



Forschungsberichte

itke

32

Miloš Dimčić

# Structural Optimization of Grid Shells

Based on Genetic Algorithms







Forschungsberichte

itke

aus dem Institut für Tragkonstruktionen  
und Konstruktives Entwerfen,  
Universität Stuttgart

Herausgeber:  
Professor Dr.-Ing. Jan Knippers

Institut für Tragkonstruktionen und Konstruktives Entwerfen:  
Forschungsbericht 32

**Miloš Dimčić:**

Structural Optimization of Grid Shells  
based on Genetic Algorithms  
Stuttgart, August 2011

ISBN 978-3-922302-32-2

D 93

© Institut für Tragkonstruktionen  
und Konstruktives Entwerfen  
Universität Stuttgart  
Keplerstraße 11  
D-70174 Stuttgart



Alle Rechte, insbesondere der Übersetzung, bleiben  
vorbehalten. Vervielfältigung jeglicher Art, auch auszugs-  
weise, ist nicht gestattet.



# Structural Optimization of Grid Shells Based on Genetic Algorithms

Von der Fakultät Architektur und Stadtplanung der Universität  
Stuttgart zur Erlangung der Würde eines Doktors der  
Ingenieurwissenschaften (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von

Miloš Dimčić

aus Belgrad, Serbien

Hauptberichter: Prof. Dr.-Ing. Jan Knippers  
Mitberichter: Prof. Dr.-Ing. Kai-Uwe Bletzinger

Tag der mündlichen Prüfung: 29. Juli 2011

Institut für Tragkonstruktionen und Konstruktives Entwerfen der  
Universität Stuttgart, 2011



If you think you can do a thing  
or think you can't do a thing,  
you are right.

Henry Ford



# Acknowledgments

I would like to thank my supervisor Prof. Dr.-Ing. Jan Knippers whose expertise and guidance made the development of new ideas possible and whose openness for those ideas led to the practical applications of my work. I thank Prof. Dr.-Ing. Kai-Uwe Bletzinger for his helpful suggestions and valuable advice coming from the deep understanding of the optimization problems and techniques. In addition I thank Prof. Dr.-Ing. Manfred Bischoff. Our few, but substantial, discussions regarding the statical aspect of the grid shell optimization were invaluable for the development of my work.

I would like to thank my colleagues from the Institute of Building Structures and Structural Design for providing a pleasant and fun working environment. Many suggestions coming from them were a very important part of the research process and for that I am very grateful. Special thanks to Petra Heim for the help with all the bureaucratic problems, thus making my life much easier.

Thanks to Ramaseshan, an employee of Oasys Limited, whose instructions on how to use their software were of great help in the development of the research process and thanks to Marie Lienhard for correcting my English and making my writings sound better and more eloquent.

I am grateful to my friends in Stuttgart for their moral support and making my time here pleasurable: Sandra, Milos, Zoca, Mirko, Drale. Thanks to my friends Goran, Drobnjak and Liske for their support from three distant corners of the world (Chicago, Berlin, Brisbane). I wish to thank my uncle Dimce for his help and support, my cousin Aleksandra for the joyful times we had together, as well as my cousins Toma and Vlada for all the interesting discussions about life and science, always providing me with new ideas. My deepest gratitude goes to Elena for her love and support during this endeavor. Finally, I would like to thank my mother, my father and my three sisters: Sonja, Anastasija and Rebeka for their encouragement and unconditional love.

*This work was done with a financial support from DAAD (Deutscher Akademischer Austausch Dienst) under the index number A/08/91066*

---

# Contents

<b>Acknowledgments</b>	<b>III</b>
<b>Abstract</b>	<b>IX</b>
<b>Zusammenfassung</b>	<b>XI</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Free Form Design . . . . .	1
1.1.1 Goals and Capabilities . . . . .	1
1.1.2 Why free form? . . . . .	3
1.1.3 Free Form Today . . . . .	5
1.2 Grid Shell as a Structural System . . . . .	5
1.2.1 Development of Grid Shells . . . . .	5
1.2.2 FEM and CNC . . . . .	11
1.2.3 Application of Grid Shells . . . . .	12
<b>2 State of the Art</b>	<b>13</b>
2.1 Grid Shell Design Methods . . . . .	14
2.1.1 Geometrical Approach . . . . .	14
2.1.2 Statical Approach . . . . .	19
2.2 Structural Optimization . . . . .	20
2.2.1 Basic Terms . . . . .	21
2.2.2 Optimization Types . . . . .	22
2.2.3 Calculus Based Optimization Techniques . . . . .	23
2.2.4 Stochastic Methods . . . . .	24
<b>3 From Surface to Grid</b>	<b>29</b>
3.1 NURBS . . . . .	29
3.1.1 Design . . . . .	29
3.1.2 Mathematics . . . . .	30
3.2 Voronoi Diagram . . . . .	35
3.2.1 Definition . . . . .	35
3.2.2 Why the Voronoi Diagram . . . . .	35
3.3 Voronax . . . . .	38
3.3.1 Force Density method . . . . .	39
3.3.2 Constrained Force Density Method . . . . .	40
3.3.3 Voronoi to Voronax . . . . .	43

3.3.4	Why <i>Relaxed</i> Meshes Are Better Than <i>Unrelaxed</i> ones . . . . .	45
3.3.5	Why Voronax Is Better Than Voronoi . . . . .	45
<b>4</b>	<b>Genetic Algorithms</b>	<b>49</b>
4.1	Algorithm . . . . .	49
4.1.1	Basic Structure . . . . .	50
4.1.2	Selection . . . . .	54
4.1.3	Crossover . . . . .	55
4.1.4	Mutation . . . . .	57
4.2	Grid Shell Genotype and Phenotype . . . . .	58
4.2.1	Chromosome . . . . .	58
4.2.2	Decoding Functions . . . . .	60
4.2.3	FEM Setup . . . . .	66
4.3	Fitness Functions . . . . .	71
4.3.1	Geometrical (Optical) Functions . . . . .	72
4.3.2	Statical Functions . . . . .	77
4.3.3	Fitness Scaling . . . . .	83
4.4	Penalty Functions . . . . .	86
4.4.1	Method . . . . .	87
4.4.2	Application . . . . .	90
4.4.3	Examples . . . . .	92
4.5	Multi-Objective Optimization . . . . .	96
4.5.1	Pareto Optimum . . . . .	97
<b>5</b>	<b>Results</b>	<b>103</b>
5.1	Different Fitness Functions . . . . .	107
5.1.1	Sum of Von Mises Stresses . . . . .	108
5.1.2	Sum of Displacements . . . . .	113
5.1.3	Load Buckling Factor . . . . .	115
5.1.4	Multi-Objective Optimization . . . . .	116
5.2	Patterns . . . . .	121
5.2.1	Start With Voronax . . . . .	122
5.2.2	Recognizing The <i>Intention</i> . . . . .	124
5.2.3	Design Our Own Grid Shell . . . . .	127
5.2.4	About the Orientation of the Structural Members - Paths and Guide Lines . . . . .	131
5.3	Load . . . . .	136
5.3.1	Case 1 : Gravitational Load . . . . .	137
5.3.2	Case 2 : Horizontal Load . . . . .	141
5.3.3	Case 3 : Partial Load . . . . .	142
5.3.4	Multiple Load Cases Comparison . . . . .	145
5.4	Support . . . . .	146
5.4.1	Case 1 : All Edges . . . . .	147
5.4.2	Case 2 : One Edge . . . . .	150
5.4.3	Case 3 : Partial Support . . . . .	152



5.4.4	Case 4 : All Edges - Movable . . . . .	153
5.5	Different Sections . . . . .	156
5.6	Summary . . . . .	159
<b>6</b>	<b>Nature</b>	<b>161</b>
<b>7</b>	<b>Conclusions</b>	<b>167</b>
<b>A</b>	<b>Application</b>	<b>177</b>
A.1	User Dialog . . . . .	177
A.2	Export Files . . . . .	179
A.3	Draw Results . . . . .	180
<b>B</b>	<b>Code Structure</b>	<b>183</b>
B.1	Software and Methods Used . . . . .	183
B.2	Data Structures . . . . .	184
B.2.1	Individual . . . . .	184
B.2.2	The Algorithm . . . . .	185
<b>C</b>	<b>Cell Recognition</b>	<b>187</b>



# Abstract

In the 21<sup>st</sup> century, as free form design gains popularity, free-form grid shells are becoming a universal structural solution, enabling merger of structure and facade into a single layer - a skin [31]. The subject of the presented work is the optimization of grid structures over some predefined free form shape, with the goal of generating a stable and statically efficient structure. It is shown how combining design and FEM software in an iterative, Genetic Algorithms based, optimization process, stress and displacements in grid shell structures can be significantly reduced, whereby material can be saved and stability enhanced.

Within this research, design and static analysis software are combined in order to perform a statical optimization of grid shells, generated over a given free form surface. A plug-in for Rhinoceros 3D (software based on NURBS [44] geometry representation) is developed, that uses Genetic Algorithms as an optimization method and implements automated iterative calls to Oasys GSA (commercial FEM static analysis software) in order to generate a statically optimal grid shell. To make this possible, within this research some new types of automatic grid generation are developed. Voronoi diagrams [11] were used together with the adapted Force-Density method [38] to develop a new type of grid structure that we called Voronax. In the presented work it was shown that, using the same free form surface, and using the same number of joints and structural members, we can generate much more efficient grid shells, when compared to the standard (uniform) grid structures, simply by modifying the structural grid, i.e., rearranging the structural members of the grid shell.

The work presented offers an explanation of the entire method and how it can be constructed. The results of the experiments are there to prove its efficiency and credibility. Once it is proved that the method works, its application can take various forms and be left to the creativity of the user and the requirements of the specific project.



# Zusammenfassung

## Motivation

Die Produktion von Bauteilen durch CNC (Computer Numerical Control) - gesteuerte Maschinen revolutionierte, gegen Ende des 20. Jahrhunderts, den architektonischen Entwurf. Tragwerke mussten nicht mehr vereinfacht werden, um so viele gleiche Elemente wie möglich zu erhalten. Heutzutage können Gebäude mit Tausenden verschiedener (einzigartiger) Trag- oder Fassadenelemente zu akzeptablen Kosten gebaut werden, indem der ganze Prozess automatisiert wird. Die Automatisierung bezieht sich in diesem Fall auf die Programmierungstechniken, die automatische Zeichnung, statische Berechnung und Herstellung individueller Elemente möglich macht, wo der "manuelle" Prozess (ein Teil nach dem anderen) zu viel Zeit in Anspruch nehmen würde. Diese Revolution im Herstellungsprozess erlaubte eine größere Freiheit im Entwurf. So genannte Freiformen, wie sie hauptsächlich im Industriedesign verwendet werden, konnten nun auch in der Architektur zu erschwinglichen Kosten eingesetzt werden. Die Tatsache, dass die Generierung der Struktur auf Basis freier Formen zu lauter Einzelstücken führt, ist nicht länger ein Hemmnis. Aber mit der Freiheit kommt die Verantwortung und die Frage effizienten Entwerfens freier Formen ist die Grundlage der vorgelegten Forschung.

Die Gitterschale ist eine Tragstruktur, bei welcher man versucht, das Verhalten einer Schale mit der Gitterform zu kombinieren. Die Kräfte sollen axial (mit so wenig Biegung wie möglich) über die Oberfläche verteilt werden und gleichzeitig soll dies mit einer Gitterstruktur geschehen, weil diese vorgefertigt und einfach zusammengesetzt werden kann. Dies erweist sich als gute Lösung, da Gitterschalen große Distanzen mit einer leichten einlagigen Konstruktion überspannen können. Das Gitter, das üblicherweise aus Stahlelementen besteht, ist oft mit Glas gedeckt, hoch transparent und somit geeignet für Dachkonstruktionen (Freiform-Dachkonstruktionen), wie in Bild 0.1 zu sehen. Gitterschalen werden gegenwärtig meist so entworfen, dass die Stäbe gleichmäßig verteilt werden (wie in den Beispielen in Bild 0.1). Aber schon die Intuition sagt uns, dass für eine unregelmäßige Form die beste und statisch wirksamste Verteilung nicht regelmäßig sein kann. Auch Tragwerkslösungen, die man in der Natur findet, zeigen uns, wie die Dichte in tragenden



**Bild 0.1:** **left**-Great Court, British Museum, London, 2000  
**center**-Sun Valley, EXPO Shanghai, China, 2010  
**right**-Septemberplein 18, Eindhoven, Niederlande, 2008.

Elementen an die entsprechenden Einflüsse von außen angepasst ist und so der Materialeinsatz gering gehalten wird. Dies ist der Punkt, an dem die Idee für diese Arbeit geboren wurde.

Da wir nicht mehr durch den Zwang zur Uniformität der Bauteile eingeschränkt werden (sie können alle einmalig sein) gehen wir mit enormen Mengen möglicher Lösungen um, z.B. mögliche Gitterschalenkonstruktionen, die über freie Formen generiert werden können. So war die Frage, wie man eine automatisierte Methode der Optimierung programmieren könnte, die die beste Anordnung der Tragelemente in einer Freiform ermittelt. Zusätzlich wurde entschieden, diese Forschung auf die Gitterschalen-Optimierung für eine vordefinierte Freiform zu beschränken. Für diese Entscheidung gibt es zwei Gründe: Erstens zeigt die Erfahrung, dass Architektinnen und Architekten nicht möchten, dass die Form ausschließlich von konstruktiven Bedingungen bestimmt wird. Zweitens wurde im Bereich der Formfindung bereits viel Forschung betrieben, nur ein Teil ist noch weniger erforscht, die geometrische und topologische Optimierung von Konstruktionen für eine vorgegebene Form. Dieser Bereich sollte nun erschlossen werden.

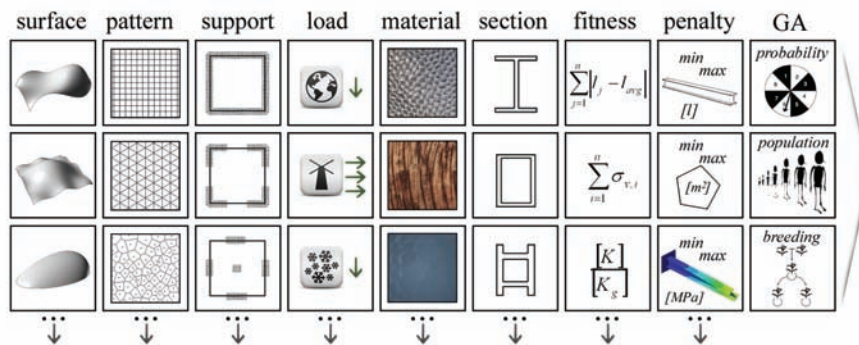
### Methode

Da die Methode automatisiert werden sollte, musste eine Art von Software geschrieben werden. Und da der Auftrag lautete, die statische Wirksamkeit zu verbessern, musste eine FE (Finite Elemente) Methode zur statischen Berechnung in den Prozess integriert werden. Dies wurde durch das Schreiben eines Plug-ins in der Computersprache C++ für eine NURBS-Geometrie-basierte Software namens Rhinoceros 3D erreicht. Das Plug-in korrespondiert mit der kommerziellen Software OASYS GSA. Als grundlegende Optimierungsmethode wurden Genetische Algorithmen (GAs) gewählt, da diese als am besten geeignete Optimierungstechnik für diese Problemstellung gelten. Es handelt sich hierbei um eine stochastische Methode basierend auf einem Evolutionsprinzip, die bei

nicht-linearen und mehrkriteriellen Optimierungsproblemen effektiv ist. Bei jeder Optimierung werden alle Lösungen in spezifischen Files aufgezeichnet, die die Zeichnung aller generierten Gitterschalen möglich macht, ebenso das Extrahieren von Graphen, die den Fortschritt (Konvergenz) des Optimierungsprozesses zeigen.

## Ergebnisse

In dieser Forschung wurde die Software so aufgebaut, dass sie eine umfassende Definition des Problems ermöglicht. Sie ist so angelegt, dass der zukünftige Nutzer verschiedene Parameter definieren kann: Oberfläche, Gitter, Stützen, Lasten, Materialeigenschaften, Querschnitt-Eigenschaften, Optimierungszielvorgabe (Fitnessfunktion), Randbedingungen (Straffunktion) und verschiedene GA Parameter. All dies ist in Bild 0.2 dargestellt. Ziel war es, die Erweiterung der



**Bild 0.2:** Die Eingabemöglichkeiten

Software einfach zu machen, so dass neue Modi der genannten Festsetzungen ohne Probleme importiert werden können. Nachdem die Eingabeparameter gewählt sind, folgt der Optimierungsprozess der Zielvorgabe und nähert sich der optimalen Konstruktionslösung an. Im Rahmen dieser Arbeit wurden Hunderte von Experimenten mit unterschiedlichen Parametern durchgeführt und einige von ihnen wurden dazu genutzt, den Grundgedanken hinter diesem Projekt und die Wirksamkeit der Methode darzustellen.

Die Versuche zeigen, wie durch einfaches Umordnen der Stäbe substantielle Unterschiede in der statischen Wirksamkeit erreicht werden können. Man sieht, wie unterschiedliche Gitterschalenkonstruktionen, die über einigen vordefinierten Formen generiert werden, mit der gleichen Anzahl von Knoten und Stäben, z.B. sehr unterschiedlichen Belastungen und Verformungen ausgesetzt sind; und wie die vorgeschlagene Methode genutzt werden kann, unter den möglichen Lösungen die optimale Struktur zu finden. Das Ziel war nicht, jede mögliche Kombination von Eingabeeinstellungen zu testen, da es praktisch eine unbegrenzte Anzahl von Möglichkeiten gibt, die erdacht und kombiniert werden können. Das ist etwas, was von Projekt zu Projekt in der Realität differiert. Stattdessen lag der

Schwerpunkt auf dem Nachweis, dass diese Methode mit jeglicher Eingabekombination funktioniert und sich der optimalen Lösung entsprechend definierten Kriterien und in festgesetzten Grenzen annähert. Diese Arbeit bietet eine Erklärung dieser Methode an und zeigt, wie man sie konstruiert. Die Versuchsergebnisse dienen als Beweis der Wirksamkeit und Glaubwürdigkeit. Nach dem Nachweis der Funktionsfähigkeit kann die Methode je nach Kreativität des Nutzers und der auftretenden Anforderungen auf unterschiedlichste Weise angewandt werden.

