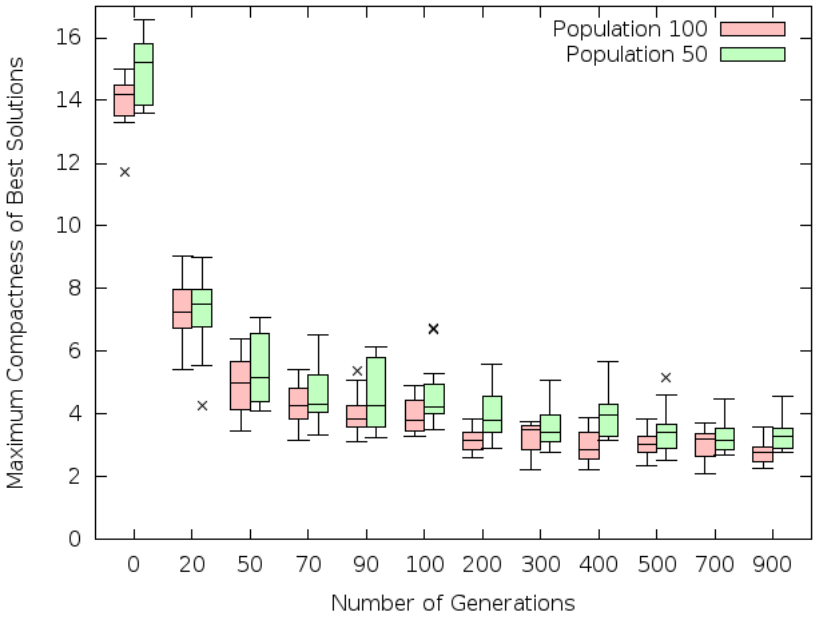


Figure 6.5: Comparison of solutions with different population sizes for different generation numbers from runs for the minimum maximum compactness.

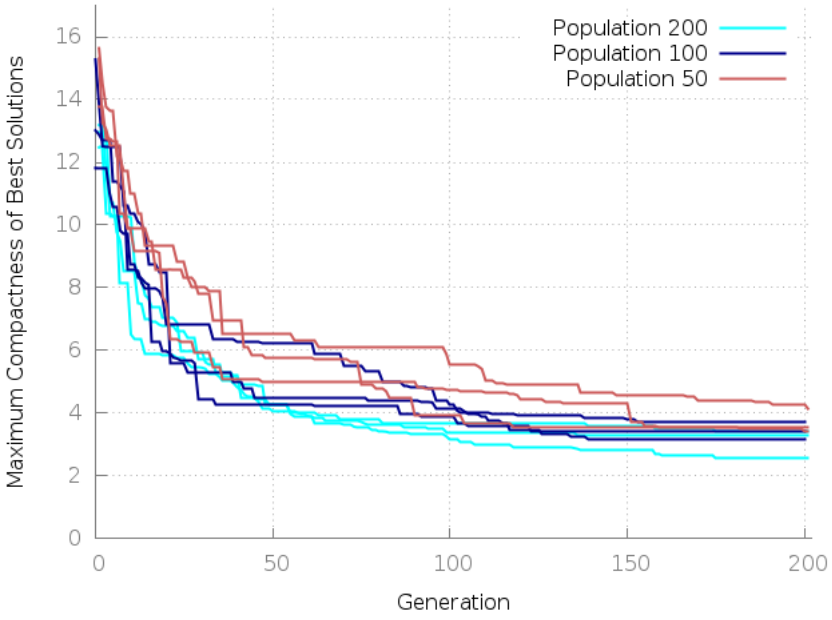


Figure 6.6: Comparison of the developments in runs for the minimum maximum compactness with a population size of 50, 100 and 200.

6.6 Crossover points

The number of crossover points defines into how many sequences the genes of the parents are divided before they are put together alternately to form the two children's genes. Different numbers of crossover points were selected for evaluation. Additionally uniform crossover was tested. When uniform crossover is used, it is decided individually for every gene of a parent to which child it shall be copied too. The second child receives the gene from the other involved parent.

The results of the tournament instances for the minimum maximum compactness in Figure 6.7 suggest that if sigmascaling is used as selection mechanism then the number of crossover points should be between 2 and 12 in order to achieve better results. There is a big difference between sigmascaling and tournament selection in the performance of runs with one crossover point or more than 15 crossover points: sigmascaling suggests they are within the worst options while this is not the case with tournament selection. The performance of sigmascaling in general seems to be stronger influenced by the number of crossover points.

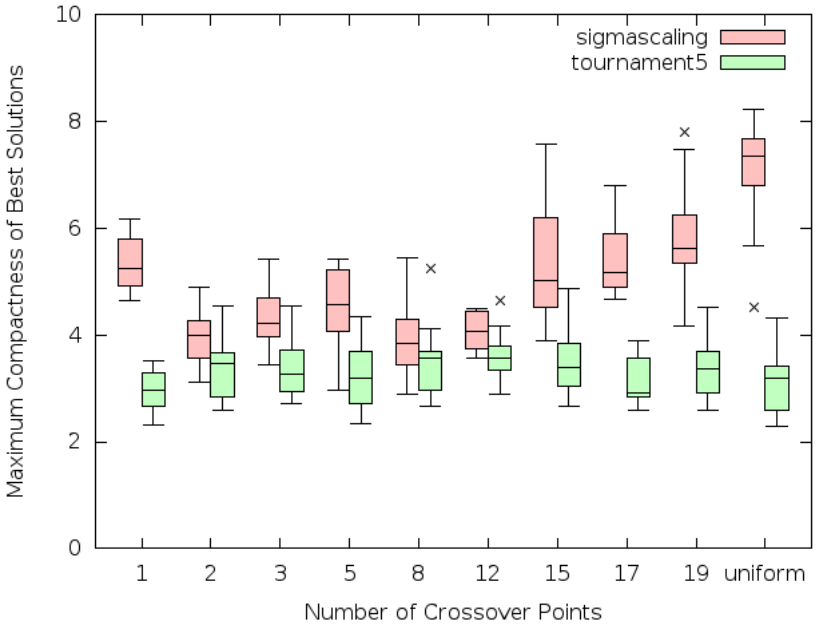


Figure 6.7: Comparison of solutions with different numbers of crossover points from runs for the minimum maximum compactness.

6.7 Tournament group size

Tournament selection is a popular selection mechanism which was used in most evaluation tests for parent selection. X random individuals are chosen and compared to each other and the best one is chosen as a parent for the next generation. This is repeated until all parents were chosen. X is called the tournament group size. If the group size is very small, selection pressure is very low and the selection mechanism is closer to random selection (it is random selection if the group size is 1). With increasing group size the selection pressure rises since a chosen individual has to compete with more other individuals over being the chosen one of the group.

Again the maximum compactness was minimised. The graphic in Figure 6.8 also puts sigmascaling in compare with different tournament configurations. It shows that with a population of 200 sigmascaling reaches a similar performance like tournament selection with

a group size of 3. But a group size between 5 and 9 performs significantly better and 5 seems to be the best option in this setting with a population size and children number of 200 and 200 generations.

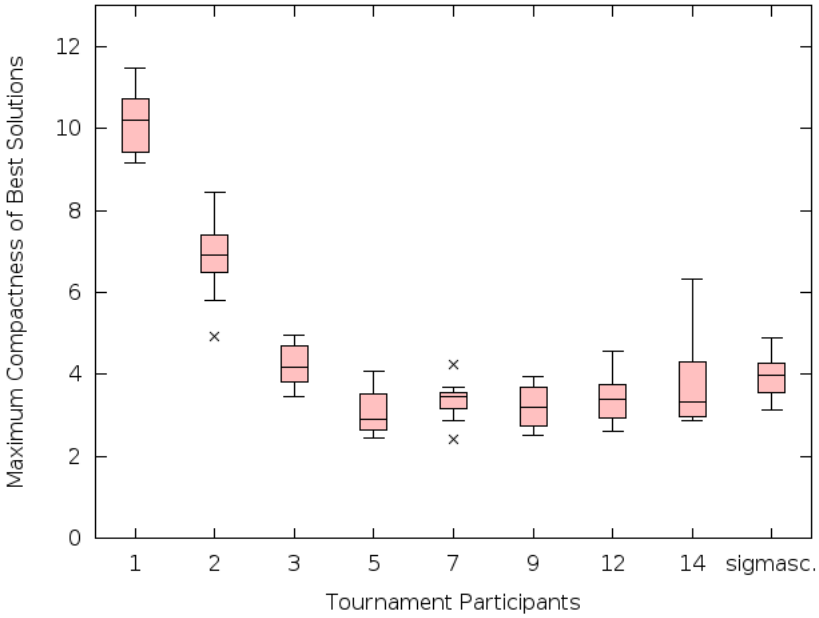


Figure 6.8: Comparison of solutions with different tournament group sizes from runs for the minimum maximum compactness.