### **Innovation**

Beim Aufbau der Optimierungssoftware wurden viele nützliche Methoden geschaffen und programmiert. Zum Beispiel ist die automatische Generierung eines Gitters (Netzes) über einer Freiform mit Hilfe von Voronoi-Diagrammen einer der entwickelten Routinen dieser Forschung. Ein auf Kraft-Dichte-Methode basierender Relaxationsprozess wurde in C++ geschrieben und dann so erweitert, dass er bei jeder Art von Netz funktioniert (jeder Art von Graph) und so die Relaxation einer Gitterstruktur erlaubt, während sie in einer vorgegebenen Oberfläche bleibt. Zusätzlich, die Einbindung der FEM-Software und verschiedene Methoden für die wirkungsvolle Nutzung von Genetischen Algorithmen (Fitnessfunktion, Straffunktion, Fitness-Skalierung usw.) waren innovativer Teil dieser Forschung und wurden der entwickelten Software beige-steuert.



# 1

## Introduction

### 1.1 Free Form Design

*The straight line belongs to man, the curved line belongs to God.*

*Antonio Gaudi [32]*



#### 1.1.1 Goals and Capabilities

Inspiration in a design process often comes from Nature. Philosophically speaking, it is not possible for it to emerge from any other source. Galileo Galilei makes that point when he says, “*For that which we imagine must be either something already seen or a composite of things and part of things seen at different times*” [15]. Everything we know about the world is derived from the perception the human being has of its surroundings. Every experience gets written into our brains over the 5-sense conductor, leaving us incapable to draw inspiration from something that is beyond our comprehension or our knowledge. If that is so, in order to talk about structural design, we have to start with the *structural characteristics* of the surrounding world. One look around will show us that Nature knows no right angle, nor does it use straight lines. Although there are lots of similarities between species of different life forms, it uses no repetition of basic structural elements for the sake of *production costs*. Actually every structural part of natural systems is unique, however big or small it is.

One glance at the human history and structural design in architecture shows a constant use of straight lines, orthogonal connections and as much repetition of elements as possible. This contrast has a pretty straightforward and reasonable justification. Many systems people made were and still are inspired and influenced by Nature, but the degree of its simplification and abstraction always depended on knowledge, production ability and resources. Because

of the lack of technology enabling the copying of Nature in an exact manner, the main principles were extracted and simplified. In order to use most of the tree trunks, they were cut longitudinal, thus producing straight wooden boards. Iron and steel were cast and rolled into straight beams because it was easier. In the XVII century *Renatus Cartesius*, better known as René Descartes, defines a *Cartesian coordinate system* for space representation, where the rectilinear convention was logical and easy to plan. Natural systems of primary and secondary structures were reproduced with simplified elements like columns, beams and ribs. Due to the uniform effect of gravitation, force slabs were built planar, for people to be able to walk on them. Experience and intuition indicated that longitudinal elements are best exploited when the gravity force acts upon them parallel to their longer axis (causing axial pressure and tension, instead of bending), which made a logical development of vertical walls and columns as basic bearing structural elements.

Everything fitted, and the complexity of building structures was mostly kept in those limits. Monumentality was achieved by varying the number and size of the elements. Although rectilinear system evolved into a rule in architecture, there have always been exceptions throughout history (Figure 1.1), and as technology development progressed they became more frequent. In the last two centuries those deviations from established norms of beauty and proportion, and the world of Euclidean geometry, were referred to by Rafael Moneo as “*forgotten geometries lost to us because of the difficulties of their representation*” [31].



**Figure 1.1:** **left**-Larabanga Mosque (mud), Ghana, 1421.  
**center**-Transfiguration church (wood), Kizhi Island, Russia, 1714.  
**right**-Traditional Chinese roof

The question is if that ubiquitous rectilinear architecture made people perceive free forms as odd. Because we are born into geometrically *controlled* houses and cities, we are used to the idea that rectilinear solutions are optimal precisely due to their simplicity. If the simplicity is something desirable or if “*simplicity is the ultimate*

*sophistication*” as Leonardo Da Vinci claimed [50], then the degree and level of simplification is where the problem lies and in the fact that its definition has constantly changed throughout history. As science goes from centimeters and millimeters over nanometers to unknown depths of elementary particles, in every field of research there is an expansion, branching and division into smaller, simpler elements that build the system. The definition of *simple* suddenly becomes blurry and it has to be carefully weighed up for each problem.

*Make everything as simple as possible,  
but not simpler.*

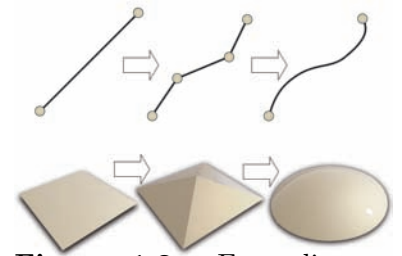
*Albert Einstein [33]*

Following that principle of constant division, straight lines can be divided and combined into polylines. By letting lengths of polyline parts converge toward zero, smooth natural curves can be obtained. Similarly, we can connect plane surfaces, multiply them and let their areas incline toward zero to form a continuous free-form surface (Figure 1.2). We are expanding the range of our possibilities and we are always able to choose the degree of simplification at the cost of production expenses.

### 1.1.2 Why free form?

If everything in Nature is made out of free (irregular) forms, we can only ask ourselves why that is the case. The reason is that all structures are reactions to the forces in Nature acting upon them, or as D’Arcy phrased it, “*The form of an object is a diagram of forces*” [55]. Since those forces are highly complex, irregular and non-uniform, structural systems adjusted themselves to resist those influences in the best possible way. In exploring what *the best possible way* means for Nature, it was realized that throughout the years of evolution it attempts to minimize material and minimize potential energy in its creations, “*for it will profit the individual not to have its nutriment wasted on building up a useless structure*” [10]. When drawing a parallel to the structural design, it is important to mention that Nature’s optimization of structures has a certain dimension of robustness and *safety coefficient*. An interesting thing is the actual *risk assessment* of Nature that can be further explored for structural optimization techniques.

*We should certainly not be so fortunate if our bony skeleton was made fail-safe against frontal impacts by a heavy motorbike. At least all kinds of movement sports would then no longer be an unmixed pleasure. The suffering, pain and depression known by everyone who has had a serious mechanical accident - all that counts for*



**Figure 1.2:** From line to spline, from plane to surface

*nothing in Nature. The suffering of the individual is readily paid as the price for the efficiency of the species. That's why trees and bones fracture!*

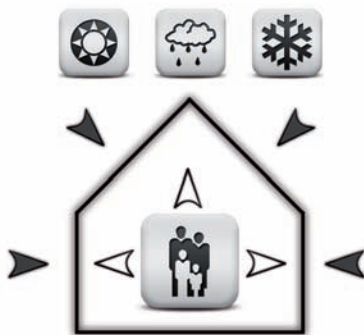
*Claus Mattheck [39]*



Challenges in architecture do have a parallel in Nature, since they are both trying to resolve the problem of optimal design for the protection and survival of living beings. In order to materialize these thoughts in terms of buildings, some of the criteria that should define form and structure have to be considered. First, there are external influences, like gravitation, sun, wind, snow, etc. A building should be intended to protect us from the destructive external influences and enable comprehension and taming of their energy. Second, there are *user requirements*, expressed as different human functions or needs. The interior of the building should be conditioned by complex combination of multiple functions as well as ventilation, air conditioning and energy saving requirements. Third, there is a creativity factor that is multiplied with free forms.

Creativity is a term not known by Nature. Nature's variety is a searching technique to find a better solution according to different criteria, i.e., a solution that will survive. In order to describe this evolutionary process in Nature and draw a parallel to our mental processes that involve the discovery of new ideas, people defined the notion of creativity. But much more important, the notion of Beauty evolved into a science of proportions and design rules, whereas Beauty is also a term not known by Nature. We use it to describe something that gives us a perceptual experience of pleasure, hence something that is *optimal* for some specific set of conditions (objective or subjective).

Combining external and internal, non-uniform influences, we have to agree that geometrical simplification principally doesn't lead to the minimal energy and minimal material solution. People inclined toward those spatial answers over the years for several reasons. An obvious one were the productions costs, followed by an extreme simplification of functional needs. Namely, gravity was one of the most important external forces and therefore considered as the dominant structural factor. Simplified functional organization fitted into the rectilinear system. That is how simple boxes were formed, whereas many other external or internal influences had to fit into those boundaries, defined by only a few factors which were taken into consideration. One of the ways to express creativity was the use of ornament, a pure decoration, made to imitate Nature's complexity and achieve Beauty, thereby having no functional justification. There was still a lot of room for architects to express their creativity but the structural system was often there to limit the imagination.



**Figure 1.3:** Complex influences require complex form

### 1.1.3 Free Form Today

*The cylinder, pyramid, cube, prism and sphere were not only the essential forms of the Egyptian, Greek and Roman architecture, as dryly observed by Le Corbusier, but were also universal geometric primitives of the digital solid modeling software of the late twentieth century. They are no longer seen, however, as some kind of unique isolated archetype, but as special cases of quadratic parametric surfaces.*

*Branko Kolarevic [31]*

In the 21<sup>st</sup> century, gravity is not as difficult an adversary as it used to be. Other environmental aspects started influencing the design of a modern building. The wind's power was embraced with smooth, aerodynamically designed, surfaces and the use of CFD (Computational Fluid Dynamics) analysis was used as a tool to design resistant and stable buildings. It was realized that energy gains could be highly influenced by the shape and organization of structures and the choice of materials. The rectilinear organization was often reconsidered and fluent and dynamic spaces became popular. With CNC machines it was possible to mill, bend, *print* or cast different materials to give almost any shape and enable manufacturing process that recognizes “no substantial difference any more in cost of producing 1000 unique objects or 1000 identical ones” [31]. So we are ready to climb one step up toward the natural structural systems. The technology exists and the principles that will enable us to use it in the best way possible have to be established.

## 1.2 Grid Shell as a Structural System

### 1.2.1 Development of Grid Shells

The categorization of structural systems can be made in numerous ways using different approaches, based on various criteria, like the shape, position, material, etc.[12]. For the sake of explanation of grid shell genesis and its advantages, a simple geometrical approach will be used. Considering the type of the force flow, i.e., the shape of the elements, systems in the history of structural design can be very roughly divided into linear and surface ones. If one dimension is much bigger than other two (Figure 1.4), the element can be considered as linear (columns, beams, etc.). If the element has two dimensions relatively proportional and much bigger than the third one it is a surface element (Figure 1.5), allowing force to disperse in two dimensions throughout the element (walls, slabs, etc.).

According to their orientation in space, elements could be vertical, like columns and walls, or horizontal, like beams, ribs and



**Figure 1.4:** Linear system

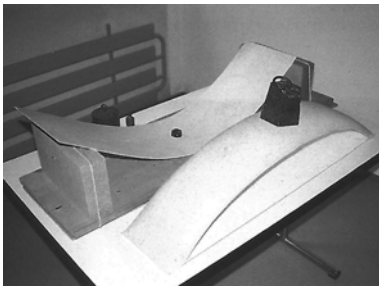


**Figure 1.5:** Surface system



slabs. Elements under slope were used mainly for roof structures, thus developing different names but still having the same principle for the distribution of load forces over one or two dimensions. A different combination of both systems is seen throughout history, changing as the technology and the discovery of new materials did. From the columns in Luxor, over Le Corbusier's *open plan* toward emergence of skyscrapers at the beginning of 20<sup>th</sup> century, the linear system progression is quite obvious. The parallel development and integration of surface elements also experienced change in material, but kept the principle.

However, straight and flat elements were not as efficient as curved ones. Heinz Isler (1926-2009), a Swiss Engineer, used a simple experiment (depicted in Figure 1.6) to show that a curved plastic element can resist more than 30 times greater load than the same element when it is flat [5]. Throughout the history those *arched* structures were made from masonry or some form of unreinforced concrete, materials strong in compression but relatively weak in tension [5]. A very nice example is the Persian monument *Tāq-i Kīsra*, with a 37 meters high central arch-dome, built in 540 AD (shown in Figure 1.7). The arch spans 26m and it was made out of bricks without centring (the supporting structure) and with the use of a quick drying cement [48]. A better solution were cast materials, and in Ancient Rome there were some versions of concrete: a mixture of lime mortar, volcanic sand, water and small stones (*cementa*) [13]. To this day, the largest unreinforced solid concrete dome in the world is the Roman Pantheon shown in Figure 1.7. It spans 43.3m with a thickness that represents only 2.8% of its diameter. Brian Cotterell and Johan Kamminga in *Mechanics of Pre-industrial Technology*, have examined this and some other domes (like Augustan Temple of Mercury and Santa Sophia) and concluded that a hemispherical dome could guaranty stability if it was thicker than 2.1% of its radius [6].



**Figure 1.6:** Model demonstrating the efficiency of a double-curved shell made from thin plastic compared to a similar plastic sheet acting as a wide flat beam over the same span. The load on the shell is 30 times the load on the flat sheet [5]



**Figure 1.7:** left-*Tāq-i Kīsra*, Persian monument, 540 AD  
right-Pantheon, Rome, 126 AD

Curved structures - shells - developed further with the introduction of reinforced concrete in the 19<sup>th</sup> century, material strong in both compression and tension. Due to the fact that it was cast, it was a homogeneous material with an even distribution of force,

thus enabling the transformation of flat slabs into shell structures. Distribution of loading pressure over a continual surface was an idea that managed to span up to  $50m$  with only  $10cm$  thick concrete shells, thus having the span/thickness ratio in the region of 500 to 1 [5]. Cast shell structures were quite an advancement because the principle of shaping the beam to follow momentum lines, thus creating arches, could now be implemented to surface elements (Figure 1.8). Following the force diagram, concrete shells were almost without moment, hence distributing the forces axially through the shell. Immense tenuity was then achieved due to smart shape design and the high pressure resistance of concrete. People like E. Torroja(1899-1961), F. Candela (1910-1997), H. Isler(1926-2009), P.L.Nervi(1891-1979), O. Arup(1895-1988) or N. Esquillan(1902-198) are responsible for beautiful shell structures around the world (some of them shown in Figure 1.9), demonstrating their possibilities. Still, the method had big restrictions. Due



**Figure 1.8:** Shell, distributing forces in all directions



**Figure 1.9:** left-Valencia Oceanographic, F.Candela, 2002.  
center-Sidney Opera House, O.Arup, 1973.  
right-CNIT Paris, N.Esquillan, 1958.

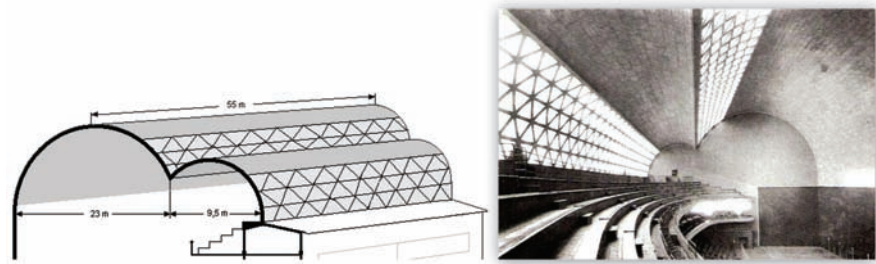
to the small thickness of shells, they carry the loads mainly through membrane forces and aren't able to resist large *out-of-plane* bending forces. One way to design a concrete shell was to generate it as a specific part of a sphere, the part that experiences only pressure. The delimitation of the two parts of the sphere, where pressure becomes tension, was experimentally established (Figure 1.10). There were other ways of finding the appropriate form, like the use of physical models that Heinz Isler and Frei Otto performed, that ensured that the bending moments are minimal. Anyway, the shapes were limited and the problems of cost were big, because it was usual that formwork and falsework represent around 50% of the total cost of the shell, even when the shuttering was reused several times [5].

The development of reinforced concrete shell structures ran parallel to that of grid shells. The basic idea was to combine linear systems with shell structures by tessellating the surface and dividing it into fields. The result was a space structure of connected linear elements. This way materials as steel or wood could have been used and, with the clever surface division and node design,



**Figure 1.10:** Pressure and tension in sphere shaped shell

take over pressure and tension forces. It was basically a mix between a grid structure and a continuum shell, an evolution of truss structures into spatially curved grids. Pier Luigi Nervi was one of the engineers that successfully combined *grid* with *shell*, but maybe the best example that demonstrates their fusion is Torroja's *Fronton Recoletos*, built in Madrid in 1935 (Figure 1.11), where the roof is made as a combination of a concrete shell and a triangular grid shell.



**Figure 1.11:** Fronton Recoletos, Eduardo Torroja, Madrid, Spain, 1935.

Materials more appropriate for grid shells, due to the prefabrication possibilities, were iron and steel. Among the first successful attempts in making a steel grid shell were the ones of Johann Schwedler and Vladimir Shukhov. Schwedler developed a *Schwedler cupola*, that was able to span distances of 25-45m [30]. Its first application was as a steel roof for the gas holder of the Imperial Continental Gas Association in 1863 (Figure 1.12). Another pioneer of grid shell structures, as they are recognized today, was a Russian engineer Vladimir Shukhov. He built a double curved quadrangular roof structures covering exhibit pavilions at the All-Russia industrial and art exhibition in 1897 (Figure 1.12). Others slowly followed them and developed it into an art.



**Figure 1.12:** left-Schwedler cupola, J. W. Schwedler, Berlin, 1863  
**right-** Exhibit pavillion, V. Shukhov, Vyksa, Russia, 1897.



## Buckminster Fuller

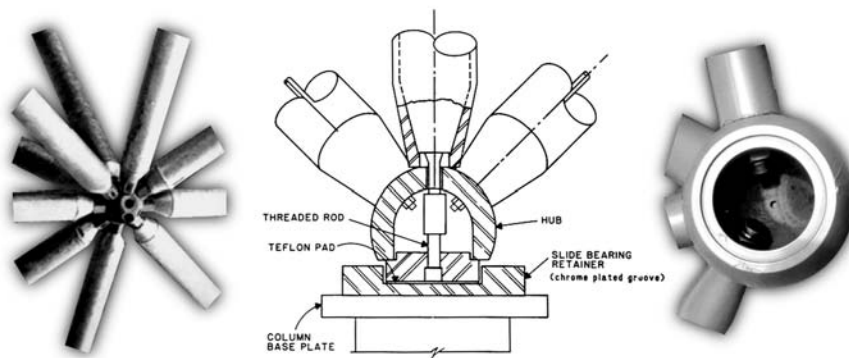
Humanitarian and energy-saving approaches lead one of the greatest minds of the 20<sup>th</sup> century, Buckminster Fuller, to popularize *geodesic domes*. Since the sphere is a geometrical shape that encloses the greatest volume with least surface, he advocated its use in form of triangulated grid shell structures. They enabled great spans with single layer lightweight grid and titillated the imagination of architects and visionaries like himself.

*Geodesic structures opened up the ability of humans to build unlimited-diameter clear-span spherical structures. By 1958 I had built a clear-span geodesic hemispherical dome of 384-foot diameter. Since then they have gone to 700 feet in diameter, and they will keep on growing in clear-span size at an ever faster rate until we enclose whole cities.*

*Buckminster Fuller [14]*

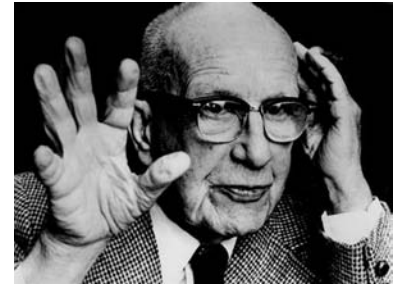
He showed that shape and form, as well as geometrical disposition of the constituting elements, play a great role in structural design and that high level of prefabrication can be used to create very light and stable structures. Although advocating simple and mainly familiar things, Fuller combined them and strengthened the idea of lightweight design thus inspiring many others to come.

## Max Mengerhausen



**Figure 1.15:** MERO system, prefabricated elements

What was welcomed in Fuller's structures was that the construction of geometrically *primitive* shapes (like sphere), enabled effective prefabrication. A very high degree of repetition of members and joints made the production easy and cheap. That was also the idea of Dr.-Ing. Max Mengerhausen, who developed a so-called MERO system (*ME*ngerhausen *RO*hrbauweise, or Mengerhausen's tubular structures). With standardized joints,



**Figure 1.13:** Buckminster Fuller



**Figure 1.14:** The Biosphere - 76m in diameter, B. Fuller, Montreal, Canada, 1967.

MERO turned grid and truss structures into one of the cheapest and most effective systems for spanning large distances, thus popularizing them immensely. Soon, a wide variety of solutions for truss element prefabrication appeared and conquered the market.

### Frei Otto

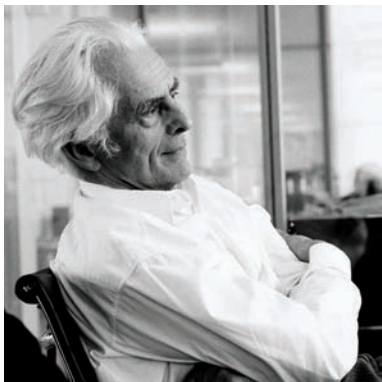


Figure 1.16: Frei Otto

Frei Otto is a German architect who founded the well known Institute for Lightweight Structures at the University of Stuttgart in 1964 and led it until his retirement. Like Fuller, he turned to Nature in quest for answers. His soap film models used for form finding are famous as well as the experimental approach in the field of hanging (cable-net) structures. He used the Nature’s minimal surface principle to design cable-net structures for the West German Pavilion at the Montreal Expo in 1967 and the roof of the 1972 Munich Olympic Arena. Double curvature was once again employed but this time with tension in the structural members.

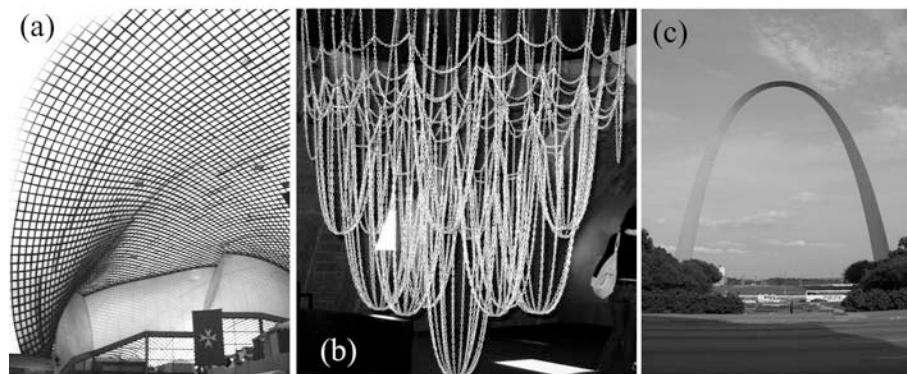


Figure 1.17: left-Multihalle, F.Otto, Mannheim, Germany, 1975. center-Catenary model at Casa Mila, A. Gaudi, Barcelona, Spain, 1910. right-Gateway Arc, E. Saarinen, St.Louis, Missouri, USA, 1965.

In year 1975, architects Carlfried Mutschler and Joachim Langner together with Frei Otto built a double-curved wooden shell structure known as the *Multihalle*, for the *Bundesgartenschau* (National garden exhibition) 1975 in Mannheim (Figure 1.17 (a)). The shape of the object was found using hanging models. Inversion of hanging cloth or membrane is “for three-dimensional problems what the catenary line is for two-dimensional arches” [5]. The principle was not new, even the example of *Tāq-i Kisrā* shows the application of the same catenary method [20]. One century before Otto, following less sophisticated attempts of Heinrich Huebsch and Giovanni Poleni, Antonio Gaudi’s experiments intended to show that the optimal shape of a structure under pressure is obtained by inverting of a suspended cable net (Figure 1.17 (b)). Thus Gaudi popularized the use of catenaries, which always produced very effective shapes (if we forget multi-objective optimization and concentrate on gravi-

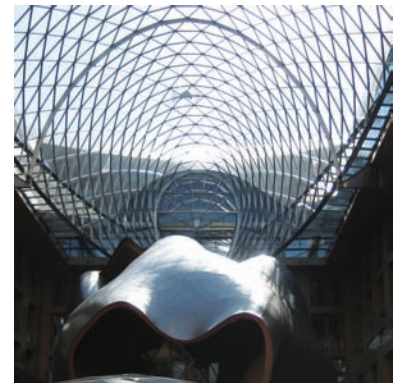
tation as the basic and main influence). Figure 1.17 (c) shows Eero Saarinen's Gateway Arc in St. Louis, Missouri, the form of which is determined by hyperbolic cosine function. Since its thickness changes, it was not a pure catenary but a form of a *weighted catenary* generated by Dr. Hannskarl Bandel (1925-1933), a German-American structural engineer [7].

As mentioned, the method was used extensively by Frei Otto at his institute, and *Multihalle* Mannheim is an actual product of such studies. Built in 1975 it expressed a new way of thinking and presented new possibilities in free form structural design. It showed that a clever shape optimization can lead to solutions with greater span and less material, a true lightweight architecture.

### 1.2.2 FEM and CNC

After the Second World War, the FEM (Finite Element Method) of static analysis started to develop and it brought about the revolution in all spheres of engineering, enabling experts to calculate the force distribution of extremely complicated systems. The main breakthrough in this field happened from middle to late 1950s through the work of John Argyris, who was appointed a professor at the Technical University of Stuttgart (today University of Stuttgart) and director of the Institute for Statics and Dynamics of Aerospace Structures in 1953 [29], and his cooperation with Ray W. Clough at Berkley. FEM is a numerical method for finding approximate solutions of partial differential equations and integral equations, hence for large structures, calculations were not possible without the computer. Throughout years FEM evolved into a software that today enables visualization of stresses and displacements in structure and wide range of static and dynamic analysis. Since the design and static analysis of very complex structures were possible, the question of their production arose.

At the end of the 20<sup>th</sup> century, CNC (Computer Numerical Controlled) machines were introduced to the building industry and started to be used widely. The first free form grid shells with controlled and precise design of their members started to emerge. A perfect example is the roof of DZ Bank in Berlin finished in 2001, designed by Frank Gehry (Figure 1.18). Schlaich Bergermann & Partner office did the structural design of the roof using a triangulated mesh to divide the free form surface, resulting in unique stainless steel rods and unique joints. Limits were pushed again and new visions appeared. If there are no more restrictions by the prefabrication of members, and all of them can be unique, it means that the form of grid shells is finally free and that possibilities multiplied enormously.



**Figure 1.18:** DZ Bank, F. Gehry and SBP, Berlin, Germany, 2001

### 1.2.3 Application of Grid Shells

Always when new tools become available the question of their proper usage arises. New technology broadens our perspective and enlarges our responsibility at the same time. Today, the possibilities we have are mainly used for a *shape driven design*, using new methods for sculptural expressiveness. The work of architects like F. Gehry, M.Fuksas or Z.Hadid is a great illustration of the principle. For structural designers that can be seen as an advantage, since the challenges are bigger and the need for creative solutions inspires new ideas.

Grid shells therefore distinguish themselves as the most general structural system, applicable to different forms. Although used mainly for roofs today, it is not limited to them and it is spreading slowly into all parts of a building. The modernistic skeleton is a system that strongly divides the structure from the facade, something that is a very rough simplification of Nature's primary and secondary elements. On our way to a more effective approach, a reasonable direction to take is the unification of structure and *skin*, thus enabling optimal use of space and much broader design possibilities.

*The building envelope is increasingly being explored for its potential to reunify the skin and structure in opposition to the binary logics of the Modernistic thinking. The structure becomes embedded or subsumed into the skin, as in semimonocoque and monocoque structures...The principal idea is to conflate the structure and the skin into one element.*

*Branko Kolarevic [31]*



**Figure 1.19:** *Skin* - structure and facade of the 18 Septemberplein in Eindhoven, Netherlands, designed by M.Fuksas, roof structure by Knippers Helbig, finished in 2008.

Considering the merger of facade and structure, and the possibility of manufacturing and assembling different unique parts, we entered the 21<sup>st</sup> century with much more freedom. In the following chapter it will be explained how that freedom is materialized and what its advantages and obstacles are.

## 2

# State of the Art

In the history of free form shaped architecture, the main accent has been on the quest for the optimal form. The mentioned work of Gaudi, Fuller and Otto demonstrates that. Different methods, physical and mathematical, were used to find the shape that would minimize stresses and enable greater spans. This enormous research activity is recognized under the name of *form finding* and the basic idea has led to the development of new techniques, like the Force-Density method that is used in research and will be addressed in Section 3.2.

The research presented is confined to grid shells designed over a predefined surface, which means that it presupposes an already determined shape and offers a structural solution based on it. This approach of structural design, without alteration of shape, was chosen for two main reasons.

The first reason is based on experience, which showed that architects today embrace the possibilities offered to them by new materials and new production methods and therefore don't want to be constrained to statically optimal forms. If they were, their design freedom would be drastically diminished. It is important to remember that the bearing structure is only a part of a building and does not, or should not, define the form alone. Function must be combined with it to find an optimal design solution. Since the function differs from object to object, depending on the needs or geographical location, and the creativity of the designer comes into play, we gain unique buildings, the shape of which can be very different from the statically optimal ones. This way of thinking could have hardly been justified in the past, due to the high costs, which leads us to the second reason for the research approach presented.

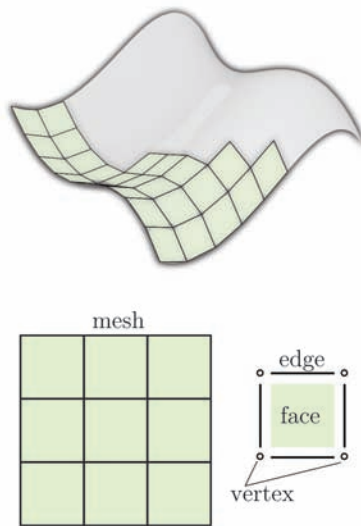
The second reason relies on the aforementioned new techniques of analysis and production. Grid shells can provide structural solutions for extremely imaginative shapes. Although it must be admitted that a statically optimal shape will generally cost less, nowadays the difference in costs is small enough to allow us to choose greater freedom in design. Today we are witnesses of a shape driven ar-

chitecture, buildings as sculptures that demand structural solutions that don't change their shape. Structural constraints slowly retreat and leave more space for creativity and functional optimization.

## 2.1 Grid Shell Design Methods

Taking a defined shape, sculpted in space, there are two basic approaches to structural design. On the one hand, the geometry of the grid shell has to be optimized. It has to be optically acceptable, and it has to satisfy some of the prefabrication conditions. Those conditions can refer, for example, to the size of the glass tiles that cover some roof structure, or the maximal and minimal acceptable length of a structural member. Whatever the conditions are, their fulfillment can be recognized as the *Geometrical approach*.

On the other hand, static analysis can lead to statical optimization of some structural solution. Whether the grid structure represents a clearly visible roof or a hidden secondary structure of some nontransparent facade, it can be optimized according to the static conditions and requirements. This type of approach will be referred to as the *Statical approach*.



**Figure 2.1:** From Surface to Polygonal Mesh

### 2.1.1 Geometrical Approach

To create the grid from a surface we *tessellate* it. The basic idea of tessellation is to create a *polygon mesh* by dividing the surface into discrete fields (Figure 2.1). A polygon mesh is a specific structure in computer graphics. According to its definition it consists of *faces* bounded by *edges*, that become structural members of the grid shell, and *vertices*, that represent joints of the members. The areas bounded by structural members will therefore, from now on, be referred to as *faces*, or later on as *cells*. In structural design so far, size and shape of the faces were highly restricted due to the material properties and manufacturing possibilities. As grid shells were (and mainly still are) used for roof structures, the covering material properties are an important factor when considering the size of the faces and edges, i.e., tile area and length of the structural members. Since glass is mainly used for its transparency, its manufacturing and static characteristics limited the mesh edge to  $\sim 3m$ . Experience additionally showed that, when building steel grid shells, the best material usage (considering the glass limits) was with  $2m-3m$  long structural members for triangular grids, and  $1.5m-2m$  for quadrangular grids. Considering geometry and topology of the mesh, different patterns (different mesh face polygons) were thereby used for the tessellation. One of the ubiquitous approaches involves triangulation of surfaces, thus creating triangulated meshes, i.e., triangulated grid shells.



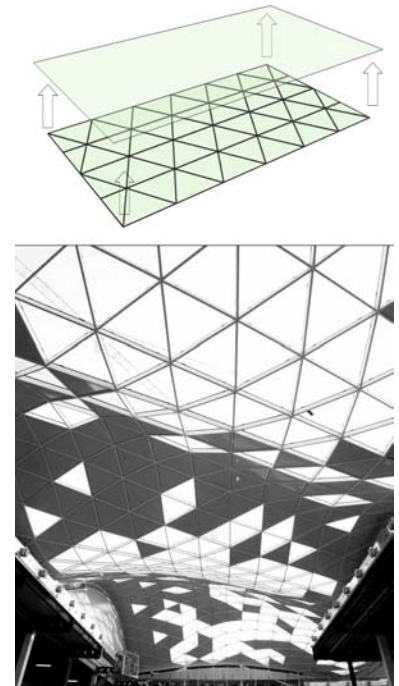
## Triangular Grid

An additional condition that the covering system imposed on the shape of faces is planarity. Namely, the production costs of double curved glass (or any other material) panels were always simply too high in comparison to single curved or flat elements. Since every triangle is by definition planar, surfaces were mainly divided into triangles, creating triangular meshes. The sides of each triangle were designed to fit the 2-3m length and numerous projects around the world emerged exemplifying the application of this system.

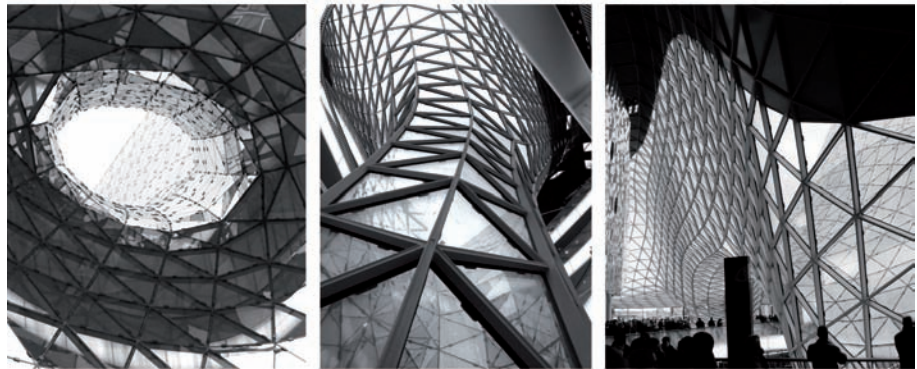
**Projection** There are several ways to design a triangular grid shell. In the project for the Westfield mall in London, structural design of the grid shell roof, over  $17.000m^2$  of free form surface, was done by the Knippers Helbig office in Stuttgart [22]. The shape was designed together with the architect as a double curved surface and the task was to offer a structural solution bounded to that surface. Since the curvature of the surface was moderate, the simple projection method was applied (Figure 2.2). A planar triangulated polygon mesh was projected vertically to the surface and therefore made production easier, since the angles between the members in every node were  $60^\circ$  in the horizontal ( $XY$ ) plane. In that way, only the slope of each of the 8500 members differed and only one angle had to be adjusted. At the time it made production costs lower but still ended up as a structure with unique members and nodes.

**Relaxation** From the simple projection method we move to more advanced techniques that were used in the project of MyZiel mall in Frankfurt (Figure 2.3). It was designed by Massimiliano Fuksas office in Rome and Knippers Helbig office did the structural design of the  $13.000m^2$  free form glass roof [22]. Following the Buckminster Fuller's sphere division principle, the surface was divided into big triangles. This is done in order to establish the basic line directions. Each of those triangles was then subdivided into smaller ones to gain a basic triangulated mesh. At the end, *mesh relaxation* techniques were applied in order to get smooth and optically acceptable structure.

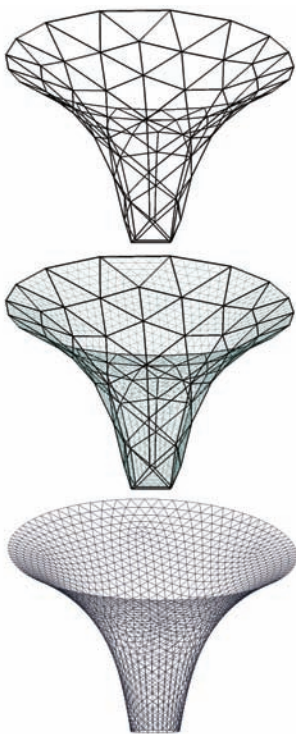
The matter of *relaxation* requires a short digression. Namely, in the year 1971 Linkwitz and Schek developed a new formulation for finding the equilibrium of forces in cable network structures [38], in the generation of the 1972 Munich Olympic Games stadium roof design. The principle was named Force-Density method (*Kraft-Dichte Methode*) and it will be explained in more detail in Section 3.3.1, together with the principles of its application in the grid shell design field nowadays. For now it is important to notice the effect it produces.