6.8 Removal of duplicates

During a run the population might contain duplicate individuals whose genes do not differ from each other. There are several possible reasons for this circumstance. First of all individuals can be chosen as parent more than once per generation. And since crossover and mutation are not applied to every individual created, a genotype could be copied more than once into the next generation without change. Another point is, since mutation and crossover are random, two operations could lead to the same genotype. Those duplicates do not add diversity to the pool. Therefore one attempt to improve the search results is to prohibit duplicate entries in the population pool. However, the results summarised in Figure 6.9 show no general noticeable improvement through this method. This option can only be used when there are enough possibilities to place the objects. Otherwise it might not be feasible to create a valid children set with no duplicates.

6.9 Counting extrema points

An option was implemented that has the following effect: every time two genotypes are compared (for example in the quicksort algorithm or parent selection) additionally to the compactness the amount of extrema points is considered. If two individuals fitness values differ only slightly (difference is smaller than 0.0000001) then the one that has less points with the currently best compactness is considered better. Less extrema points means that a smaller amount of gene changes are needed to achieve a smaller maximum compactness or a bigger minimum compactness.

Figure 6.10 compares solutions for runs with and without counting extrema points. Comparisions for different target functions were made. The runs with $\mu = \lambda = 50$ are also

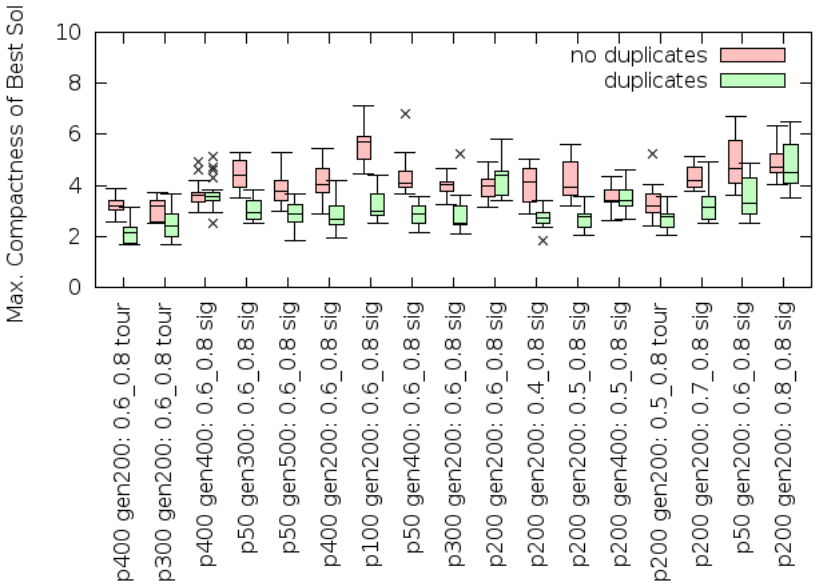


Figure 6.9: Comparison of solutions with and without removal of duplicates from runs for the minimum maximum compactness.

compared to runs with $\mu = \lambda = 100$. For the minimisation of the maximum it can be seen that although counting extrema points led to better solutions for the runs with the smaller population, doubling the population and children size led to even better results. Counting extrema points did not lead to notable improvements when added as option to the run with the higher population size. The option also shows a small improvement for the maximisation of the minimum and maximum.

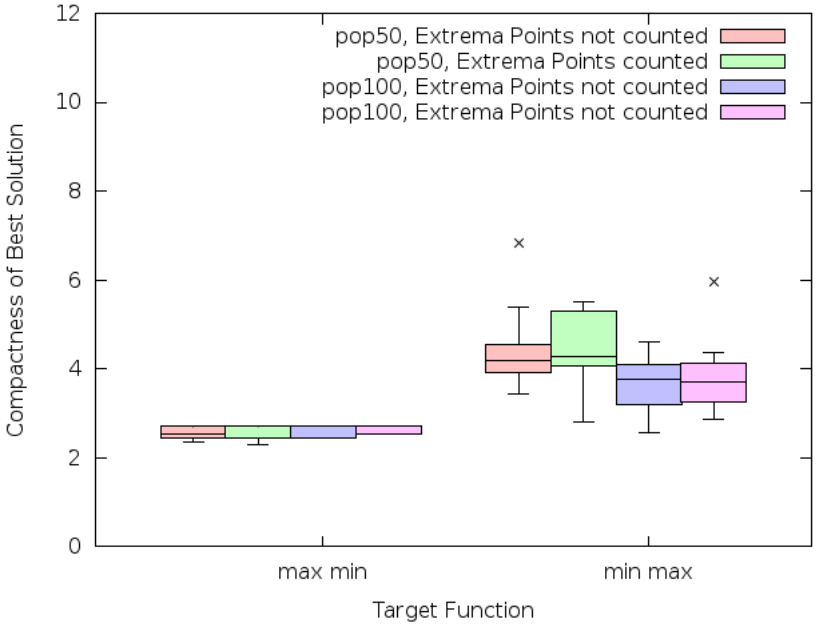


Figure 6.10: Qualities of solutions with and without counting extrema points for different target functions and two different population sizes.

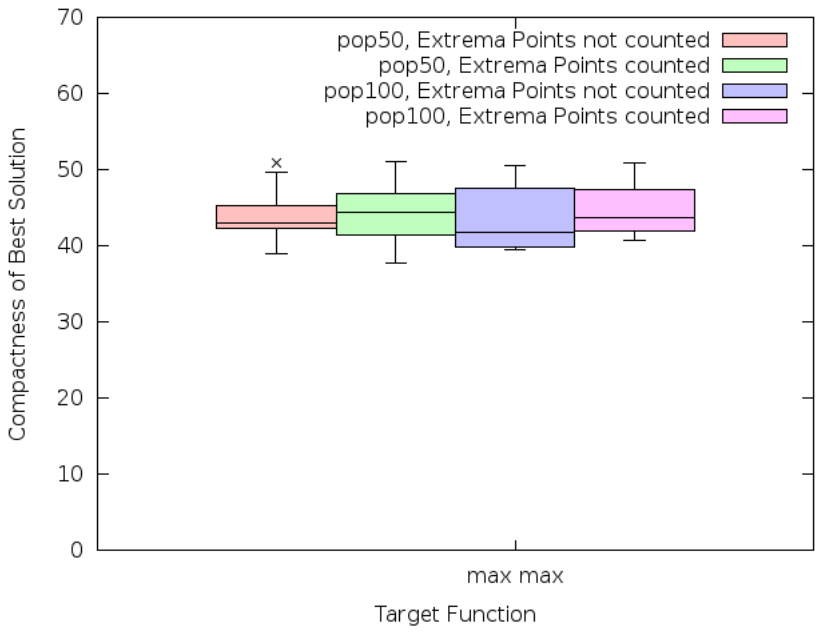


Figure 6.11: Qualities of solutions with and without counting extrema points for finding the maximal maximum and two different population sizes.

6.10 Influence of instance size scaling on run time

In order to investigate the influence of the size of the instance on the run time an instance was chosen and scaled to create two additional instances for comparison. The single "room" is quadratic and measures 30x30 fields. 20 objects with the shape shown in Figure 6.12 shall be placed. The two instances for comparison are "half" which is 15x15 and defines that 10 objects shall be placed and "double" which is 60x60 and defines that 40 objects shall be placed. Every run calculated 100 generations. The series was executing once for the minimum maximum and once for the minimum average.

As Table 6.1 shows, within the minimum maximum target function the ratios from small to room and room to double is quite constant. Regardless of the population number a run with an instance of the size doubled has a 22 to 27 times longer run time. Interestingly the ratios for the minimum average target function suggest that the up scaling of the instances had less effect on the minimum average instance and shows very coherent numbers. Runs for the room instance where 14 to 15 times slower than runs of the small instance while changing to the double instance made the run 17 times slower. Figure 6.13 shows the run times of room compared to half and double and a comparison of the three instances with different population sizes.