**Figure 2.2:** Westfield Shopping Center, London White City, completed in 2007. Architects: Buchan Group International, Structural Engineering:Knippers Helbig



**Figure 2.3:** MyZeil, Frankfurt, Germany, completed in 2008. Architect:M. Fuksas, Structural Engineering:Knippers Helbig

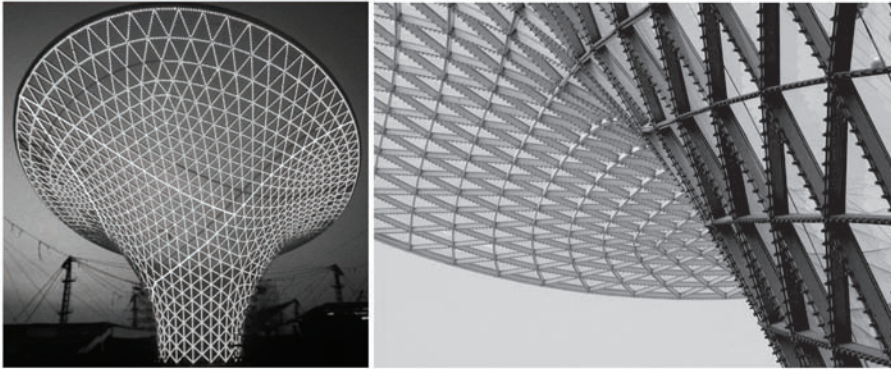


**Figure 2.4:** Design process of the EXPO Shanghai Sun Valleys, from large triangles (up), their division (middle) to relaxation of the grid (down)

**Controlled relaxation** The *relaxation* techniques were applied at a higher level of complexity for another structural design of Knippers Helbig office. It was the EXPO Axis, a 1km long membrane roof in Shanghai and the focus is on so called *Sun Valleys* - 41.5m high funnel-like structures made as triangulated single layer grid shells [51]. As with the MyZiel roof, the structure was divided into large triangles only to be subdivided further into smaller ones. The *relaxation* method (explained in Section 3.3.1) was then applied to smoothen the grid with some additional adjustments. This three-step process is depicted in Figure 2.4. Since the huge membrane roof had to be connected to each of the six different Sun Valleys, the areas where that occurred should have had a greater density in order to take over the immense tension forces. This was managed with the control of the *tension factor* of the members around the critical points, therefore *pulling* the mesh closer to them and creating a greater density in those areas, thus enabling better static response. This is explained in Section 3.3.2 and depicted in Figure 3.16, and the use of *tension factors* is explained thoroughly in Section 3.3.1 together with the relaxation method. For now it is important to notice that the relaxation technique can be controlled and finely tuned to suit a number of geometrical or statical conditions.

Experience with various aforementioned grid shells was an actual inspiration, the initial spark to investigate further possibilities of their optimization. It triggered the need to search for methods beyond the trial and error techniques that were used, and to see how the whole process can be automated. Before the innovative methods are explained, the subject of grid shells with different polygonal structures will be briefly addressed.





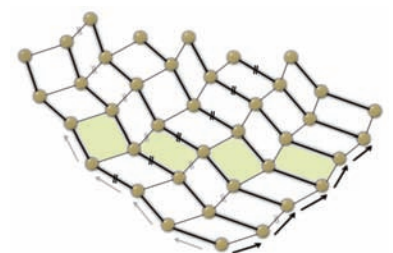
**Figure 2.5:** One of six *Sun Valleys* for EXPO 2010, Shanghai, China SBA Architects and Knippers Helbig Advanced Engineering [51]

### Quadrilateral Grid

Until now, the alternatives for triangular meshes were mainly the quadrilateral ones. They had the advantage of being lighter and less complicated to build, due to the simpler joints. In the triangulated meshes, joints had to be designed to transfer the forces of 6 different members connected in one point. Quadrilateral grid structures have joints with only 4 members, but this simplification led to new problems. The first one was stability, due to the transfer from highly rigid triangle to *movable* rectangles, the stability of the whole structure came into question. It was realized that the buckling of complete system presents a very big danger in a lightweight grid shell design. As structures became lighter, due to the smart use of FEM analysis software, buckling, i.e., stability of the system moved up on the ladder and became the primary condition. As it can be seen in numerous buildings with quadrilateral grid shells, there are steel cables (bracings) spanned over the diagonals of the quadrilateral to basically transform them into a triangular grid. However, that is not always the case, and one of the fine examples is the Cabot Circus shopping mall roof in Bristol, shown in Figure 2.6 [25]. Here the quadrilateral grid shell is constructed without the bracings, showing that they are not always a necessity.

Another problem with quadrilateral grid shells was a geometrical one. Used for roof structures, covered with glass, planarity of mesh faces had to be ensured. Planar quad meshes can be achieved in several ways. One way is by translating a polygon against another polygon, resulting in quadrilaterals with two pairs of parallel sides, as shown in Figure 2.7. The parallelism of two pairs of opposite sides ensures the planarity of the quadrangles. Rotational surfaces, such as cone or hyperboloid can also be represented as planar quad meshes, known as rotational PQ mesh [17].

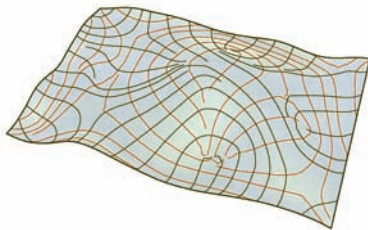
Other, geometrically more complicated methods, involve *conjugate curve networks* and subdivision methods [17, 49]. The more



**Figure 2.7:** Quadrilateral mesh, where translation ensures planarity of faces



**Figure 2.6:** Cabot Circus, Bristol, England, Schlaich Bergerman & Partners, 2008. [25]



**Figure 2.8:** Example of a free form surface whose principal curvature lines are not suitable as the basis for the layout of a planar quad mesh [17]



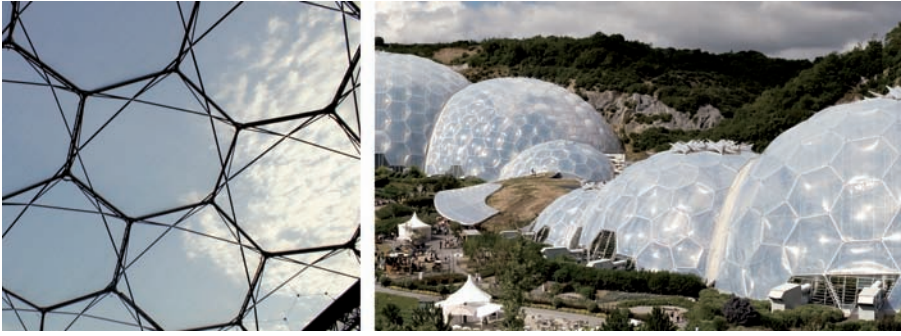
**Figure 2.9:** Buckyball

complicated the surface is, the less possible planar quad solutions there are. The limits are therefore substantial, since there are usually only few possible planar quadrilateral solutions for a free form surface, that are often not statically and optically acceptable. Sometimes the principle curvature lines can hardly be translated into a sensible grid, as in the example shown in Figure 2.8. Those limits can make us ask, why we are focusing on solutions that meet the demands of old and rigid materials instead of trying to overcome those restrictions. Some visionary thinking has to be done to prepare us for the coming of new materials with new performance ranges, where the curvature of faces will not be a problem.

### N-gon Grids

There are sphere tessellation methods that can be described as *cutting* the sphere with planes. Transferring them to free form surfaces, they show the generation of polygon meshes mainly composed of hexagons and pentagons, the shape of which depends on the concavity and convexity of the surface [17]. The cutting method was developed from the planarity condition and the grid structure depends heavily on the free form shape, thus making the system not applicable to surfaces with large curvature changes. In Figure 2.9, an example of a *buckyball*, named after Buckminster Fuller, is shown. This is one of numerous possible patterns of dividing spheres or planes (which are then transferable to open surfaces). Some of those regular n-gon grid structures will be addressed in Section 5.2 and it will be shown how they can be used for free form surface tessellation. A nice example of hexagonal structure is also the *Eden Project* in St Blazey (UK), designed by Nicholas Grimshaw, shown in Figure 2.10.

Along with new solutions, the limit of planarity will probably be transcended by new materials and CNC production of double curved elements, but until then, the planarity condition remains



**Figure 2.10:** Eden Project, St Blazey, Cornwall, UK, Nicholas Grimshaw, 2001. [52]

one of the important factors in grid shell design, often limiting the statical optimization. The difficulties concerning the stability of structures, i.e., the problem of mobility of rectangles, pentagons, hexagons or any kind of  $n$ -gons, can be resolved in future with more intelligent joint design that will be able to take over the forces in a way that insures the stability of the whole structure. Several imaginable scenarios are possible when the development of new materials is considered, especially when their elastic limit is investigated. The condition of rigidity of structures can then be reconsidered and the structures could maybe *breathe* and move. Moving in that direction, the clever combination of materials and their geometry has to be extensively explored.

### 2.1.2 Statical Approach

So far, the geometrical approach for grid shell design has been described. It was pointed out above that certain adjustments of the grid density had to be made for the EXPO Axis project, in order to optimize the structure statically. That is a small example of how geometry has to be altered to fit statical conditions. Theoretically, the design process can be solely guided by the statical behavior of the structure, but in practise it is always a combination of optical and statical conditions, as well as manufacturing constraints. However to find an acceptable solution, combined cross-sectional, geometrical and topological optimization is usually performed. In the field of free form structural design today, optimizations have one thing in common - the use of the trial and error approach.

Grid shells are optimized optically and after a smooth and acceptable mesh is obtained the cross-sectional optimization is performed in a slow, non-automated iterative process. There is a lot of research done in geometrical and topological optimization of truss structures [56, 2, 57, 26, 23] and aforementioned cross-sectional optimization of grid shells [2, 34, 34]. However, when it comes to free form design, the research mostly turns to form finding and geomet-

rical optimization of existing triangular or quadrilateral structures by the small movement of their nodes and orientation of their members [56].

When it comes to comprehensive design of grid shells over a predefined free form surface to fit the static, optical and manufacturing conditions in the best possible way, people often rely on experience and intuition, ending up with a trial and error method that is, due to the strict deadlines, limited to a small number of attempts. The goal of the statical approach is to find a structure that will use minimal amount of material and satisfy static conditions, like permissible stress or stability. Respectively, to answer what structure within specific boundaries (like a predefined surface in this case) requires minimum material, i.e., minimum cost and maximum performance, an appropriate method has to be developed that will enable thorough examination of the *search space*.

The rigidity of the joints in a grid structure, and their beam element composition, makes the design much more complicated than truss structure design (where the elements are pin-joined, i.e., take over only axial forces). The design of the grid shell beam element has to account for three moments and three axial forces. Therefore it requires an engineer to choose a combination of cross-section size and shape in a way that insures that the moment of inertia and cross-sectional surface can resist all the forces as well as satisfy the buckling conditions. Considering the entire structure with possibly several thousand members, the interdependence between them gets extremely complicated. The number of possible combinations and variables, i.e., the gigantic *search space*, is exactly the reason why the geometrical and topological optimization of these structures presents quite a challenge. In the following chapters it will be shown how the static analysis approach can be applied in a free form grid shell design.

## 2.2 Structural Optimization

Regardless of the approach, grid shell design, or any kind of design, is a process of optimization. Every decision making begins with a creation and selection of some solutions that are picked and altered according to some set of objectives. In structural engineering it basically comes down to optical and statical conditions that, as it is already emphasized, should always be considered together. Optimization in structural engineering should always be multi-objective and restrained, and the challenge is to incorporate as many objectives as possible, and examine the search space in as much detail as possible to find a set of acceptable solutions.

### 2.2.1 Basic Terms

In order to simplify the whole process, some of the basic terms of structural optimization will be explained. In every structural optimization we have to define what we can change in order to find the optimal solution. This leads to the definition of a set of *design variables*. The next step is to define the problem, a goal, that should be pursued by changing those variables. That goal comes in the form of *objective functions*. In structural engineering, a number of restrictions have to be considered. Conditions like material properties, production costs and possibilities, etc. have to be kept in mind, and an optimal solution has to meet these conditions. They are defined as a set of *constraints* that limit the search space to *feasible* solutions. After these basic definitions, a method of optimization has to be chosen, or developed, that will give us the best solution in the shortest time.

#### Degree of freedom - design variables

The structure's freedom to transform is always expressed over design variables, often denoted by a vector  $\mathbf{x} = (x_1, x_2, x_3 \dots, x_n)$ . In structural design the choice of possible variables can be narrowed down to: material properties, cross-sections of structural members, geometry and topology of the structure. These parameters can be defined as continuous or discontinuous, regarding the values they can have. If the parameters are continuous it is assumed they can take any value in a specific range, while discontinuous variables can only have isolated values. Although optimization processes are usually simplified by making all variables continuous, a good example of a discontinuous one is the cross-section, as we are often able to choose only the ones offered by the manufacturer. Choosing the appropriate variables is the first step toward a successful optimization process and will be addressed in Section 4.2.

#### Problem formulation - objective function

After the *participants in the game* are selected, the goal has to be chosen, an objective expressed as a function  $f(x)$  or set of functions  $f(x) = [f_1(x), f_2(x), \dots, f_m(x)]$ . They are called *objective functions*, and depending on their number, an optimization can be single-objective or multi-objective. In the research presented, both of these optimization types will be addressed in more detail in Section 4.3. The objective function is usually a simple mathematical definition of a term we want to minimize or maximize. In structural engineering it mainly describes a minimization problem.



### Constraints

In the optimization process, a set of constraints is taken into account to define the feasibility of the solution. It can be visualized as bounding the infinite search space, thus creating a finite search space where the optimal solution has to be found. If we generalize structural design problems as minimizations of functions, the general form of the function with constraints would be:

Minimize:

$$f(x), x = (x_1, x_2, x_3, \dots, x_n) \quad (2.1)$$

under constraints:

$$g_i(x) \geq 0 \quad , \quad i = 1, \dots, K \quad (2.2)$$

$$h_j(x) = 0 \quad , \quad j = 1, \dots, P \quad (2.3)$$

Here  $g_i$  and  $h_j$  are inequality and equality constraints and they represent the limits of our design solutions, i.e., the borders of our search space. There can be more than one constraint (up to  $K$  or  $P$ ) that eventually help in the distinction between feasible and infeasible solutions. The feasibility in structural design is usually rigidly strict. Restrictions can refer to yield stress of a material, or production capabilities, for example. The application of constraints in grid shell optimization is explained in much more detail in Section 4.4.

### 2.2.2 Optimization Types

Following the design variable selection, four main directions in structural optimization can be distinguished:

1. Material Optimization
2. Cross-sectional Optimization
3. Geometry optimization
4. Topology optimization

The research described in the following chapters does not address material and cross-sectional optimization. One of the reasons is that those problems are relatively easy to describe in the form of mathematical functions, making them suitable for calculus based methods. Since lots of research is already done in that field, the idea was to turn to less explored geometrical and topological optimization of grid shell structures. The number of possible geometries and topologies is enormous and therefore the search space is much bigger than in the first two mentioned optimizations. An additional problem is the large number of variables that have to be considered, leading to multi-objective and non-linear optimization. Such problems are mainly impossible to represent in the form

of continuous differential functions and therefore different methods for their solution have to be considered. In the following sections, possible optimization techniques will be presented with the brief explanation of their advantages and disadvantages.

### 2.2.3 Calculus Based Optimization Techniques

Calculus based methods use calculus, as the name implies, to solve mathematically defined objective functions. If the problem is represented as a nonlinear function of single or multiple design variables, the solution is to find extremes of that function. The basic differentiation can be made between constrained and unconstrained methods. The methods will simply be mentioned in order to show their diversity and basic approach.

**Unconstrained Methods** The problems that can be presented as a function of one variable can be easily solved by searching the extremes of the function i.e. its minimum and maximum. In order to extract the needed values, different methods can be used and the basic categories are zeroth, first and second order methods.

*Zeroth order methods* use only the value of the function and try to find the solution through a series of function evaluations, usually applying iterative interpolation processes. For one variable functions, the Bracketing method, Quadratic Interpolation, Fibonacci and the Golden Section Search, etc. are used. For multiple variable functions methods like the Sequential Simplex Method and Powell's Conjugate Gradient Method are often applied [19].

*First order methods* use not only the values of the function but also its first derivatives with respect to their variables. One of the most applied first order methods are the Bisection Method and the Davidon's Cubic Interpolation method, and they use derivatives of the function to find its minimum. With multiple variable functions they span over several different Conjugate Gradient techniques like Beale's and Fletcher-Reeves' method [19].

*Second order methods* use first and second derivatives of the function, aside from its values. In one variable and multiple variable functions, Newton's method is one of the most efficient, leading to development of other, Quasi-Newton, techniques.

**Constrained Methods** The methods described above belong to the unconstrained optimization methods. As structural optimization problems have to satisfy a number of limits and conditions, like maximum displacement, buckling load factor limits or frequency constraints, calculations become much more complex. The optimization then turns to new mathematical areas of Lagrange Multipliers, Quadratic and Nonlinear programming, reduced Gradient Methods and the usage of penalty functions.

### Applicability of Calculus Based Techniques

The techniques mentioned above require a problem that can be presented as a function of one or multiple variables. To guarantee a good solution, the function should preferably be unimodal and continuous and for the use of more effective first and second order methods, the function has to be differentiable. Design variables have to be continuous or otherwise isolated values make the search space discontinuous and disjointed and the derivative information is either not defined or useless. Another common disadvantage is the difficulty of distinguishing between the global and the local minimum of the function and it happens very often that they converge to one of the local minima depending on the search starting point [19].

Since the structural optimization of a grid shell over a specific free form surface is a highly nonlinear process, with discrete design variables, that can hardly be presented as a unimodal, continuous and differentiable functions, rising popularity of stochastic methods proves their suitability for the aforementioned task.

#### 2.2.4 Stochastic Methods

*Since the real world of search is fraught with discontinuities and vast multimodal... It comes as no surprise that methods depending upon the restrictive requirements of continuity and derivative existence are unsuitable for all but a very limited problem domain.*

*David Goldberg*

Talking about the stochastic search, we can start with enumerative optimization methods, where every point in the search space is analyzed. They don't differ much from pure random searches, which do the same thing with a random choice of points in a search space. Although those two principles are very logical from a human, trial and error point of view, they are highly inefficient and, with multi-modal and multi-variable search space, the time to find an optimal solution is simply not acceptable. In the field of stochastic search, we can recognize a large number of other techniques including perturbation, hill climbing, simulated annealing as well as *Swarm Algorithms*, whose usage is slowly becoming significant in the field of architecture and structural engineering. However, for the sake of brevity, all mentioned stochastic methods will not be addressed here. Instead, we will concentrate on Genetic Algorithms and Evolution Strategies. Let us consider in what ways Genetic Algorithms are better for the purpose of grid shell optimization than any of the aforementioned (calculus based or stochastic) methods.



## Genetic Algorithms (GAs)

Probabilistic methods of optimization rely on random selection factors and probabilistic decision. The most effective of them are Genetic Algorithms and Evolution Strategies that come from the naturally observed phenomena. The difference between those two is often vague, as they both rely on the collective learning paradigm and implement the same principles of *population*, *mutation*, *recombination* and *selection* [53]. The basic principle is extracted from Nature and its selection method. Charles Darwin, in his book *On the Origin of Species*, describes extensively the principles of *natural selection*, thus defining the ground rules of Evolution.

*If during the long course of ages and under varying conditions of life, organic beings vary at all in the several parts of their organization, and I think this cannot be disputed; if there be, owing to the high geometrical powers of increase of each species, at some age, season, or year, a severe struggle for life, and this certainly cannot be disputed; then, considering the infinite complexity of the relations of all organic beings to each other and to their conditions of existence, causing an infinite diversity in structure, constitution, and habits, to be advantageous to them, I think it would be a most extraordinary fact if no variation ever had occurred useful to each being's own welfare, in the same way as so many variations have occurred useful to man. But if variations useful to any organic being do occur, assuredly individuals thus characterized will have the best chance of being preserved in the struggle for life; and from the strong principle of inheritance they will tend to produce offspring similarly characterized. This principle of preservation, I have called, for the sake of brevity, Natural Selection.*

Charles Darwin [10]

Since the appearance of the first amoebas, life was sustained through reproduction, i.e., multiplication. In this process, Nature started introducing small random mutations. Those mutations introduced diversity and, in combination with the environment, were responsible for the creation of unique individuals. Each of the newborns was alive for a different amount of time and some of them reproduced more than the others. Eventually, the ones that lived longer and reproduced more, passed their genetic information to their offspring, thus enabling the survival of their *specie*. British philosopher Herbert Spencer used Darwin's idea and made parallels to the capitalistic economy, thus coining the term *survival of the fittest* [9]. That principle, where the best designs survive and bad

ones disappear is the basic idea behind the optimization with Genetic Algorithms. Their development is not surprising at all, since if we think about it, every design process, happening in our minds, follows the same logic. Every time we try to think of some acceptable design solution we: 1.Generate a set of random solutions, 2.Select some of them, 3.Alter them, 4.Combine them to develop a new set of solutions, 5.Repeat 2-4 until we find a satisfactory design. That is exactly how Genetic Algorithms work and the process described has one to one correspondence with GAs in the form of: 1.Generation, 2.Selection, 3.Mutation and 4.Crossover. Therefore it is clear that every design process is an optimization and its results depend on our ability to choose the best solution from all possible solutions.

Computational implementations of evolution started in the 1950s. In the 1960s Ingo Rechenberg defined the *Evolution Strategies (ESs)* and used them as an optimization technique to solve engineering problems [45]. Following the same principles of evolution, in 1975 professor John Holland published a book called *Adaptation in Natural and Artificial Systems* [21]. It is referred to as the primary monograph on the topic of Genetic Algorithms and it is definitely responsible for their popularization. His work was continued successfully by his student David Goldberg [16].

*Genetic algorithms (GAs) are an accepted system of optimization suitable for multi-objective and highly non-linear optimizations. It is a stochastic method that is however no simple random walk. GAs efficiently exploit historical information to speculate on new search points with expected improved performance.*

*David Goldberg [16]*

As Goldberg points out, Genetic Algorithms are not a simple random search. Probably the most difficult part to explain is how and why that is true. That is the question that will be answered easily after presenting the characteristics of GAs and the way of their application for grid shell design in Chapters 4 and 5. It will be shown that, although based on an initial random choice of solutions in the search space, and additional random influences of crossover and mutation, the method does converge very fast to a global optimum, if the problem is described appropriately and the functions and GAs settings are intelligently tuned. Their search doesn't have a single starting point but a population of points, which ensures a better coverage of the search space. The size of the population determines the coverage and it can be defined according to the problem.

GAs do not work with parameters directly. There is a system of coding that enables a large number of parameters for a single

problem to be combined into a finite-length string. In that way a difficult or in some cases impossible mathematical definition of a set of parameters enables their definition and optimization.

GAs do not require that the problem be presented as a continuous differential function, since they do not work with derivatives. Instead, they use an objective function (payoff information) which enables us to address a much larger set of problems and, maybe even more importantly, easily upgrade the system.

GAs differ in their use of probabilistic rules, not deterministic ones. With the rise of quantum mechanics we are more open to the notion of chance playing an important role in the world around us. The evolution process is therefore seen as a probabilistic interaction between the different environmental variations and it is applied in such a manner for the structural optimization. Charles Darwin showed that Nature uses random mutation and in that way creates a huge variety of designs [10]. As time goes by only the fittest survive and in that way optimize the design. This is a very simple principle that can be applied in structural engineering efficiently.

### **Comparison of Genetic Algorithms and Evolution Strategies**

We'll make a brief digression to address the question of GAs and ESs. Namely, it is often confusing to determine differences between those two techniques, since they are based on the same principles and on the conceptual level those differences tend to disappear. An additional reason for confusion is that in practice it is very common to combine those two techniques in order to come up with a good optimization process. In the research presented that is also the case.

What they have in common are the principles of *population*, *mutation*, *recombination* and *selection*, as it was already mentioned. They differ only in their implementation. The main difference is in the *coding* of a *chromosome*. GAs work in general on fixed-sized bitstrings and ESs work on real-valued vectors. In other terms, GAs work with the *Genotype* level of individuals and ESs with the *Phenotype* level of individuals [53]. GAs are more artificial, whereas ESs tend to imitate the process of natural selection with a smaller measure of mathematical simplification. There are a lot of other minor variations in selection, mutation and reproduction that are not so significant, since every method has a version that can be implemented with both strategies. So there is no need to go into unnecessary detail. A thorough explanation of our implementation can be found in Chapter 4, where all the terms that may seem abstract now will be clarified with the demonstration of their application. It will be seen that GAs can be considered as the base method of our research, with some coding and reproduction meth-

ods taken from ESs. But generally, there is no reason to make clear distinctions. Both principles are *Evolution* methods, with lots of variations in implementation that can be set up to response to a specific problem. Nevertheless, since the research is based mostly on processes typical for Genetic Algorithms, they will be referred to as the basic principle used, throughout all of the following chapters.

# 3

## From Surface to Grid

Considering the topic - *free form grid shell design* - the use of *free form* and *grid* have to be addressed in more detail. A short explanation of NURBS is offered as a basic mathematical tool for surface representation. The genesis of grid structures is then presented in the form of different tessellation techniques. Namely, Voronoi and *Voronax* diagrams together with some other possibilities will be shown, and the principles and methods of their application for grid shell design will be demonstrated.

### 3.1 NURBS

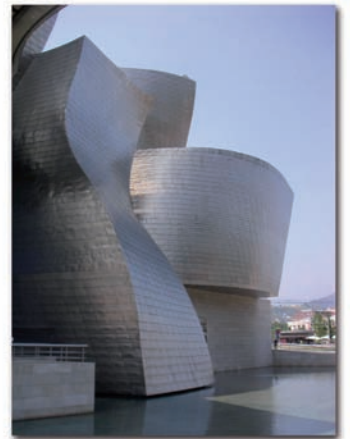
#### 3.1.1 Design

Contemporary free form shaped buildings have manufacturing principles with the roots in the aeronautical and ship-building industry. The majority of construction methods in free form architecture today (taking Frank O. Gehry's buildings as an example) is made with the help of vast experience in ship, aeronautical and car industry. Aside from manufacturing methods, the software able to represent free form surfaces, with mathematical precision, was initially developed for ship and car bodies.

Pierre Bézier, an employee of Renault, and Paul de Casteljaou from Citroën, pioneered the principle in the 1950s with the polynomial representation of curves. From Bézier splines the problem was generalized to create non uniform, rational B-Splines and was eventually developed into Non Uniform Rational Basis Spline surfaces, or NURBS surfaces.

Due to the fact that it is possible to represent practically any shape with the use of NURBS, they entered the CAD world in the 80's and prospered very fast to become the main tool today for the geometrical representation of free form in all fields of design.

The research presented is made with the help of Rhinoceros 3D software, a commercial NURBS based 3D modeling tool. One of the creators (Lowell Walmsley) once said to me, "*We made it for*

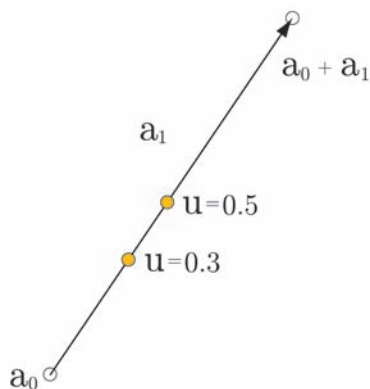


**Figure 3.1:** up-Guggenheim Museum, F.O.Gehry, Bilbao, Spain, 1997  
down-Citroen DS, Series1 (1955-1962)

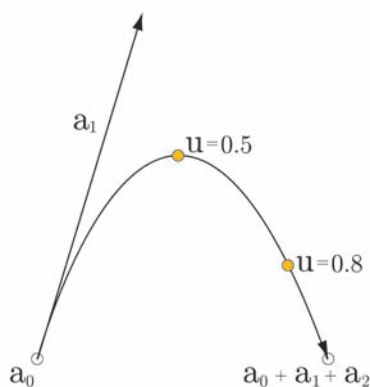
the industrial designers. We never imagined it would be used for houses, shopping malls or even airports”.

### 3.1.2 Mathematics

Putting all the design advantages aside, the mathematical description of NURBS surfaces will be used to explain why their parametric definition makes them perfect for the research presented. The detailed mathematical description is highly complex and therefore, an attempt will be made to sum it up into a short history of the development of free form geometrical representation. The comprehension of polynomial and parametric nature is important for the further understanding of arguments for their application. To make it as simple as possible, an evolution from straight lines to NURBS curves will be shown. We will start with the parametric definition of a curve, and see how that parameter ( $u$  for curve and  $u, v$  for surface) remains as a constant factor in all functions, while they are expanded with the addition of new variables, like *weights*, *knots*, etc.. Although every method along the way can be easily expanded from curve to surface definition, for the sake of brevity, that expansion will be addressed only when NURBS geometry is clarified.



**Figure 3.2:**  $C(u) = a_0 + a_1u$ , Power based curve of degree 1



**Figure 3.3:**  $C(u) = a_0 + a_1u + a_2u^2$ , Power based curve of degree 2, parabolic arc

#### Power Basis Curve

In a simple straight line definition, a distinction is made between explicit form  $y = mx + b$  and parametric form  $x = x_0 + au, y = y_0 + bu, z = z_0 + cu$ . If an independent parameter  $u$  is created, the function of a curve in space can be represented over coordinates  $x, y, z$  which are then expressed over that parameter. The parameter is defined inside a specific interval  $[a, b]$ , usually  $[0, 1]$ .

$$C(u) = (x(u), y(u)) \quad , \quad a \leq u \leq b \tag{3.1}$$

From here an  $n$ th degree power basis curve is defined as:

$$C(u) = (x(u), y(u), z(u)) = \sum_{i=0}^n a_i u^i \quad , \quad 0 \leq u \leq 1 \tag{3.2}$$

where  $a_i = (x_i, y_i, z_i)$  are vectors. To illustrate that, in Figure 3.2 and 3.3 curves of degree 1 and 2 are shown and it is clear how the change of parameter  $u$  from 0 to 1 gives all the points on the curve.

#### Bézier Curves

Since the power basis curve is unnatural for interactive shape design, Bézier curves were developed as a mathematically equivalent form, yet more suitable for geometric modeling, i.e., manipulation

of the shape with the use of computer. The general form of an  $n$ th degree Bézier curve is:

$$C(u) = \sum_{i=0}^n B_{i,n}(u)P_i \quad , \quad 0 \leq u \leq 1 \quad (3.3)$$

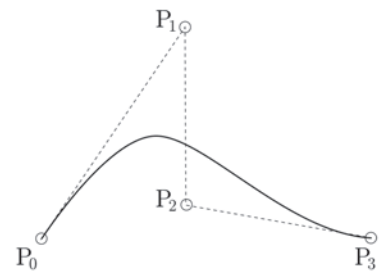
Two important ingredients are introduced here. First,  $P_i$  as a vector of control points and second, a basis function  $B_{i,n}(u)$ . The  $P_i$  is represented over coordinates in space  $P_i = (x_i, y_i, z_i)$  and basis functions are known as Bernstein polynomials of degree  $n$ :

$$B_{i,n}(u) = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \quad , \quad 0 \leq u \leq 1 \quad (3.4)$$

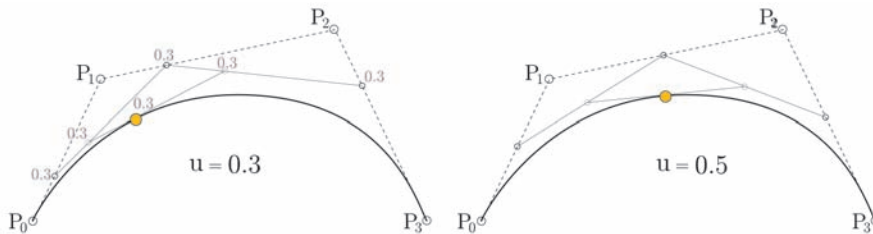
For the sake of brevity a detailed definition must be avoided and a simple example (Figure 3.4) of a cubic (3<sup>rd</sup> degree) Bézier curve will make things clearer. Degree 1 would be a straight line, degree 2 a parabolic arc, and if degree  $n = 3$  the expression is:

$$C(u) = (1-u)^3 P_0 + 3u(1-u)^2 P_1 + 3u^2(1-u) P_2 + u^3 P_3.$$

$P_0, P_1, P_2$  and  $P_3$  represent control points, and the polygon between them represents the approximate shape of the curve. By simply changing the parameter  $u$  from 0 to 1 and importing that value into the Equation 3.1.2, we get all the points on the depicted curve. One of the numerically stable ways to evaluate Bernstein polynomials in a recursive manner is a De Casteljaus's algorithm. It is based on a simple principle of recursive subdivisions of line segments. In Figure 3.5 it is shown how the curve is formed when parameter  $u$  travels from 0 to 1, defining along the way a division point of all the lines that connect the control points. That means that if, for example,  $u = 0.3$ , we define points on all lines with that ratio ( $P_0 + 0.3P_1, P_1 + 0.3P_2, \dots$ ), and on all the new created lines too, until we get the single point (marked yellow) that belongs to the Bézier curve.



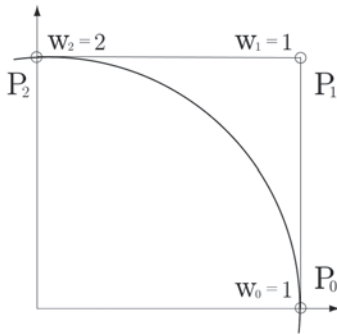
**Figure 3.4:** 3<sup>rd</sup> degree Bézier curve



**Figure 3.5:** De Casteljaus's recursive algorithm

### Rational Bézier Curves

Since Bézier Curves cannot be used to precisely represent conic sections (like circles, ellipses, hyperbolas, spheres, etc.) using polynomials, *rational functions* are implemented, introducing the concept



**Figure 3.6:** Circle arc, defined with the help of weights added to Bézier curve

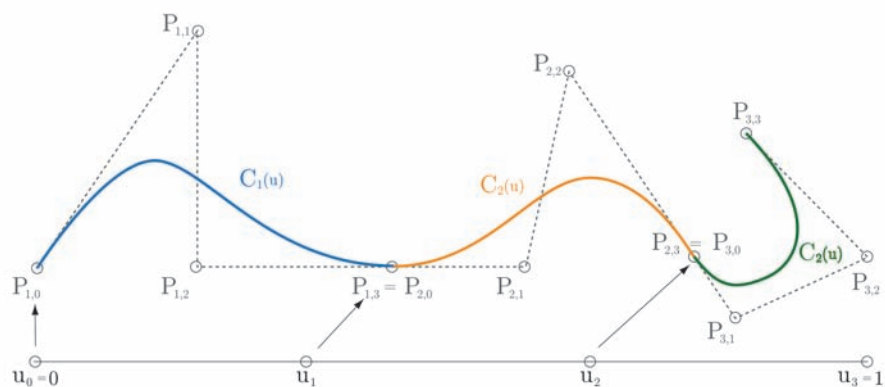
of *weight*. If there is a *weight vector*  $\{w_i\}$ , the  $n$ th degree rational Bézier curve can be expressed as:

$$C(u) = \frac{\sum_{i=0}^n B_{i,n}(u)P_iw_i}{\sum_{i=0}^n B_{i,n}(u)w_i} \quad , \quad 0 \leq u \leq 1 \quad (3.5)$$

In this way simple conical sections, like circle, could be represented as shown in Figure 3.6. We basically just added another factor (a number) that will *pull* the curve toward the control point (that is why it is called *weight*), thus correcting the curves and giving us an additional control. By manipulating weights we have more control in design.

### B-Spline Basis Function

To go further in generalization, the restrictions of rational Bézier curves were considered. The main downside was that they consist only of just one polynomial or rational segment. A curve through  $n$  data points therefore needs an  $n - 1$  degree Bézier curve, making it complex and numerically unstable [44]. The solution was to use *piecewise polynomial* curves, which overcome some of the drawbacks of regular Bézier curves. The curve is then basically divided into  $k$  segments and a vector of breakpoints  $U = \{u_i\}, 0 < i < k$  between those segments is constituted. It is called *knot vector* and breakpoints are called *knots*. The knot vector determines where the polynomials start and stop in the parameter range as the curve is drawn. In Figure 3.7 a piecewise polynomial function is shown, with 3 polynomial segments in Bézier form. This method of rep-



**Figure 3.7:** Piecewise polynomial function

resentation still had problems with continuity and representation. To solve those problems, a generalization of Bézier curves is made, with the help of B-Splines basis functions. The definition of  $i$ th B-Spline basis function of  $p$ -degree (order  $p + 1$ ) over a knot vector  $U = \{u_1, \dots, u_m\}$  is expressed as:



$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u \leq u_{i+1} \\ 0 & \text{otherwise} \end{cases} \quad (3.6)$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \quad (3.7)$$

A B-Spline function therefore consists of  $n$  Bézier curve segments with  $C^2$  continuity, thus enabling the conversion of any Bézier curve to a B-spline and vice versa.