	Min Max		Min Avg	
population	**small : room**	**room : double**	**small : room**	**room : double**
50	1 : 22.2	1 : 21.2	1 : 14	1 : 17
100	1 : 27	1 : 21.8	1 : 14.4	1 : 17.2
200	1 : 23.5	1 : 23	1 : 15	1 : 17.3

Table 6.1: Ratio between instance run times of the minimum maximum and minimum average runs.

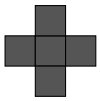
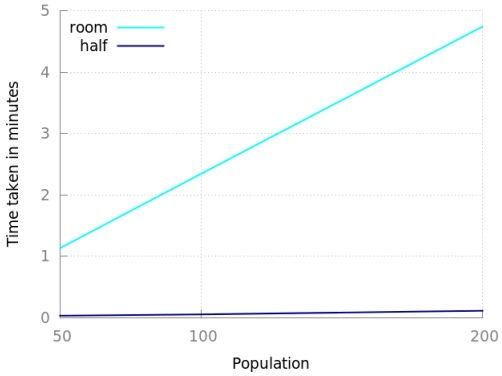
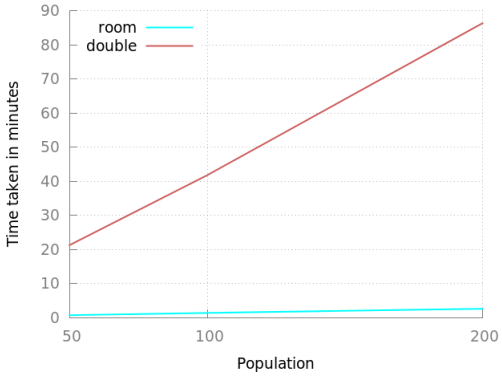


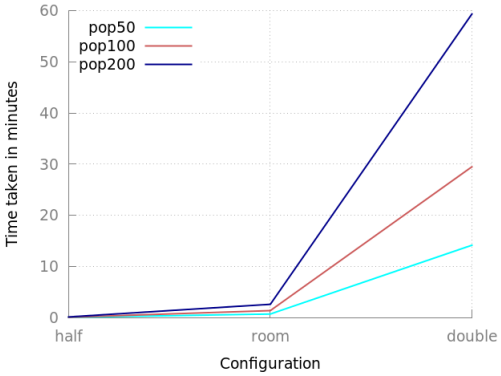
Figure 6.12: Object shape for the scaling test.



a) run time of room compared to half



b) run time of room compared to double



c) run time of the three instances compared by population sizes

Figure 6.13: Comparison of run times for the minimum maximum compactness between different instance sizes.

6.11 Scaling of complexity

In order to analyse the impact of a growing number of objects on the quality of results a small test series was performed. Several runs were performed for each configuration, the target function should minimise the maximum compactness. This series clearly shows that the amount of objects, which have to be placed to form an optimal placement together, has a big influence on the performance of the algorithm. Figure 6.14 provides an overview of the average of maximum compactness values reached with the different configurations.

In the first configuration only one big element (displayed in Figure 6.15a) had to be placed and all resulted in the same solution which is displayed in Figure 6.16a. While the environment (a rectangular room 18 cells high and 44 cells wide) stayed the same for all runs, the object configuration was altered.

For the second run the big object was cut in half and two identical objects (displayed in Figure 6.15b) had to be placed. Although some runs still yielded the same result like the first configuration, some other formations occurred. The two objects were stacked on top of each other, but the position of the formation in the environment varied, which led to a higher average compactness value (see example in Figure 6.16b).

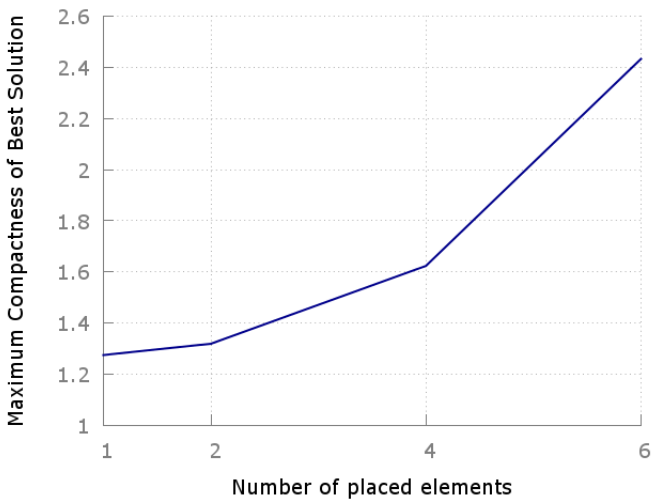


Figure 6.14: Comparison of the quality of the solutions for minimising the maximum compactness with different object numbers.

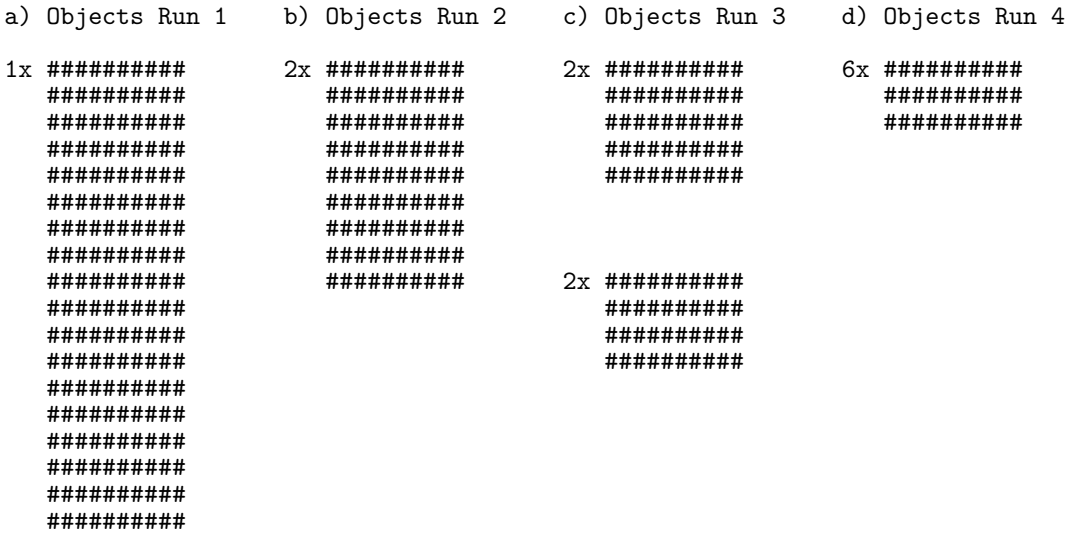
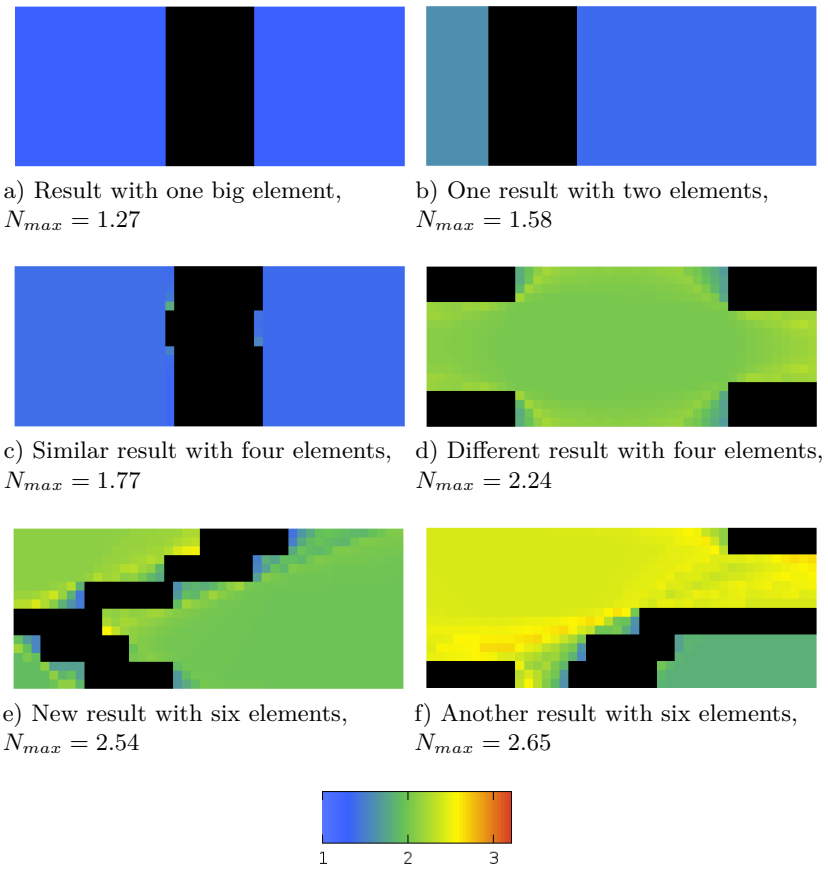


Figure 6.15: Objects that were placed for the complexity runs 1-4. The objects of each of the configurations b-d can be stacked up to the object of configuration a for run 1.