### B-Spline Curves

Now, we have seen that a B-Spline Basis function basically combines  $n$  Bézier curves into one expression. Pretty straightforward, a B-Spline basis function is then used to define a B-Spline curve. So combining it with control points, a  $p$ th-degree B-Spline curve can be represented as:

$$C(u) = \sum_{i=0}^n N_{i,p}(u) P_i \quad , \quad a \leq u \leq b \quad (3.8)$$

There  $P_i$  is again a control point vector and  $\{N_{i,p}(u)\}$  represents  $p$ th degree B-Spline basis functions, defined over a specific non-periodic (and non-uniform) knot vector  $U$ :

$$U = \{a, \dots, a, u_{p+1}, \dots, u_{m-p-1}, b, \dots, b\}$$

This flexibility in the knot vector mapping (0,0,0,1,4,4,5,7,8,11,11,11) is what the phrase *non-uniform* in NURBS refers to. A *uniform* knot vector, e.g.,  $[0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1]$  would ensure the distances between knot values to be equal.

### Rational B-Spline Curves and Surfaces

Since B-Spline curves are still polynomial curves that cannot represent many useful simple curves such as circles and ellipses, a generalization of B-Splines led to Rational B-Splines. By introducing rational functions and weight, like in the example of rational Bézier curves, a general definition for a  $p$ th degree NURBS Curve is obtained:

$$C(u) = \frac{\sum_{i=0}^n N_{i,p}(u) P_i w_i}{\sum_{i=0}^n N_{i,p}(u) w_i} \quad , \quad a \leq u \leq b \quad (3.9)$$

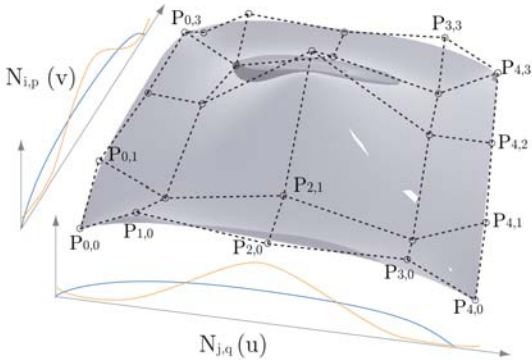
Where control points  $\{P_i\}$ , weights  $\{w_i\}$  and  $\{N_{i,p}(u)\}$  define a  $p$ th degree B-Spline basis function over a specific non-periodic and nonuniform knot vector  $U$ .

Going from curve to surface is relatively simple. We only have to import another parameter  $v$  for the second direction, together with

its own B-Spline basis function and its own knot vector. And when another dimension is added, we expand the definition and finally create a NURBS surface of degree  $p$  in the  $u$  direction and degree  $q$  in the  $v$  direction as a *piecewise rational polynomial function*:

$$S(u, v) = \frac{\sum_{i=0}^n \sum_{j=0}^m N_{i,p}(u) N_{j,q}(v) P_{i,j} w_{i,j}}{\sum_{i=0}^n \sum_{j=0}^m N_{i,p}(u) N_{j,q}(v) w_{i,j}}, \quad 0 \leq u, v \leq 1 \quad (3.10)$$

Now  $\{P_{i,j}\}$  represents a network of control points in two directions  $(i, j)$ ,  $\{w_{i,j}\}$  represents their weights and there are non-rational B-Spline basis functions in two directions  $\{N_{i,p}(u)\}$  and  $\{N_{j,q}(v)\}$  defined over two knot vectors  $U$  and  $V$ . In Figure 3.8 there is a simple graphical representation of a NURBS surface and its control points.



**Figure 3.8:** NURBS surface defined over  $u$  and  $v$  parameters with the help of a control point network

**So Why NURBS?**

The brief analysis of NURBS curves and surfaces was made to show that, apart from the huge design advantages and ubiquitous application, any kind of spatial free form surface can be represented as a polynomial function of two independent parameters  $u$  and  $v$ . Respectively, any point on the surface has a unique  $u$  and  $v$  value and therefore the 2D tessellation technique can be used for the division of spatial surfaces. Transformation from  $xy$  to  $uv$  coordinates, and vice versa, is computationally pretty straightforward. In the following chapters it will be explained how this characteristic enables the usage of 2D Voronoi diagram algorithms to form a great range of grid shell solutions over a predefined free form NURBS surface. Additionally, it will be shown how a variety of 2D tessellation patterns can be easily applied to define a grid over any NURBS surface.

## 3.2 Voronoi Diagram

### 3.2.1 Definition

Named after Ukrainian mathematician Georgy Voronoi, the Voronoi diagram is a system of space decomposition, also known as Voronoi decomposition or Voronoi tessellation. In some cases it is referred to as Dirichlet tessellation, named after the German mathematician Johann Peter Gustav Lejeune Dirichlet, who used the diagrams in his studies 50 years before Voronoi, but the Ukrainian mathematician defined general  $n$ -dimensional case in 1908 and therefore holds the credit.

If there is a set of points  $P = \{p_1, p_2, \dots, p_n\}$  in the plane (called *sites* or *Voronoi seed*) and if  $\text{dist}(p, q)$  is the Euclidean distance between two points in space, then the Voronoi diagram of  $P$ , or  $\text{Vor}(P)$ , can be defined as the subdivision of the plane into  $n$  cells, one for each point in  $P$ , with the property that some point  $q$  lies in the cell corresponding to a site  $p_i$  if and only if  $\text{dist}(q, p_i) < \text{dist}(q, p_j)$  for each  $p_j \in P$  and  $j \neq i$  [11].

In simple terms, for a set of points a plane is divided into Voronoi *cells* in a way that each cell belongs to a specific point (*site*) and that every point in that cell is closer to that site than any other. The border lines between cells are bisectors. Every point on one bisector is at an equal distance from the two neighboring sites. Those border lines between cells form the Voronoi Diagram. An example of the plane subdivision with the Voronoi diagram for 9 points is depicted in Figure 3.9, and the Voronoi diagram with the larger set of points in Figure 3.10.

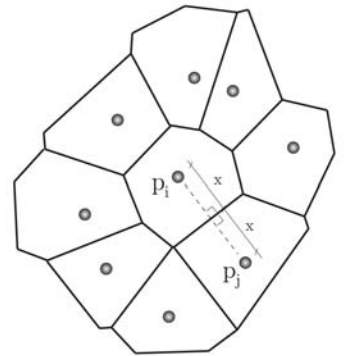


Figure 3.9: Voronoi diagram

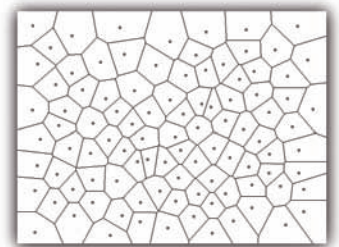


Figure 3.10: Voronoi diagram for random set of points

### 3.2.2 Why the Voronoi Diagram

#### Nature

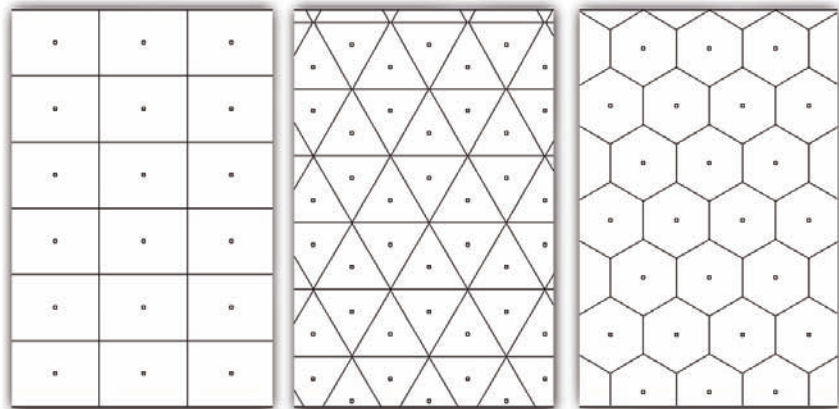
Voronoi diagram appears all around in Nature. At the microscopic level, it is recognized in the basic principle of cell division. At the macroscopic level, the pattern on a giraffe's skin or the tessellation of a turtle shell can be easily identified as having the same principle. The familiar pictures of cracks in the dry earth are also a result of scattered points of contraction resulting in cracks dividing the earth into Voronoi cells. This is depicted in Chapter 6, where the connections between natural and Voronoi-based structures are presented.

#### Cell Approach

Working with cells, not grid lines, turns out to give more freedom when considering how to generate a grid structure over some free form surface. The network of lines is a finite graph and, as already

familiar from graph theory, graphs are defined by *vertices* and their connections. In order to realize patterns and possibilities of connections between random points, different constraints regarding grid shell structures have to be considered. For example, no two members should cross each other. That one simple condition introduces a high level of complexity, sometimes solvable (like Delaunay triangulation for triangular grids), but generally making subdivision of surfaces into acceptable n-gon meshes a very difficult task. It is often easier to design a polygon mesh when it is observed as a group of faces (fields, cells) then as a connected system of lines. We can experiment with different cell size and different density, thus trying to find an optimal configuration without thinking about grid lines and their connectivity.

An additional property is that the average number of edges of a Voronoi cell is less than six [24], which is usually structurally acceptable. In Section 3.1, the parametric nature of NURBS surfaces was explained, and it was stated that a relatively simple algorithm for 2D Voronoi tessellation based on the seed points can be easily mapped to a NURBS surface topology over  $uv$  parameters. In this way, a 2D Voronoi algorithm can be applied to generate a grid shell over some spatial free form surface. With the careful *planting* of Voronoi seeds, any kind of grid can be generated, regular or irregular. It is possible to build triangular, quadrangular, hexagonal grids and to combine them (Figure 3.11), but also very free, natural-like, Voronoi diagrams as the one in Figure 3.10 are also possible.



**Figure 3.11:** Voronoi diagram for geometrically regular set of points

The combination of the Voronoi diagram and NURBS surfaces for the design of grid shells will be described in detail in Section 4.2. Before that, it is important to mention the basic algorithm that was implemented.

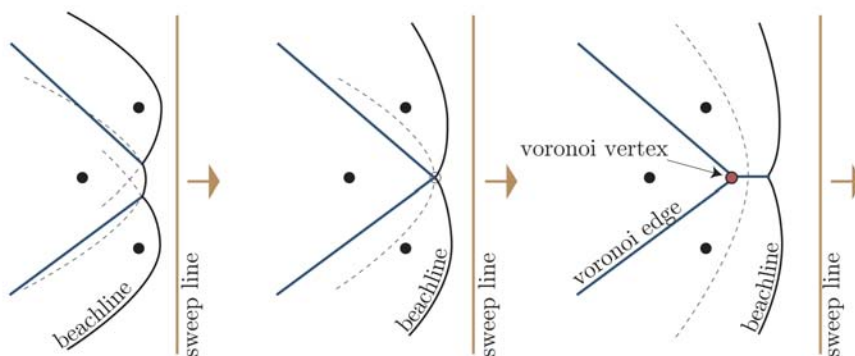
### Fortune's Sweep Line Algorithm

Computation of the Voronoi Diagram can be done in a relatively simple way, where for each point  $p_i$  the common intersection of the half planes  $h(P_i, P_j)$ , where  $j \neq i$ , is found. This algorithm is however very inefficient, needing  $O(n \log n)$  time for each Voronoi cell, thus  $O(n^2 \log n)$  to compute the whole diagram.

Steven Fortune developed an algorithm that computes the whole Voronoi diagram in  $O(n \log n)$  time, using one of the standard techniques in Computational Geometry - the *sweep line* method [11]. Hence, the method known as *Fortune's sweep line algorithm* is used in the research.

The strategy of this method is to sweep a straight line from left to right (or top to bottom) over a plane containing a set of sites (points). Since the parabola represents a set of points equidistant from an observed site and a line, for each site on the *front*, a complex so-called *beachline* is created at every stage of the sweep line movement. Now, as the line progresses, the intersections between *beachline* parabolas actually draw the Voronoi diagram. The moment parabolic arcs disappear from the beach line, vertices of Voronoi diagram are created.

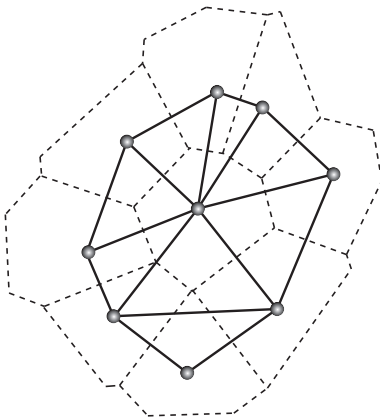
A detailed description of the algorithm and some solutions for a set of special cases is offered in [11]. In Figure 3.12, a three step progress is shown for three sites. Parabolas define equal distances between points and sweep line (yellow), blue lines represent the emerging Voronoi Diagram and the red point is a newly created Voronoi vertex. As mentioned, points on the NURBS surface are basically mapped to the  $XY$  plane, according to their  $uv$  parameters, and after the Voronoi diagram is computed, its vertices are mapped back onto the surface. The mapping will be addressed in more detail in Chapter 4.



**Figure 3.12:** Fortune's sweep line algorithm

### Delaunay triangulation

Having a set of points in plane  $P$ , we can connect them to create a triangulation of the plane. If the circumcircle of any triangle of the network is empty, i.e., doesn't contain any other point, then the triangulation is called a Delaunay triangulation. Fortunately there is a one-to-one correspondence between the bounded cells and the vertices of  $Vor(P)$  that makes its *dual graph* - Delaunay graph  $DG(P)$ , from which we obtain a Delaunay triangulation by adding edges [11]. That means that any Voronoi diagram can be triangulated, by connecting its seed, thus having all the properties of a Delaunay triangulation. Nodes of this graph are sites of the Voronoi diagram and it has an edge between two sites if their corresponding cells share an edge. A small example is shown in Figure 3.13, where the points represent sites, dashed lines represents the Voronoi diagram and full lines its dual graph, the Delaunay triangulation. This method of division was invented by, and named after a Russian mathematician Boris Delaunay [11] in 1934 and some of the characteristics of such plane division are:



**Figure 3.13:** Delaunay triangulation, Voronoi diagram's dual graph

1. No two edges ever cross each other - it is a plane graph.
2. The circumcircle of any triangle in a Delaunay triangulation contains no point of  $P$  in its interior.
3. Any Delaunay triangulation of  $P$  maximizes the minimum angle over all triangulations of  $P$ .
4. Each vertex has on average six surrounding triangles.

Delaunay triangulation is considered as a tessellation method for the research since it has a number of good properties that can be useful in a grid shell design. It is however used in a small number of experiments because triangular grid shells are heavily studied and applied so far, hence there is not much more space left for innovation. The detailed proof of properties and a detailed description of the Delaunay triangulation can be found in [11, 24].

In Section 5.4 it will be demonstrated why there were a small number of optimizations done with Delaunay triangulated grids. The results of such optimizations could hardly be interpreted, i.e., the practical use of such optimizations showed to be small. One of the goals of future research is to investigate this further and to be able to do optimizations with Delaunay triangulated grids that will result in optically and statically acceptable solutions.

## 3.3 Voronax

Thinking about ways to alter the Voronoi diagram and adjust it to fit our goal of optimization, an iterative version of the Force Density method was developed, as part of the presented research, to *relax*

the grid structure while keeping it on the surface. Voronax is one of the most important and innovative parts of the work introduced. It is offered as a new system for grid shell design, but much more important, it can be used as a guide to show the best distribution of grid shell structural members over a given surface. The Voronax grid can be examined to see how the member density is distributed. In that way, a pattern of behavior can be extracted, making us realize how to distribute structural members over a predefined free form surface to gain an optimal solution. An explanation of the Force Density method and Voronax generation follows, together with its implementation and the argumentation for its use.

### 3.3.1 Force Density method

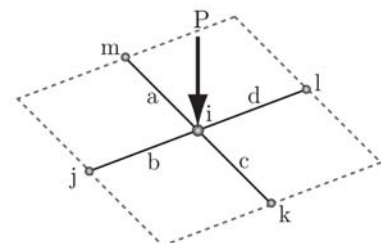
#### History

The method was developed while searching for the analytical solution that would describe experiments done by Frei Otto, for the cable-net roof design of the 1972 Munich Olympic Games stadium [27, 35]. Since the physical models couldn't be precise enough to estimate the final form and eventually derive the cutting pattern for a cable net, Linkwitz and Scheck [38] formulated a system of equilibrium of forces in 1971 that was named the Force Density method (*Kraft-Dichte Methode*).

#### Method

It is stated that pin-joint network structures assume the state of equilibrium when internal forces  $t$  and external forces  $s$  are balanced. Figure 3.14 demonstrates the principle found in one structural joint.

The internal cable forces are defined as  $t_a, t_b, t_c, t_d$ . In order to decompose forces of member  $a$  into three main axes, force  $t_a$  is multiplied by  $\cos(a, x)$ ,  $\cos(a, y)$  and  $\cos(a, z)$ . The cosine values can therefore be defined as the projection of lengths in form  $(x_n - x_i)/a$  for the member  $a$  and its  $x$  component. This is then done for all cables in one joint  $(t_a, t_b, t_c, t_d)$ . In order to solve the system, some initial values of the internal forces have to be established. They are referred to as *tension factors* and will be addressed in the next section. The force  $P$  acting in joint  $i$  is hence decomposed into three components  $p_x, p_y, p_z$  and we get the following system of equations:



**Figure 3.14:** One joint in the structure with  $P$  as a resulting force from all connected members

$$\begin{aligned} \frac{t_a(x_m - x_i)}{a} + \frac{t_b(x_j - x_i)}{b} + \frac{t_c(x_k - x_i)}{c} + \frac{t_d(x_l - x_i)}{d} &= p_x \quad (3.11) \\ \frac{t_a(y_m - y_i)}{a} + \frac{t_b(y_j - y_i)}{b} + \frac{t_c(y_k - y_i)}{c} + \frac{t_d(y_l - y_i)}{d} &= p_y \\ \frac{t_a(z_m - z_i)}{a} + \frac{t_b(z_j - z_i)}{b} + \frac{t_c(z_k - z_i)}{c} + \frac{t_d(z_l - z_i)}{d} &= p_z \end{aligned}$$

Solving the system gives the resulting force  $P_i$ , and after the assembly of the system of equations for the whole structure, forces in all joints can be calculated in order to move the structure into the state of equilibrium. The system of equations assembled in that way is extremely sparse and can be solved efficiently with different methods (like Conjugate Gradient method). More about that can be found in [37, 36].

### 3.3.2 Constrained Force Density Method

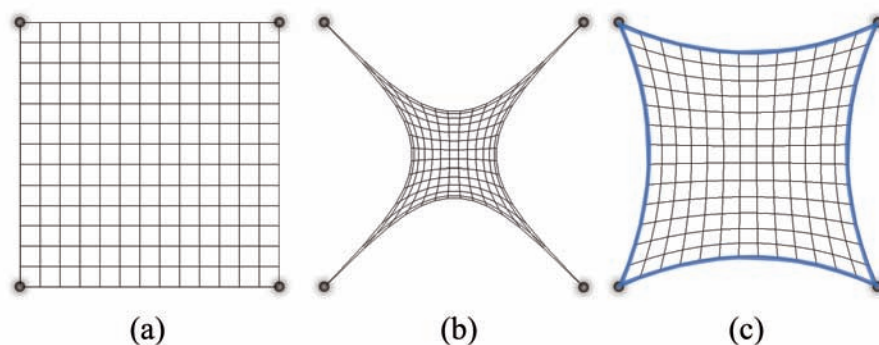
How can a search tool for minimal surface solution in cable-net and membrane structures be used for a free form grid shell design? In Section 2.1 it was mentioned that *relaxation* method was used for the MyZeil project in Frankfurt [22], and also for the EXPO project in Shanghai [51]. An explanation of the ways that Force Density method was adjusted for the purpose of grid shell design follows.

#### Tension factor

Grid shells are also network structures, and they differ from cable-net structures in that their members can take over pressure (in addition to tension). However, that doesn't change the situation greatly, since the resulting force in every joint can still be found, and an equilibrium state can still be assumed to be the balance of internal and external forces. It was mentioned that in order to find a solution, initial forces have to be given. Those forces were presented above as *tension factors*, assigned to each bar in the structure. Their values have meaning, however, only in comparison to other member values and are therefore relative. The structure with a tension factor of 1 or a tension factor of 100 in all of its members has the same isotropic property. This value can be represented as the prestress value in the cable-net structure. To demonstrate the effect it produces, a *relaxed* structure, with fixed corners, where all the cables have the same tension value, is shown in Figure 3.15 (b), and in 3.15 (c) the same structure with the values of edge cables 10 times higher is given as an example. Figure 3.15 (a) shows the grid before the application of the relaxation algorithm.

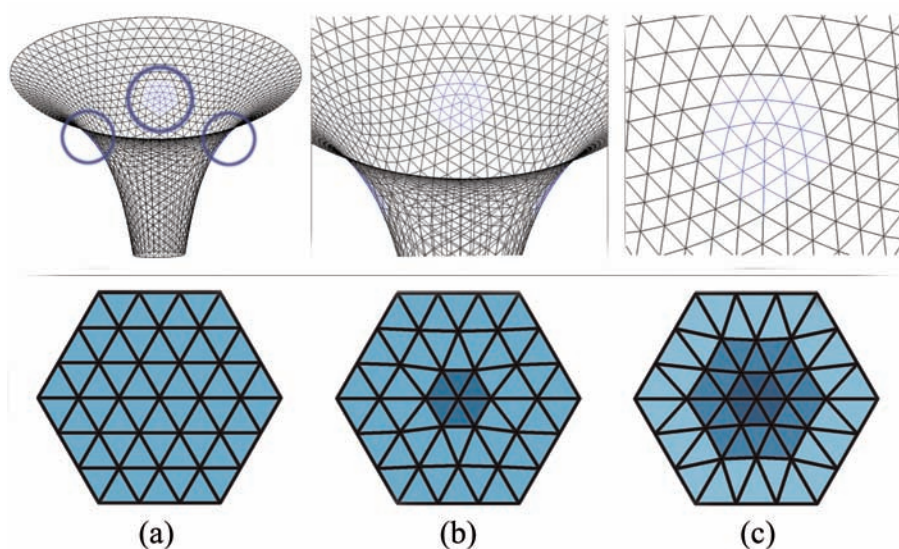
In that sense, in the funnel-like structures of the EXPO Axis in Shanghai, the structural members near the membrane roof connections were given higher tension factors, thus pulling the structure





**Figure 3.15:** Mesh relaxation

toward them and resulting in statical optimization, depicted in Figure 3.16. On the left (a), part of the relaxed mesh with a uniform tension factor (10) is shown. Then we see the same structure relaxed with the tension factor of members in the center (marked dark blue) enlarged to the value of 15. And in the third version (c), we can see a relaxed structure with factor 20 in the central members, 15 in members surrounding them and factor 10 in the outer members. The mesh is therefore *pulled* toward the center point.



**Figure 3.16:** **up** - One *Sun Valley*, EXPO 2010, Shanghai, with the areas where the grid was *pulled* and **down** - the abstraction of the pulling method

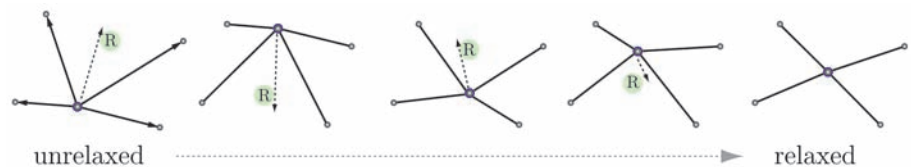
Now, it is clear that any set of forces can be added to the equations. For example, in each node an additional vertical force can be applied to simulate the gravitational effect, or a horizontal force to simulate wind. The system can therefore be solved to gain equilibrium in the structure for any set of external and internal influences. Talking about *boundary conditions*, any node in the structure can

additionally be fixed in space or linked to some free form curve or surface as we will see.

For the research presented, the focus is on the geometrical advantages of relaxed structures. Hence, the tension factors were always set to be uniform without any external forces. That was enough to ensure the advantages needed, i.e., the creation of smooth and uniform grid. Experimenting with different sets of boundary conditions wouldn't be important for the description of the method and the proof of its efficiency. But all design variables that are involved in the optimization process, as well as this one, can be researched in much more detail. Each one has the ability to be branched out and be formulated in an infinite number of ways. However, that possibility is important when a specific project is observed, and when the exact conditions and needs are known. That is why the fundamentals of the optimization algorithm will be set here, and the possibilities will be explained, without going into numerous experiments just to show that it can be done.

### Iterative Process and Surface Constraint

In order to keep the structure on the predefined surface, a different approach from the mathematical solution of system of equations had to be implemented. Namely, not as precise, but very effective, an iterative algorithm was used. Assuming the same principle of equilibrium, the goal is to geometrically arrive at the solution of minimum potential energy in the system. Figure 3.17 shows the principle where one node is observed and a resulting force from the connected members is acting upon it. What happens is that, for the observed node, a resulting vector is calculated from the *tensions* in its members (as shown in Equation 3.11). Tensions are vectors, with their values and directions. If some external forces affected the node additionally, they would just be added into the calculation. The node then moves in the direction of the resulting vector, as that is its natural way to reduce potential energy. That happens in successive iterations and the resulting vector gets shortened with the specific *limitation decay* factor.

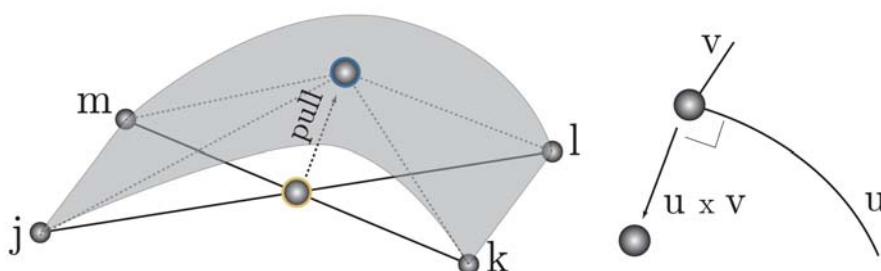


**Figure 3.17:** Iterative Force Density method

A single iteration for the whole structure consists of the movement of all nodes in the direction of the resulting force  $R$  (calculated from its neighboring points) and for the distance represented by the value of this vector. After a specific number of iterations,

the accuracy of which can be easily controlled, the structure moves toward the equilibrium state. It is very important to mention that the method is not as precise as the mathematical solution and is sensitive, in the sense that the initial state can influence the final result. Therefore, the closer the initial state is to the equilibrium state, the more precise the solutions are.

With this iterative method however, the introduction of constraints is pretty easy and straightforward. Each node in the structure can be affected by one of three modes of constraints. It can be free, fixed or linked to some curve or surface. Since the research tries to define a method of grid shell optimization over a given free form shape, all the nodes are either restrained to the surface or to the edge curves. If the node is linked to a curve or surface object, it is repeatedly pulled onto it, following the shortest path. Therefore, after each iteration, each vertex of the structure is pulled onto the surface, keeping the topology of the structure intact. *Pulling* of the vertex basically refers to an algorithm that finds the shortest distance between the point and the observed NURBS curve or surface. The principle of finding that shortest distance, i.e., the point on the surface at which the normal goes through the pulled point, is represented in Figure 3.18. It can be seen that to find the shortest path, we have to find the point on the surface in which the surface normal goes through the pulled point. Mathematically, the surface normal is defined as the cross product of the two tangent vectors, obtained as partial derivatives of the intersecting  $U$  and  $V$  isocurves at the observed point.



**Figure 3.18:** Keeping the grid on the surface by pulling its joints

### 3.3.3 Voronoi to Voronax

#### Getting Free From the Mesh

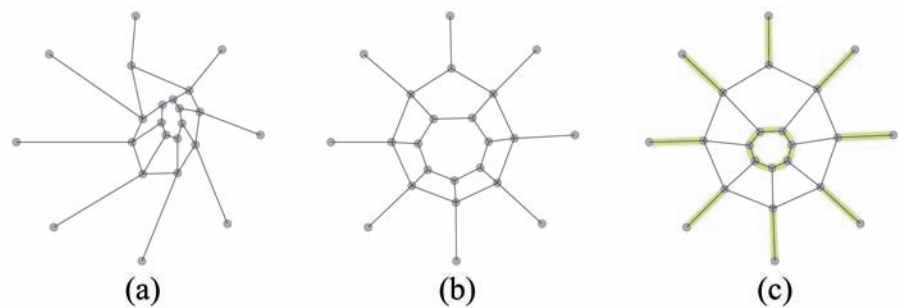
In the examples above, a *mesh structure* was used for an implementation of relaxation algorithms. In computational geometry meshes are data structures with node (*vertex*) and face information. Each vertex is defined over its unique number and coordinates that define its place in Euclidean space. Each mesh face has its number too and an array of vertices that define its borders. From this array,

face edges can be extracted, considering the order of vertices. The orientation of a face can be determined depending on the clockwise or counterclockwise progress of the vertices. Mainly due to the rendering algorithms, in most CAD programs mesh faces are limited to 4 points, therefore enabling the creation of triangular and quadrangular meshes. In that way, a light ray, bouncing from it, could be easily interpolated and calculated for all points on the face according to the 3 or 4 face vertices. Additionally, for grid shell design, the tessellation of surfaces into triangles and quadrangles is what was needed, so they were mainly modeled as polygon meshes.

Since the Voronoi diagram has cells with more than 4 edges, the constrained force density algorithm had to be expanded to work for any graph, i.e., for any connected system of points.

### Graph relaxation

In Figure 3.19(a) we see a simple 2D graph, made out of 23 connected points. In Figure 3.19(b), the same graph is relaxed, thus assuming an equilibrium state. On the right hand side, the same graph is shown relaxed, but with the tension factor doubled in the green members, thus showing how easily the relaxation process can be used as a design tool and how it can be controlled to achieve different solutions. The relaxation of a graph could be done without constraints (link to a surface or a curve), as depicted in Figure 3.19, but our goal is to relax generated Voronoi diagrams while keeping them on the predefined surface. Those types of structure, developed



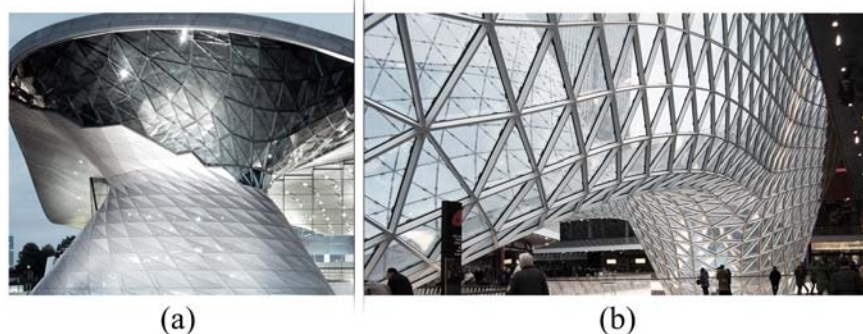
**Figure 3.19:** Graph relaxation

as a part of this research, are named Voronax structures (**Voronoi + Relax**). Voronax, or  $VorX(P)$ , can therefore be defined as a relaxed Voronoi diagram structure. Relaxation of graphs works in the same way as mesh relaxation. The only difference is that the number of members connecting in one joint is not fixed and it can be bigger than four. That is why we first have to analyze each joint in the structure, to see how many members are connected to each. Then we have to calculate the resulting vector for each joint, using the length of its belonging members and their tension factors. When we have the resulting vectors, we continue with the iterative

relaxation process as described before. This method is used to relax Voronoi structures, keeping the grid on the predefined surface with the already explained *pulling* system, ending up with Voronax grid structures.

### 3.3.4 Why *Relaxed* Meshes Are Better Than *Unrelaxed* ones

Having shown the basic principles and the upsides of the method, its applications will be briefly considered. The application of the relaxation method results in geometrically smooth grid structures. The human eye has a remarkable pattern recognition ability and incredible precision of error detection when it comes to grid structures. It is therefore very important to be careful in any kind of *tile* design. One approach is to arrange the tiles of some structure in a way in which they don't form long guide lines. If they do, then the lines must be smooth and continuous, in order to satisfy optical conditions. In Figure 3.20, an example of two grid shells is shown to demonstrate the idea. In the BMW Welt building (a), in the upper part of the hyperboloid, the lines of the structure are broken and not continuous. In the MyZeil grid shell roof example, on the right, it is clear that structural lines are smooth and continuous. This is the effect gained with the use of relaxation methods. In the process of finding the equilibrium state of some structure, it makes the angles between its members change in relation to their neighboring ones, thus changing gradually and resulting in smooth transitions in the grid.



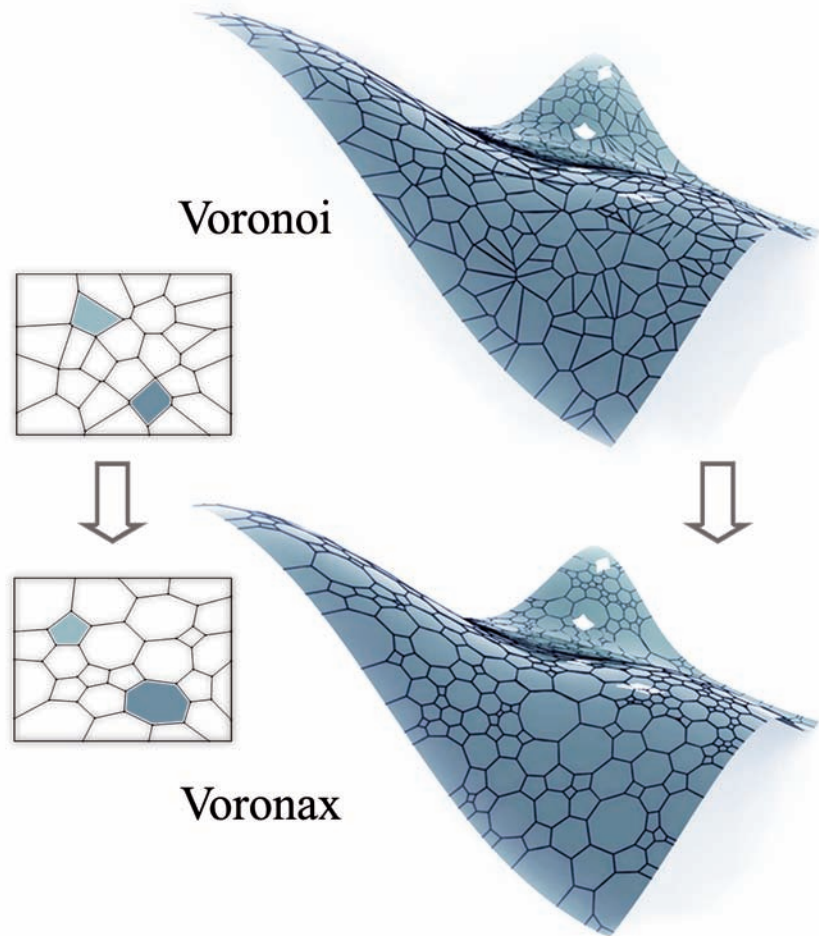
**Figure 3.20:** Discontinuous grid lines and relaxed continuous grid lines

### 3.3.5 Why Voronax Is Better Than Voronoi

Voronoi structures suffer large length deviations of their members. Although the differences in cell size can be relatively controlled by adjusting the distance between the seed points, huge differences in member length is something that cannot be avoided that easily.



However, Voronax fixes the problem. One of the basic rules in Nature is uniform distribution of stress in search of equilibrium and minimum potential energy. That is why after relaxation (which has the same goal) is performed, member lengths are much more uniform, and so are the angles between them, therefore resulting in less distorted polygons. The difference can be seen in Figure 3.21, where a Voronoi grid structure is shown together with the relaxed, Voronax version.



**Figure 3.21:** Voronoi and Voronax

The Voronoi diagram has an interesting property considering its cells. Namely, an average polygon in a Voronoi structure has less than 6 edges and Voronax keeps that property since it doesn't change the topology of the initial Voronoi structure. Voronoi cells are always convex polygons which is another advantage and good reason for their application in grid shell design. However, angles in those polygons are not uniform and large differences can occur. It can be easily seen that Voronoi polygons are distorted. In the relaxing process that problem is also solved and Voronax structures have convex polygons with much more uniform angles. In Chapter

6 the basic difference between Voronoi-like and Voronax-like structures that we find in Nature will be explained. Structures that grow slowly and cannot adjust their structure easily in time are basically Voronoi like structures. Foam is a nice example of an adjustable structure that finds an equilibrium at every moment in time, and that is exactly what Voronax structures do.

So far we have seen how we can generate a large variety of grid structures, regular and irregular, over a given predefined free form surface. It was shown how relaxation algorithms can be used to obtain smooth and optimized grid shells. The Voronax principle was introduced, resulting in a structure that has many good properties, suitable for structural design, and especially for structural optimization, as it will be demonstrated in the following chapters. That being said, in the following chapter the details of our Genetic Algorithms application will be presented and eventually the results of the experiments done with it.