For the third run four elements of two different sizes (see Figure 6.15c) should be placed, they add up to the same shape again. Figures 6.16c and 6.16d show examples of newly occurring results of this configuration. Although the program still most of the time found solutions which stack all objects on top of each other, it does not achieve perfectly stacked objects often. Also, a solution with one object in each corner emerges.

The fourth configuration defines six objects of the same size (displayed in 6.15d which add up to the object of the first run again. Many different new placings occur of which two examples are displayed in Figures 6.16e and 6.16f).

At this small scale there is hardly any difference in the run times. But it is noticeable that the runs with four and six objects take about one second longer on average than the runs with one or two objects.



a) Result with one big element, $N_{max} = 1.27$

b) One result with two elements, $N_{max} = 1.58$

c) Similar result with four elements, $N_{max} = 1.77$

d) Different result with four elements, $N_{max} = 2.24$

e) New result with six elements, $N_{max} = 2.54$

f) Another result with six elements, $N_{max} = 2.65$

Figure 6.16: Example results of runs for the minimum maximum compactness with a different number of elements that all add up to the same object. The runs were performed with the following parameters: population: 15, children: 30, generations: 40. The colour key for compactness values is shown underneath.

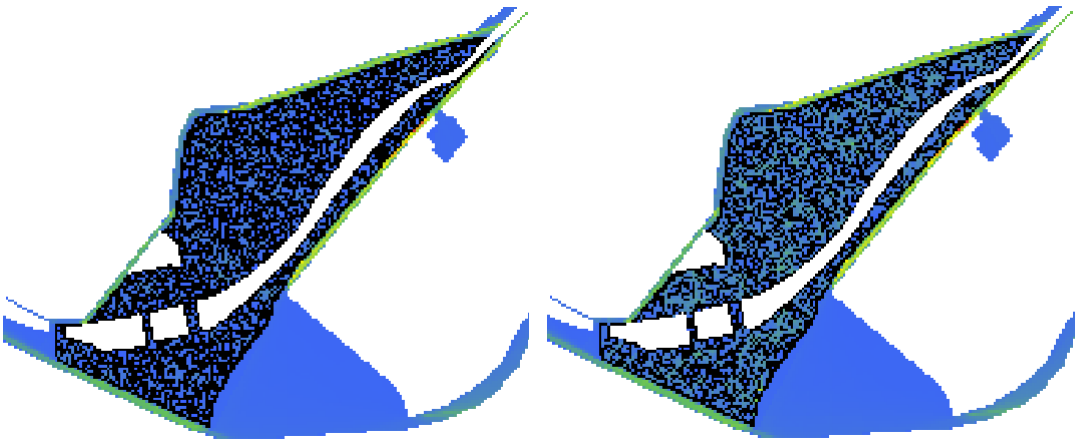
6.12 Build footprint and cross floor area

The overlap option allows objects to be placed above each other. One big instance (Figure 6.18) is used to analyse the effect of the overlap option when many objects that occupy only one field are placed. The dimensions of the central station instance are 193x162 and it resembles the central station in Vienna. Many of its unblocked cells are irreclaimable and therefore not available for placing. 12000 single cell objects are to be placed on this plan.

The fields that are occupied by objects form the building footprint of the solution. The floor area of the formed structures is the total amount of placed obstacles and therefore the same for any solution in this test series. We will have a look at the floor area ration (FAR) which is computed as the ratio of the floor area to the building footprint. The optimisation of compactness for single point target functions does not find improvements frequently. Since there are so many objects and points involved the chance that a random action leads to a better result is small. Therefore in this section only global properties are considered.

a) One solution for the maximisation of the average compactness for the central station instance. FAR: 3.94

b) One solution for the maximisation of the deviation of compactness values for the central station instance. FAR: 3.02



c) One solution for the minimisation of the average compactness for the central station instance. FAR: 2.55

d) One solution for the minimisation of the deviation of compactness values for the central station instance. FAR: 3.24

Figure 6.17: Comparison of placing solutions with overlaps for the central station instance.

The tests ran with a population of 50, 70 children and 200 generations. The maximisation of the average compactness led to the biggest average floor area ratio of 3.8. The FAR value of the maximisation of the deviation of compactness values is a lot lower (2.93 on average).

Figure 6.18: Instance: central station

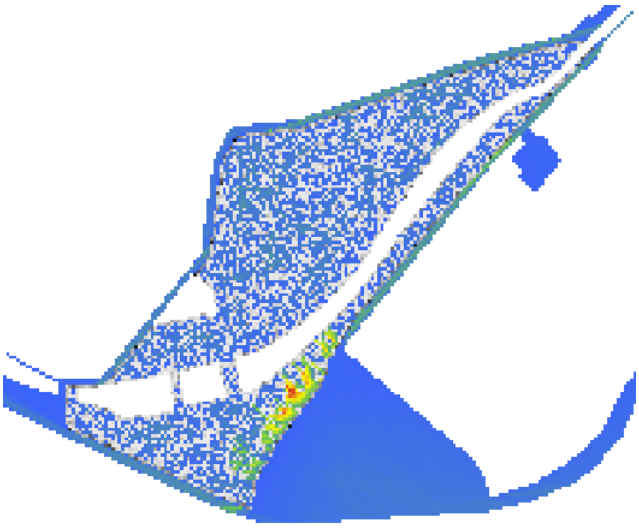


Figure 6.19: Heat map of a solution for a maximum average run of the main station instance. The compactness values of the free fields is coded as usual (blue for low values, red for high values) and the obstacles are shown in light grey for single obstacles up to black for the maximum obstacles overlapping in one place.

The second highest average FAR value of 3.45 was reached with the minimisation of the deviation. The minimisation of the average area only led to an average FAR of 2.54 (example solution in Figure 6.17c). When looking more closely at the two solutions of the minimum target functions it can be noticed that there are more small open spaces in the minimum deviation solution in Figure 6.17d which resembles the higher FAR value.

As can be seen in Figure 6.17a the maximum average solution shows more longer straight lines that contain higher compactness values which can hardly be found in the maximum deviation solution in Figure 6.17b. The figures show that the placing for the maximum average and maximum deviation make use of the free irreclaimable space in the bottom right of the environment. By placing the obstacles diagonal of each other many points with long narrow views into the free space emerge. They contain the highest compactness values in the maps. The solutions for the minimum average and minimum deviation on the other hand block the view into that free space with obstacles that were placed right next to each other instead of diagonal.

Figure 6.19 shows the obstacles coded in grey scale from light grey for few obstacles to black for the most overlapping obstacles. Most overlapping obstacles are collected at the border to area where no obstacles can be placed. This is probably because of the mechanism that tries to solve overlaps.

While there are noticable differences between the used target functions, 200 generations is not a lot for such a big instance. But the central station instance has many defined objects and overlaps are allowed. The optimal solution might be to put all obstacles on top of each other or to build on every available unblocked space. After a run with $\mu = \lambda = 50$ for 5000 generations of maximising the average compactness for example only few obstacles remained visible on the 2D map (see Figure 6.20). This might be an undesired effect. In order to guide the optimisation process it could be useful to set a maximum building hight.

6.13 Performance check against a local improvement

In order to evaluate if, after the end of the evolutionary algorithm, movements of one element exist that would lead to an improvement, a local search algorithm was created. This local

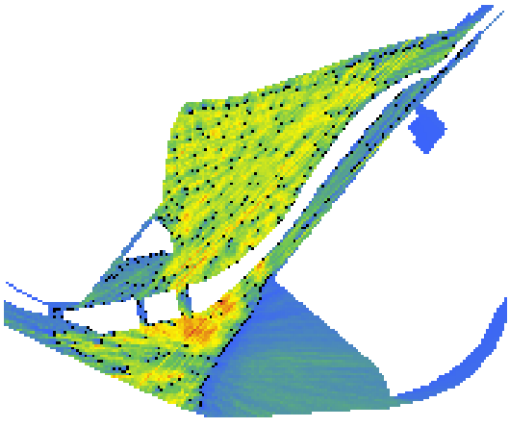


Figure 6.20: After 5000 generations most obstacles were placed on top of each other in order to generate a high average compactness.

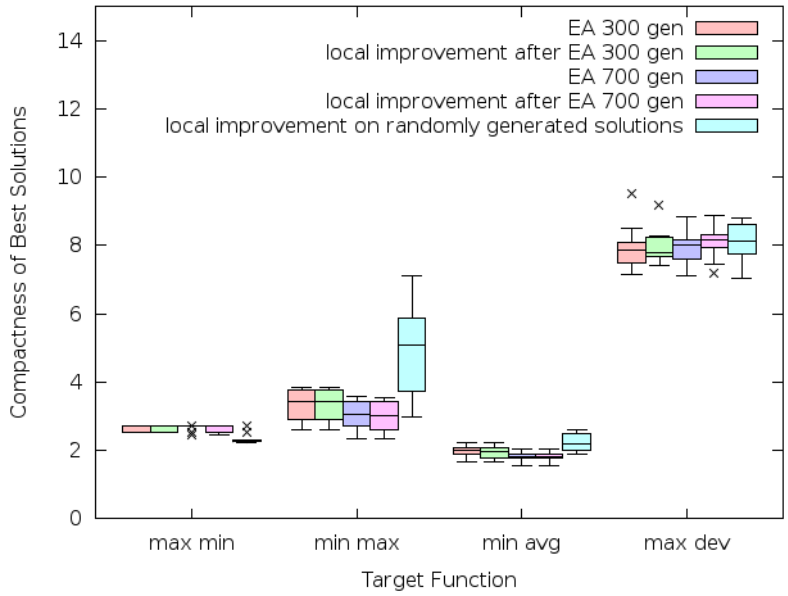


Figure 6.21: Comparison of EA runs with local improvement on the solutions of those runs and with local improvement on randomly generated solutions.

search tries to further improve a given placing solution. It chooses one placed object at random and evaluates each possible movement of it. It iterates through the environment fields in a spiral that starts at the element's original position. If a position is found that leads to a higher fitness value for the solution, the object is moved there and the next object is chosen. The algorithm ends if no improving movement can be found for any of the objects of the current solution.

Figure 6.21 shows a comparison of runs with the EA and local improvements. EA runs were executed with 300 and 700 generations. The local improvement was applied to all solutions afterwards. For comparison the local improvement was also applied to randomly generated solutions. Regarding the used target functions the local improvement after the EA runs was most useful for the maximisation of the deviation of compactness values. The rest of the improvement runs could not notable improve the solution quality. The comparison with the local improvement runs that were applied to randomly generated solutions show that a better starting solutions leads to a higher quality solution of the local improvement. Often by moving only one object no improvement can be found although better solutions exist.

CHAPTER 7

Heuristic Approach

7.1 Introduction

The evolutionary algorithm is useful to find good solutions but its usage has some drawbacks. Many parameters need to be configured. The results for the different environments show for example that the population size, children size and the generation number which work well for one environment might be too small for a bigger environment. The population number and children number also influence the performance of the group size for tournament selection. As Rittel and Webber [1] mentioned, time is an important factor and for an automatic planning tool it is important to present diverse solutions within a short period of time. Especially for larger environments the run time of the evolutionary algorithm takes several hours or even days. Therefore this chapter discusses the development of a fast construction heuristic approach which is able to produce solutions which often reach reasonable compactness values and produces different solutions within short time.

By analysing the results of the EA runs an emerging pattern can be discovered that supports high compactness values. This pattern can be used to develop simple rules for object placing for a heuristic. The Shadow Casting algorithm is used as a basis and is extended with object placing rules. This enables the heuristic to use field of vision information to decide where to place the objects. This technique results in a fast heuristic that can achieve interesting results in simple environments.

7.2 Analysis of evolutionary results

The analysis of common emerging picture for the different target functions was already presented in Section 5.2. As pointed out in that context, the minimisation of the compactness value (in discrete space) can be achieved by simply forming a rectangle around the point that shall be minimised. The points with maximum compactness show isovists which are shaped star-like with many long “fingers”. This shape is more complex and interesting. The heuristic shall take as input an environment and a point in that environment. It shall then place obstacles in a way that the preset point will have a high compactness. When looking at the position of placed objects around the vision points of isovists with high compactness, it stands out that the objects are placed diagonal to each other. They seem to be arrange in a diagonal line that allows the observer to look through between them. This results in many narrow vision lines. This observation will be used in the construction of the heuristic.

7.3 Basic deterministic heuristic

For simplicity we decide that the heuristic will be placing objects of the size of one cell. There is no specific number of objects that need to be placed. The heuristic decides on its own how many objects it needs to place in order to reach a high compactness. Since the end product shall produce different results it needs to be non-deterministic. However the first version will be a deterministic one.

For the heuristic the shadow casting algorithm can be reused. While calculating the field of vision it will also place obstacles based on the information it gathered so far. It will calculate the distance to the first visible obstacle in each direction. This value is then divided by a constant and the result is taken as the distance in which obstacles shall be placed. From the distance and the slope it is then possible to calculate where the first object shall be placed. Once this point is reached within the shadow casting algorithm an obstacle is placed on it. Afterwards it is calculated where the next point shall be placed. This is simply the cell diagonal of the last placed obstacle in order to achieve diagonal lines of obstacles which offer many small points through which the observer can look through.

In the following section the design process for the algorithm will be described in more detail.

Algorithm design

The diagonals that pass right through the vision point offer good possibilities for long lines of sight with a high perimeter value. In the maximum compactness point lit maps they are usually free of any placed obstacles. Therefore the heuristic shall not place any obstacle directly at the diagonals. The best point to start an obstacle line is therefore one place next to the diagonal in a distance to the vision point that blocks the view to the cells right next to the diagonal but leaves enough free space on the other side of it to produce more "fingers".

Assume that one object o is placed in distance d to the vision point p. The further the walls behind o are from p the bigger the non-visible area grows that lies in the shadow of o. If the shadowed area reaches a specific size then the compactness can be increased by removing o and placing two new obstacles diagonal of each other in a bigger distance to p than d. This will result in an additional visible area that can be seen when looking between the two new obstacles and therefore increases the compactness. The quality of the distance is therefore influenced by the distance to the environment obstacles. The heuristic will simple divide the distance between the vision point and the closest environment obstacle by a constant *distDivider*. In order to evaluate the quality of different *distDivider* values experiments with different distances are executed and analysed.

The heuristic needs to iterate through the cells around the vision point, for which the shadow casting algorithm is used. The heuristic is built upon it and extends the *castLight* Algorithm 5.8. Before the *castLight* call for each octant the variables *objOuterDist* and *objInnerDist* variables are set to 0. These contain the *deltaOuter* and *deltaInner* values at which the next obstacle shall be placed. This means that in the program part where it is decided if an obstacle shall be placed, the current *deltaOuter* will be compared to *objOuterDist* and the current *deltaInner* will be compared to *deltaInnerDist*.

After a cell was set visible a call to *calcObjToPlace* (see Algorithm 7.1) is inserted. If it is the first call for this octant it calculates the initial *objOuterDist* and *objInnerDist*. This is done by calculating the slope for the current cell which is then used to call *getDistToNearBlock*. This function calculates the nearest obstacle on the slope and returns its distance to the vision point. The distance is divided by the *distDivider* value and the result is assigned as the first *objOuterDist* value. The *objInnerDist* is simply one less than the *objOuterDist* so that the cell next to the diagonal on the side closer to the vision point is chosen.

Algorithm 7.1: calcObjToPlace

input : castLight state, direction d, boolean xInner
output: new to be placed object distance
$curSlope \leftarrow state.deltaInner/state.deltaOuter$;
$dist \leftarrow getDistToNearBlock(startx, starty, curSlope, d, xInner)$;
$dist \leftarrow round(dist/distDivider)$;
$objOuterDist \leftarrow dist$;
$objInnerDist \leftarrow objOuterDist - 1$;

After the call to *calcObjToPlace* the function *checkObjToPlace* is called. This function (shown in Algorithm 7.2) evaluates if an object shall be placed in the current cell and calculates the new *objOuterDist* and *objInnerDist* by reducing the inner distance and increasing the outer distance. The resulting cell in which the next obstacle shall be placed is therefore diagonal to the one in which the last object was placed.

Algorithm 7.2: checkObjToPlace

input : castLight state, direction d, boolean xInner
output: new to be placed object distance, new object placings
if $objOuterDist = abs(state.deltaOuter)$ & $objInnerDist = abs(state.deltaInner)$ **then**
    $objectPlacings.add(state.curX, state.curY)$;
    $objOuterDist++$;
    $objInnerDist--$;

Algorithm 7.3 shows how the nearest obstacle on a specific slope is found. Each iteration the distance *deltaOuter* to the origin point is increased by one. The coordinate of the outer direction is simply the distance while the coordinate of the inner direction is calculated using the distance and slope. If the analysed cell is blocking or not visible the distance is calculated with the Pythagorean theorem and returned.

Algorithm 7.3: getDistToNearBlock

input : vision point x, vision point y, slope, direction d, boolean xinner
output: distance to first obstacle on slope in the passed direction
$p \leftarrow Point(x, y)$;
$deltaOuter \leftarrow 1$;
while *true* **do**
    if *xInner* **then**
        $dY \leftarrow deltaOuter$;
        $dX \leftarrow deltaOuter * slope$;
    else
        $dX \leftarrow deltaOuter$;
        $dY \leftarrow deltaOuter * slope$;
    $p.y \leftarrow y + dY * d.deltaY$;
    $p.x \leftarrow x + dX * d.deltaX$;
    if *p is part of env obstacles, out of map or not visible* **then**
        return $\sqrt{deltaOuter^2 + (deltaOuter * slope)^2}$;
    $deltaOuter--$;

Evaluation

In the following section this heuristic solution is evaluated. As a first step a *distDivider* value needs to be chosen which is then used in the following test runs. Afterwards the solutions that are found by the heuristic are compared to some solutions which were found by the evolutionary algorithm.

Evaluation of divider value

In order to evaluate which divider values lead to good results several test runs with three different instances were performed. In each test environment one point was marked as the point at which the compactness shall be maximised. The results which are presented in Figure 7.1 show that a value of 4.7 leads to a local maximum in all three instances. In the BienalePavilion instance the marked point was closer to a wall than in the other two instances. Higher *distDivider* values lead to smaller distances to the vision point and very closely placed obstacles can block big parts of the vision field. Although there might be different optimum values for *distDivider* for different instances, 4.7 seems to lead good results for all used test cases and will be used in the following.

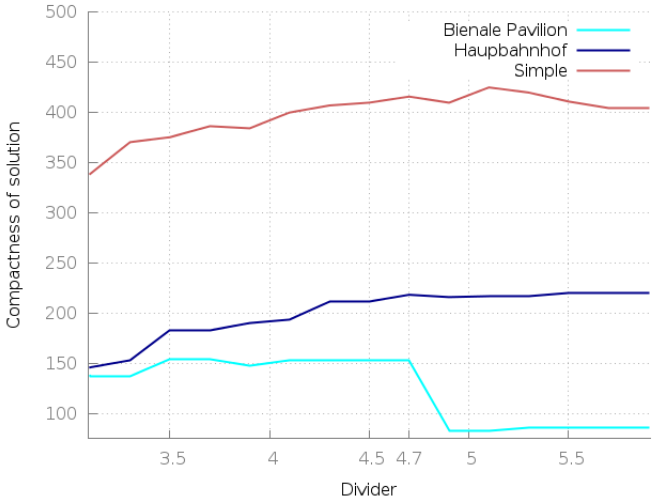
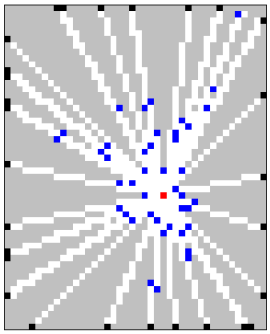


Figure 7.1: Compactness comparison of solutions of runs with different *distDivider* values.

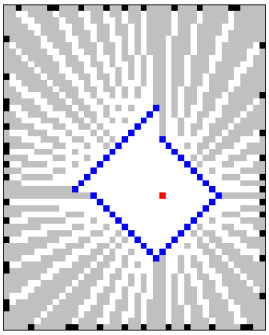
Comparison with evolutionary runs

After the *distDivider* value was chosen the heuristic is applied to different test instances. Additionally to instances which were used previously for the evaluation of the evolutionary algorithm, another instance was used. This instance, called “Simple”, is one square room, bigger than the Room instance and without any obstacles. Figure 7.2 shows an example result found with the evolutionary algorithm in comparison with the result that the heuristic calculated. The evolutionary algorithm had 70 objects available for placing, which are more than the heuristic used. The compactness the heuristic reached was even better than the example result that was found by the evolutionary algorithm.

Figure 7.3 shows the heuristic results for the Main Station instance for a point similar to the one that was found by the evolutionary algorithm. Please note that the heuristic ignores irreclaimable points which allowed it to set the obstacles a little bit further from the point than the evolutionary algorithm could. The evolutionary algorithm reached a compactness of 186.44, while the heuristic result has a compactness of 311.52 for a similar point. The evolutionary run for 5000 generations which found this solution ran for almost two days.



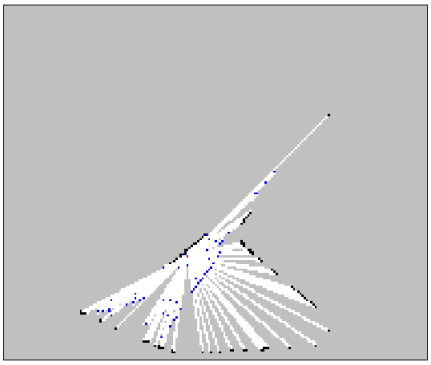
a) Lit map for a placing found by the evolutionary algorithm for the Simple instance using a population of 150 and 200 generations.
Compactness: 207.5



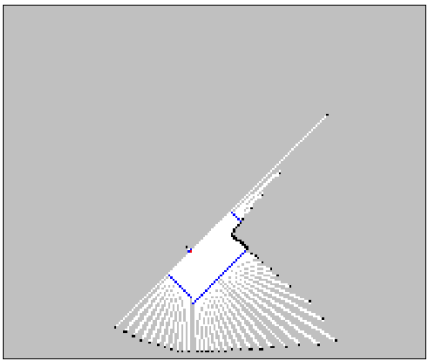
b) Lit map for the placing found by the deterministic heuristic for the Simple instance.
Compactness: 218.39

Figure 7.2: Lit maps for maximum compactness points of the Room instance. Red: vision point. Black: environment obstacles. Blue: placed obstacles. Grey: non-visible area.

The heuristic is able to provide results almost instantly. The open irreclaimable space in the south of the environment naturally helps to produce high compactness values when random objects are placed at the border to the irreclaimable space. This influences the evolutionary algorithm and maximum points always seem to be located in that area. But the heuristic expects that the point at which the compactness shall be maximised is provided as an input. Setting the point at another place shows that an even higher compactness can be reached. An example can be seen in Figure 7.4.



a) Lit map for a maximum compactness point for a maximum average run for the Main Station instance.
Compactness: 186.44



b) Lit map for the placing found by the deterministic heuristic for a similar point.
Compactness: 311.52

Figure 7.3: Lit maps for maximum compactness points of the Main Station instance. Red: vision point. Black: environment obstacles. Blue: placed obstacles. Grey: non-visible area.

While the heuristic returns better results than the evolutionary algorithm for the previous instances, it also has its drawbacks. The Bienale Pavilion instance contains environment obstacles around the middle of the room. If the point that shall be maximised is positioned close to these blocking objects then the distance to the wall is in those directions very small. This causes the obstacles to be placed very close to the vision point. The obstacles therefore block big parts of the view which results in a lower compactness. Figure 7.5 shows that only one of four sides is unblocked and offers longer lines of view.

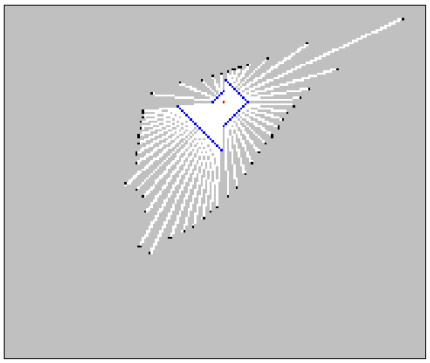
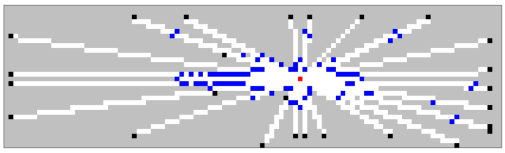
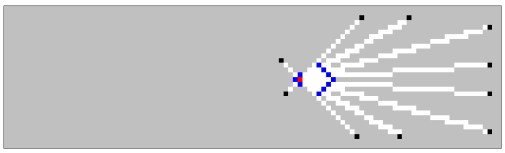


Figure 7.4: Lit map for the placing found by the deterministic heuristic for a different point of the Main Station instance. Compactness: 415.72



a) Lit map for a maximum point found by the evolutionary algorithm for the Bienale Pavilion instance.
Compactness: 202.51



b) Lit map for the placing found by the deterministic heuristic for a similar point.
Compactness: 91.3

Figure 7.5: Lit maps for maximum compactness points of the Bienale Pavilion instance. Red: vision point. Black: environment obstacles. Blue: placed obstacles. Grey: non-visible area.

7.4 Introducing diversity: a random factor

Since automatic planning tools shall present different solutions which provide new ideas to the planner, a random factor needs to be implemented. The basic deterministic function calculates the distance between the vision point and the walls on the diagonal once for each octant. But just one or several steps away from the diagonal the distance to the wall following the new slope could already be very different. Recalculating the distance for every slope would result in placings that do not follow the diagonal placing pattern and therefore in a lower compactness. But recalculating at some good points could also lead to a higher compactness. This will be used for the random component in the following solution. An option will be implemented into the *checkObjToPlace* function (extension shown in Algorithm 7.4).

Algorithm 7.4: checkObjToPlace

```
input  : castLight state, direction d, boolean xInner
output: new to be placed object distance, new object placings
if objOuterDist = abs(state.deltaOuter) and objInnerDist = abs(state.deltaInner)
then
    objectPlacings.add(state.curX, state.curY);
    lastODistInner = objInnerDist;
    lastODistOuter = objOuterDist;
    objOuterDist++;
    objInnerDist--;
    if rand > limit then
        newDistanceCalculation(state, d, xInner);
```

If a produced random number exceeds a preset limit then the distance will be calculated for

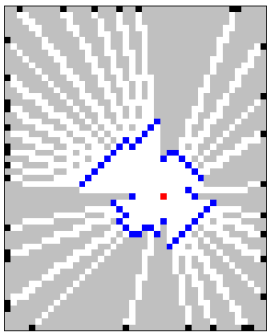
the current slope. The object's distances in inner and outer direction are saved before they are adjusted. The recalculation function is shown in Algorithm 7.5. The next object will be placed one cell closer to the vision point in inner direction. This ensures that one object is placed per inner distance. If the new outer distance is the same like the previous one it is increased by one. This shall prevent that two objects are placed right next to each other forming a blocking row or column. If the new *objOuterDist* is smaller than the current distance the *castLight* algorithm operates with then the octant needs to be relaunched because the distance in which the object shall be placed has already been processed completely. In case of a relaunch the obstacles that were placed so far are kept and the *objOuterDist* and *objInnerDist* values are kept as well.

Algorithm 7.5: newDistCalculation

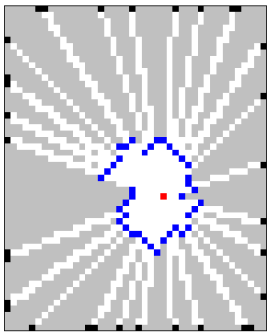
input : castLight state, direction d, boolean xInner
output: new to be placed object distance, if octant needs to be relaunced
$curSlope \leftarrow state.deltaInner/state.deltaOuter$;
$dist \leftarrow getDistToNearBlock(startx, starty, curSlope, d, xInner)$;
$dist \leftarrow round(dist/distDivider)$;
$objOuterDist \leftarrow dist$;
$objInnerDist \leftarrow abs(deltaInner) - 1$;
if $lastODistOuter = objOuterDist$ **then**
  $objOuterDist++$;
if $objOuterDist < state.distance)$ **then**
  $state.relaunchOctant \leftarrow true$;

Figure 7.6 and 7.7 each show two different examples of the non-deterministic heuristic for the Simple and the Main Station instances. We have seen bigger compactness values in the previous solutions which were found by the evolutionary algorithm or the deterministic heuristic. But the values are still at the higher end and the solutions present different ideas. The algorithm is fast and can produce ten different results in less than one second for instances of this size.

Especially in bigger environments it can be interesting to set more than one point at which the compactness shall be maximised. The Main Station offers a lot of space and includes different sections which fields of visions only overlap slightly. Figure 7.8 shows a solution for an example in which 14 maximising points were set. They were processed sequentially with each one treating the obstacles that were set for the previous points like environment obstacles. Figure 7.8a displays the compactness map for the environment and all obstacles that were placed while Figure 7.8b shows the merged visible area for all 14 vision points.

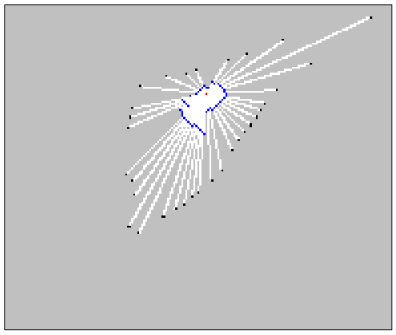


a) Lit map for a solution found by the heuristic with a random component.
Compactness: 193.36

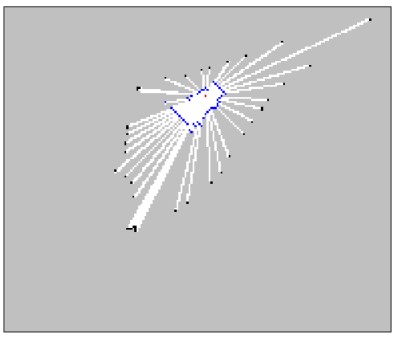


b) Lit map for another solution found by the heuristic with a random component.
Compactness: 194.46

Figure 7.6: Lit maps for two solutions for the Simple instance found by the heuristic using a random component. Black: environment obstacles. Blue: placed obstacles. Grey: non-visible area.

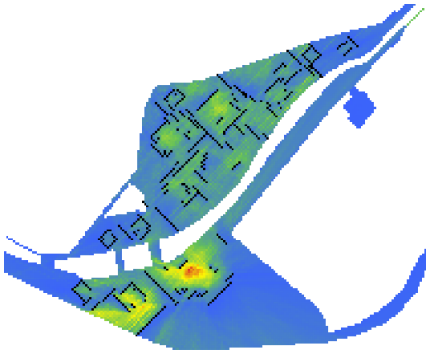


a) Lit map for a solution found by the heuristic with a random component.
Compactness: 373.4



b) Lit map for another solution found by the heuristic with a random component.
Compactness: 302.22

Figure 7.7: Lit maps for two solutions for the Main Station instance found by the heuristic using a random component. Black: environment obstacles. Blue: placed obstacles. Grey: non-visible area.



a) Compactness map for the environment with all obstacles set for all maximisation points.

b) Lit map for the same solution that shows in white the area that is visible from any of the maximisation points, which are marked in red.

Figure 7.8: Solution for the Main Station instance in which a maximising compactness solution was calculated for 14 points sequentially.

CHAPTER 8

Conclusions and Future Work

8.1 Conclusions

For this master thesis two algorithms were developed that place multiple objects in a discrete 2D environment. The first one is an EA and optimises diverse objective functions based on compactness values in the solution environment. A version of the Shadow Casting algorithm was used to calculate the fields of vision that were needed to calculate the compactness.

The exploration of the parameter settings showed that settings with increased generation number reached a better solution quality. For higher generation numbers an increase of the population size led to more significant improvements than a further increase of the generation number. This may indicate that the search space needs to be explored broadly in order to find the best solutions. This is probably linked to the solution representation. Moving one obstacle slightly can lead to a big change in compactness and therefore fitness value. As was shown by comparing the EA performance with a local improvement algorithm it is common that although moving one obstacle of a solutions leads to no improvement better solutions might exist. A solution representation which has the property that small changes in an individual lead to small changes in the fitness value would be desirable. But the results also showed that for most of the used target functions the EA still performs better than a simple local improvement on a random solution. Reproducible patterns emerged for the different evaluated target functions which, again, confirmed the usability of the compactness in target functions and provided the possibility to explore the properties of different solutions.

In the developed EA the mutation rate was implemented as an overall chance to mutate one gene. If many obstacles have to be placed the mutation of only one gene has only a small chance to influence the target function value, especially if a local property should be optimised. To counter this shortcoming multiple mutation steps were executed depending on the amount of obstacles to be placed. This problem could be overcome by using a mutation rate as mutation chance per gene instead, which is common practice for EAs but was not used in this thesis.

An architectural designer needs to explore different solutions. While the EA did produce interesting and useful results it took several hours to provide results for big instances, which often is not practical. Based on the analysis of the patterns that the solutions of the EA showed for the maximisation of the maximum compactness, a heuristic was developed. This second algorithm is based on the Shadow Casting algorithm and can produce solutions to maximisation placing problems with simple objects very fast. Multiple points can be defined and the objects are placed in a way such that those points reach a relatively high compactness.

The heuristic places the objects in a distance to a maximisation point that is based on the distance between the point and the closest wall in the direction of the obstacle placement. It

was demonstrated that the heuristic can be used to create interesting placings that offer points with high compactness values and enclosing but not fully connected structures. The heuristic was created in order to calculate solutions with high compactness areas which offer spiky fields of vision. The experiments show that this heuristic can be successfully used to generate vision points with a high compactness. The heuristic can also be executed multiple times for different points in one environment sequentially which creates a solution that contains several points with a higher than average compactness. The amount of maximisation points as well as their distance to each other can influence the pattern of the obstacle placings and the scale of the compactness values.

This thesis showed one possibility to development a fast heuristic that is based on the Shadow Casting algorithm. Such heuristics can enhance the working process of architects by providing multiple inspiring solutions within short time.

8.2 Future Work

The EA offers an option to allow obstacles to be stacked above each other. If this option is activated the best solutions may contain only a few points on which many stacking obstacles were placed. In such cases it could be interesting to employ a maximum building hight in order to control the amount of obstacles that can be placed above each other.

While the heuristic did show interesting and satisfying results, it was demonstrated that in the used approach walls that are placed close to the maximisation point can lead to view blocking placings. Those configurations lead to compactness values which are a lot lower than the maximum that can be reached with a better placing. Improvements or adjustments for such special cases would be necessary. It may be useful to compare the current line of sight length to the line of sight length to the left and right in order to find out if placing an obstacle at this position would block a long view line fully instead of dividing it into two. In this case the placing positions should be adjusted.

Low compactness values for fields of vision of points can be reached by enclosing the points with a closed rectangular object placing. The heuristic could be extended to providing both, low and high compactness values, by adding rectangular walls around minimum points. The distance of the walls to the minimum point and the decision if the maximisation should be done before or after the walls for the minimum points was set would depend on the application. A high average compactness might be reached by combining several maximum point creations. The optimal amount and location of those points need to be investigated.

The heuristic decides on its own how many single point objects it will place in the world. More objects could easily be placed without changing the compactness values of the maximum points if they are placed in an area that is not part of any of the fields of vision of the maximum points. Many extensions could be added to the heuristic in order to guide the placing process. For example defining a minimum or maximum distance of obstacles to the maximum points. This extension could be added in the *newDistCalculation* by adjusting the calculated *dist* value if it is out of the set boundaries.

A different approach could be to design desired fields of vision for environments and create the object placings in such a way that they form those fields of vision. Such a reverse approach could be promising to create environments with customised fields of vision.

In the Chapter 2 it was discussed that the representation of real environments in discrete space can have a big effect on its compactness. In continuous space a circle has the lowest possible compactness. In discrete space the lowest compactness is reached by any rectangle. This has to be considered for real life applications. If rounded corners or rooms shall be considered it might be necessary to develop a solution in continuous space.

Bibliography

[1] Rittel HWJ and Webber MM. *Dilemmas in a General Theory of Planning.*
Elsevier Scientific Publishing Company, Policy Sciences 4, Pages 155-169, 1973

[2] Donath D, König R, Petzold F, Schneider S and Knecht K. *KREMLAS - Entwicklung einer kreativen evolutionären Entwurfsmethode für Layoutprobleme in Architektur und Städtebau.*
Verlag der Bauhaus-Universität Weimar, 2012

[3] Batty M. *Exploring isovist fields: space and shape in architectural and urban morphology.*
Environment and Planning B: Planning and Design, volume 28, Pages 123-150, 2001

[4] Turner A, Doxa M, O'Sullivan D and Penn A. *From isovists to visibility graphs: a methodology for the analysis of architectural space.*
Environment and Planning B: Planning and Design 28(1), Pages 103–121, 2001

[5] Conroy and Dalton. *Spatial Navigation in Immersive Virtual Environments.*
PhD Thesis, University College London, 2001

[6] Wiener JM, Hölscher C, Büchner SJ and Konieczny. *Gaze Behaviour during Space Perception and Spatial Decision Making.*
Psychological Research, Springer, Pages 1-17, 2011

[7] Schneider S and König R. *Exploring the Generative Potential of Isovist Fields.*
Generative Design - Volume 1 - eCAADe 30, Pages 355-364, 2012

[8] Benedikt ML. *To take hold of space: isovists and isovist fields.*
Environment and Planning B, Volume 6, Pages 47-65, 1979

[9] DeLanda M. *Deleuze and the Use of the Genetic Algorithm in Architecture.*
Between Bladerunner and Mickey Mouse: New Architecture in Los Angeles Exhibition, 2001

[10] Tandy CRV. *The isovist method of landscape survey.*
Symposium: Methods of Landscape Analysis, Pages 9-10, 1967

[11] Koch D. *Isovists revisited. Egocentric space, allocentric space, and the logic of the Mannequin.*
Proceedings: Eighth International Space Syntax Symposium, Santiago de Chile Pontifical Catholic University of Chile, Paper Ref #8144, 2012

[12] Bergström B. *FOV using recursive shadowcasting.*
`http://www.roguebasin.com/index.php?title=FOV_using_recursive_shadowcasting`, visited on 08.12.2015

[13] Jones G. *Genetic and Evolutionary Algorithms.*
Encyclopedia of Computational Chemistry. John Wiley & Sons, 1998

[14] Bäck T, Hammel U and Schwefel HP. *Evolutionary Computation: Comments on the History and Current State.*
IEEE Transactions on Evolutionary Computation Volume 1, No. 1, Pages 3-17, 1997

[15] Rozenberg G, Bäck T and Nok J. *Handbook of Natural Computing*
Springer, 2012

[16] Bäck T, Foussette C and Krause P. *Contemporary Evolution Strategies.*
Springer, Natural Computing Series, 2013

[17] Eiben AE, Smith JE. *Introduction to Evolutionary Computing.*
Springer, Natural Computing Series, 2003

[18] Simon D. *Evolutionary Optimization Algorithms.*
John Wiley & Sons, 2013

[19] Mitchell M. *An Introduction to Genetic Algorithms.*
MIT Press, 1998

[20] Kruse R, Borgelt C, Klawonn F, Moewes C, Steinbrecher M and Held P. *Computational Intelligence.*
Springer, Texts in Computer Science, 2013

[21] Frazer J. *An Evolutionary Architecture.*
Architectural Association, London, 1995

[22] Knecht K. *Generierung von Grundriss-Layouts mithilfe von Evolutionären Algorithmen und K-dimensionalen Baumstrukturen.*
D. Donath und R. König, Working Papers Informatik in der Architektur, Nr. 9, 2011

[23] Romero J and Machado P. *The art of artificial evolution : a handbook on evolutionary art and music.*
Springer, Natural Computing Series, 2008

[24] O'Neill M and Ryan C. *Grammatical Evolution.*
IEEE Transactions on Evolutionary Computation, Vol. 5, No. 4, Pages 349-358, 2001

[25] Quiroz JC, Louis SJ, Banerjee A and Dascalu SM. *Towards creative design using collaborative interactive genetic algorithms.*
IEEE Congress on Evolutionary Computation, Pages 1849-1856, 2009

[26] Coates PS and Hazarika L. *The use of Genetic Programming for applications in the field of spatial composition.*
Proceedings of the 2nd Generative Art Conference (GA1999), Milan: Generative Design Lab Milan Polytechnic University, 1999

[27] Elezkurtaj T. *Evolutionäre Algorithmen zur Unterstützung des kreativen architektonischen Entwerfens.* PhD dissertation.
Vienna University of Technology, Institut für Architekturwissenschaften, 2004

[28] Howard E. *Improved Shadowcasting in Java.*
`http://www.roguebasin.com/index.php?title=Improved_Shadowcasting_in_Java`, visited on 08.12.2015