### 3. From Surface to Grid

---



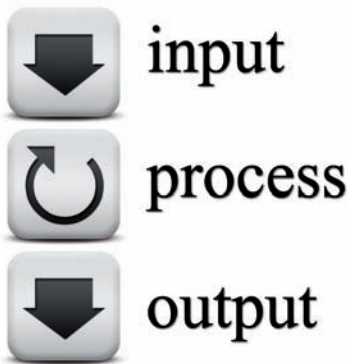
# 4

## Genetic Algorithms

So far, NURBS geometries were described, as well as Evolution-based optimization techniques. New surface tessellation methods were shown, with the use of Voronoi diagrams, together with newly presented Voronax structures. All these elements can be used and combined into a software that helps designing free form grid shells. To be specific, explained methods are materialized in the form of a plug-in for Rhinoceros 3D software, written in C++ language. For the FEM analysis, Oasys GSA Analysis software was used and called automatically from the C++ code with the use of OLE Automation. In the following sections, the structure of the complete algorithm and the implementation of Genetic Algorithms will be presented, thus giving form and shape to terms such as *selection*, *mutation* or *crossover*.

### 4.1 Algorithm

The explanation of the implemented algorithms will be divided into sections, according to the different GAs routines. After the basic conceptual structure is shown, selection, mutation and reproduction (crossover) techniques will be discussed together with other aspects of grid shell representation and evaluation, in an Evolution-based optimization process. Before describing the whole GAs process, it is very important that the notion of *chromosome* is clarified. Its definition and application for the research will be explained in detail in the following section, but for now, it is important to mention that in Genetic Algorithms each individual is represented through a unique chromosome. It is basically an array of numbers, a coded string, that can be used to easily manipulate individuals, combine them, mutate them and eventually store them. It is important to comprehend this, so that the following explanation of the basic algorithm structure, as well as the storing of chromosomes, don't seem abstract. The beauty of programming is precisely that absence of abstract notions. Everything has to have a physical description, that eventually has to be broken down into 0s and 1s.



**Figure 4.1:** Information flow

### 4.1.1 Basic Structure

If we simplify things to the basic level, we can recognize three elemental parts of the whole process: 1. Information input, 2. Optimization process and 3. Information output.

#### Input

The first step of the optimization is the selection of a predefined NURBS surface, designed in Rhinoceros 3D. Then the Windows dialog opens up and enables the user to tune his optimization process. To avoid vagueness, an actual Windows dialog from the created plug-in is shown in Appendix A, with the explanation of some standard values used for the experiments. Dialogs are different for different surface tessellations (quadrangular, Voronoi, etc.), because they work with different sets of geometrical input data. Therefore the most complicated and the most efficient pattern - Voronax - is chosen for the representation in the appendix. Here at the beginning, the input data will only be listed and briefly described, because some of the terms will still be abstract. In the following sections and chapters they will all be addressed and their application will be explained. We start now with the basic categorization of input data into: 1. GAs specific variables, 2. Geometrical variables and 3. Evaluation variables.

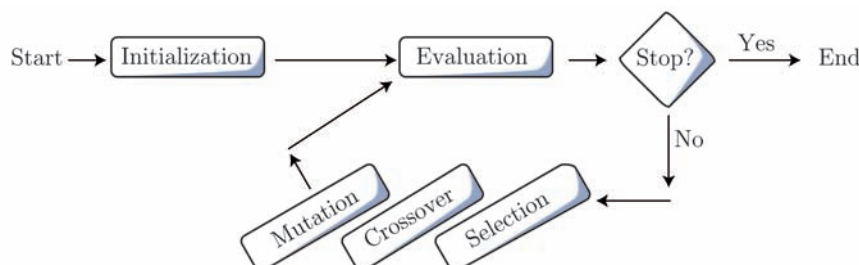
**GAs Specific Variables** These are the variables that define basic GAs parameters. We have *crossover probability* and *mutation probability* that define the chance of an individual to be combined with another one or to be mutated, and they will be addressed in the following Sections 4.1.3 and 4.1.4. *Generation size* and *maximum number of generations* determine how many individuals will be generated in each generation and after how many generations should the optimization process stop. The number of generations is not so important at the beginning, since it is always possible to take the last generation and continue the optimization process until it converges. That is why the possibility of choosing *random generation* or *txt generation* exists in the dialog. Choosing one option, we can define the zeroth(initial) generation, to be randomly generated or imported from a prepared *txt* file. Very important input information is the *chromosome length*. It can be defined by the user directly or indirectly over some other geometric variables, which will all be explained in the following sections. For now, it is important to remember that the chromosome length can be defined at the beginning, and that it then directly determines the number of Voronoi seeds used for the grid structure generation, and from there the number of structural members of each generated grid shell in the optimization process.

**Geometric Variables** These are the variables that define the geometric properties of the grid shell. Namely, the basic pattern is chosen at the very beginning, and in the case of Voronoi-based patterns (Voronoi, Voronax, Delaunay) we can define how many Voronoi points (seed) we want on the surface (*in field*) or on the surface edges (*points on edge U, points on edge V*). For other patterns, like quadrangular or hexagonal, this definition looks a bit different and it will be clarified in more detail in Section 4.2.2, with the explanation of *decoding functions*. The geometry can be additionally controlled with parameters like *minimal allele distance*, which in this case defines the minimal distance between Voronoi seed points. This prevents very small Voronoi and Voronax cells, and also prevents duplicated points which can lead to program errors. Since Voronoi-defined structures can share the same input data, it is clear that the choice between Voronoi, Voronax or Delaunay structure can be defined in the same dialog.

**Evaluation Variables** These are the variables used for the evaluation of the solution, i.e., generated grid shell. They mainly refer to physical characteristics needed for FEM analysis, like material, cross-sections, load, support, etc., and the definition of different objective and constraint functions. Fitness functions and Penalty functions will be thoroughly explained in Sections 4.3 and 4.4, and there it will be clarified how the aforementioned variables affect the entire optimization process.

### Optimization process

A simplified graphical representation of the process is shown in Figure 4.2, deprived of all additional extensive branching that goes out of every presented part. A more elaborated scheme is shown in Appendix B, where the structure of the actual plug-in code is graphically represented.



**Figure 4.2:** Basic GAs Loop

After the initial generation is randomly created (or imported), all its individuals are evaluated, i.e., their *fitness* is determined. The process then enters a loop for a defined number of generations. In this loop, a selection is made to choose the individuals for

breeding, based on their fitness. Chosen individuals are then bred (crossed), mutated and evaluated again, thus prepared for the next iteration. The process stops after a predefined number of generations. The code can be easily altered to stop when a specific fitness value is reached, but experience showed that it is better to continue with the optimization until we are sure that the acceptable convergence has been reached. The processes of selection, crossover and mutation will be explained in the following sections, and the evaluation process, as the most important and most complicated one will be addressed in Sections 4.3, 4.4 and 4.5.

### Output

For the potential analysis of all solutions in the entire generated population, after the optimization process was completed, information had to be kept in separated files. In the research, three different text files were generated during the process and they contain information that can easily be extracted and analyzed. The examples of some actual generated files are presented in Appendix A.

**File 1 : GAs General Description** This is a file containing all of the basic information about the optimization parameters and about the individuals in population. That information consists of their number, rank in the generation, fitness value, scaled fitness value, etc.

**File 2 : Solution Chromosomes** In the Chapter 4.2 it will be explained how the *chromosome* works and what it looks like. For now, it is only important to mention that in GAs each individual (grid shell) is represented through a unique *chromosome*, stored in this second file. That means that each individual from the entire generated population (which sometimes reaches 40.000 individuals) can be extracted from this file, graphically represented (drawn in space) and then examined separately. This file can also be used to extract the last generation. Afterward that generation can be used as an initial generation (instead of the randomly created one) at the beginning of the process. In that way we can break and continue the optimization at will, however many times we want, and the algorithm will continue to make progress as if it never stopped.

**File 3 : Graph Information** Every optimization process can be very effectively evaluated and examined by looking at the progress of the specific values throughout generations. This file contains information about each generation that enables us to draw graphs of their progress. Those values are: 1. The maximum fitness value in a generation, 2. The minimum fitness value in a generation, 3.

Average fitness in a generation, 4. Sum of all fitness values in a generation.

### Clarification

A small digression is needed to avoid confusion. In Figure 4.3 there is a graphical representation of the 11 steps that are performed to transform an individual chromosome into a grid shell structure and evaluate it. Basically all of these steps are repeated for each individual solution in every generation. A list of sections where explanations for each individual step can be found is on the left. First, the basic GAs procedures (Selection, Crossover, Mutation) will be addressed, to show what the optimization process algorithmically does. Each of those steps works with chromosomes (explained in Section 4.2.1), as basic representations of individual grid shells. Detailed explanation of steps 4-11 follows right after the presentation of the basic GAs methods. Therefore, everything that is not clear in the first part will definitely be clarified in the second part, and the complete procedure will be covered in this chapter.

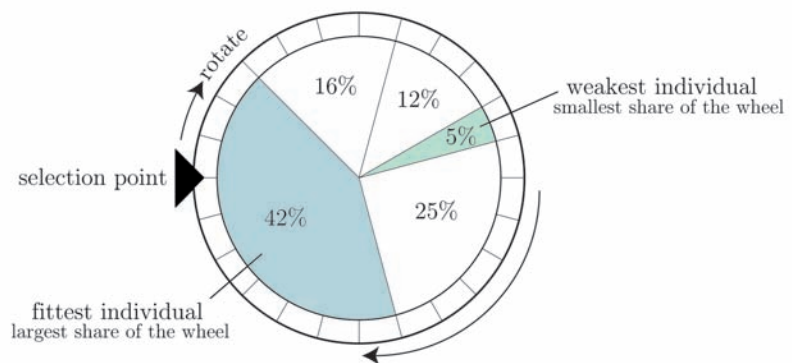
Sections			
4.1.2	<b>1</b>	Select Parents Selecting individuals to participate in the next generation	SELECTION
4.1.3	<b>2</b>	Produce a child Combining the selected individuals to create new ones	CROSSOVER
4.1.4	<b>3</b>	Mutate and Fix Chromosome Random mutation of individuals to introduce diversity Checking if chromosome can be decoded, and fixing errors	MUTATION
4.2.2	<b>4</b>	Decode Transform the chromosome into the points on the surface	GENERATION
4.2.2 (3.2.2)	<b>5</b>	Calculate Voronoi Generate Voronoi Diagram from the obtained points	
4.2.2 (3.3.3)	<b>6</b>	Relax structure Use relaxation to generate Voronax grid from the Voronoi diagram	
4.2.3	<b>7</b>	Prepare for FEM Automatic definition of: nodes, elements, supports, loads, etc.	
	<b>8</b>	FEM Static Analysis Automatic call to the FEM software to perform the static analysis	
4.3 4.5	<b>9</b>	Evaluate Retrieve data from the FEM software Calculate fitness value according to the chosen fitness function	EVALUATION
4.4	<b>10</b>	Penalize Calculate error value according to the chosen penalty functions	
4.3.3	<b>11</b>	Get Final Fitness Value Calculate final value using fitness scaling techniques	

**Figure 4.3:** All the operations performed on one individual grid shell

### 4.1.2 Selection

In Genetic Algorithms, the process of selection chooses individuals of one generation for crossover (*breeding*), according to their fitness. By giving *fitter* solutions greater chances of *survival*, it is responsible for the convergence of the entire process toward the best fitness solution. In order to imitate the process of Natural selection, different methods have been developed in the history of GAs optimization. Some of the less used methods include *tournament selection* and *ranking*. These strategies are so-called *elitist strategies*, since they only allow the small number of best solutions (elite) to survive. In a tournament selection, a tournament type of competition is simulated between individuals in one generation, in which their fitness determines their chances for winning. Only a small number of *winners* are then chosen to participate in the next generation. The ranking technique is similar, all individuals in one generation are ranked according to their fitness and only a couple of top ranking individuals are used for reproduction. These methods enable fast convergence at the expense of modest exploration of the search space. In simple terms, they are fast but not so efficient, with a large probability of convergence to the local optimum of some specified function.

A method used in the research, and the most ubiquitous one in GAs application, is the *roulette selection* method. Roulette selection doesn't *kill* any of the solutions. Instead, it gives them all a chance of survival, proportional to their fitness. In this way the variety of solutions is bigger and the good information (genetic material) from bad individuals can be preserved. The name comes from the basic algorithm that can be presented easily as a roulette wheel as depicted in Figure 4.4.



**Figure 4.4:** Roulette Wheel Selection

If we denote  $n$  as the number of individuals in one generation,  $x_i$  as the  $i$ th individual of that generation, and  $f(x_i)$  as the fitness value of that individual, then we can express the possibility  $p(x_i)$  of the individual to be chosen for reproduction as:

$$p(x_i) = \frac{f(x_i)}{\sum_{j=1}^n f(x_j)} \quad (4.1)$$

If the generation has 5 individuals, we fix one *selection point* and rotate the wheel 5 times. The individuals that have the larger fitness value, thus covering a greater surface of the wheel, will have a better chance of being selected. This is a system that relies on probability, a random spin of the wheel, and it is sometimes hard to comprehend its reliability. But after a huge number of experiments, it is clear that the system of chance works and selects individuals exactly proportionately to their fitness value. To be clear, at the end of the selection process, it means that some individuals will be chosen several times, and some will not be chosen at all. In that way good solutions are multiplied and the survival of their genetic information is ensured.

### 4.1.3 Crossover

After the selection process is done, the selected individuals are ready for reproduction. Some of them will just be copied into the next, new generation, and some of them are *crossed*. The *crossing* of individuals is performed in order to expedite convergence and diversity of solutions. It is an act of combining chromosomes from two *parents* in order to create two *children*. This process has several phases.

The pairs for breeding are initially randomly chosen, from the selected generation. This is one of the reasons the number of individuals in a generation should be even, so that they can all pair up. When the pairs are chosen, the *crossover probability factor* determines their chance of being bred, i.e., crossed. If the parents are not crossed, they are simply copied into the next generation. The crossover probability factor has to be well tuned in order to enable pure replication of good solutions on one side and still allow the exploration of the search space by combining them on the other. That always depends on the problem, and experience has shown that the best factor is around 0.6, meaning that 60% of the individuals in a newly created generation are crossed in order to produce new, different solutions, and 40% of the individuals are just copied in order to support the survival of the fittest.

Crossover can be done in many ways, and that choice can have some effect on the convergence. Namely, Genetic Algorithms are successful precisely because of the fact that they converge to a global optimum instead of being a simple random walk. One of the factors that enable convergence is the crossing process that produces children with the genetic material of their parents. In that way, fitter parents will most likely (but not always) produce a fit child. By explaining the possible methods, the principle will

be clearer. Crossing is performed on chromosomes, which fully represent the individuals, as their *genotype*.

### One point crossover

In Genetic algorithms, one point crossover is the most ubiquitous crossover method. A randomly defined position along the chromosome string  $k$  is introduced. It is always between 1 and the string length, leaving minimum one gene on the left or on the right of it. Two new chromosomes are then created by swapping all the characters between positions  $k + 1$  and  $l$ , as shown in Figure 4.5. Now, it is clear that the first part of the chromosome is not changed, therefore enabling genetic information of the parent to be passed on to the child, and the changed part enables diversity and exploration of the search space. This method is extremely suitable for binary coding of the chromosomes, since the position of the gene (*locus*) plays an important role. The coding of the chromosome in GAs is usually made in such a way that the influence of the gene is the greatest in the first locus and diminishes the further it moves away from it. Therefore an *inheritance factor* is big with this type of crossover and typical GAs binary coding. That means that the children will be very similar to their parents, and it is mentioned several times that this similarity is always a trade off between fast convergence and a good exploration of the search space [16].

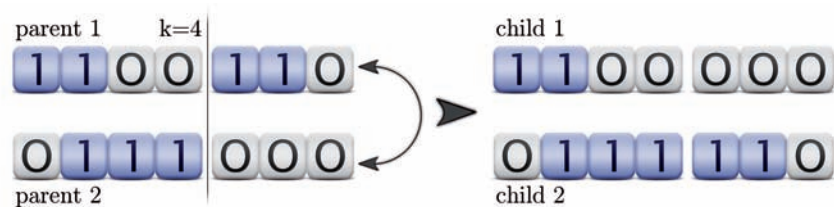


Figure 4.5: One Point Crossover

### Two and more point's crossover

Respectively two (or more) point crossover has two or more division points where the chromosome is divided into parts. The process differs from one point crossover only by the number of chromosome parts that are exchanged. By multiplying those parts we eventually come to a situation where every gene in one chromosome can be exchanged with another one. Of course, if we exchange all of them that would only lead to swapping the parents. That is why new probability factor is introduced, to control that swapping, which leads to the so-called *uniform crossover*.



## Uniform crossover

This is a method more ubiquitous in the Evolution strategies. There, a similar version called *discrete recombination* is used. In our research, we don't use GAs standard binary coding. Instead, we use real-valued chromosome, borrowed from ESs, and therefore a uniform crossover is much more suitable. Why real-valued chromosome is used and how it is applied will be explained in Chapter 4.2. For now it is enough to mention that in the binary coding the *locus* of the gene, i.e., its position is very important. A binary coded chromosome 000111 is therefore very different from 111000 in a standard GAs coding. In the real-valued chromosome used in research, position of the gene is irrelevant, and a chromosome 3-14-35-45 is the same as 35-13-45-3. This is not a general rule, since the position can be accounted and have influence on the individual. But for the purposes of this research, that additional effort was not needed. The crossing method used in the research is therefore shown in Figure 4.6. Each allele of both parents is swapped with the corresponding allele of the other parent with a probability factor of 0.5. In that way each parent allele has a 50% chance of being replaced with the one from the other parent. The process is repeated twice for the generation of each child. The probability factor can be changed and it is inversely proportional to the *inheritance factor*, i.e., the smaller it is, the larger part of the observed parent will simply be copied, i.e., inherited.

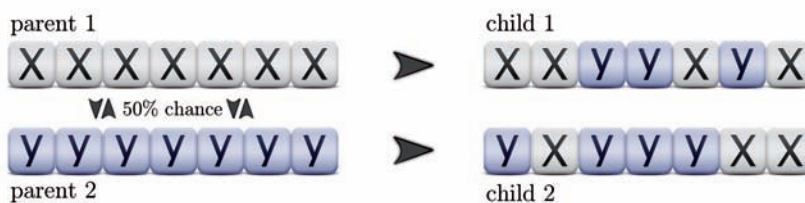


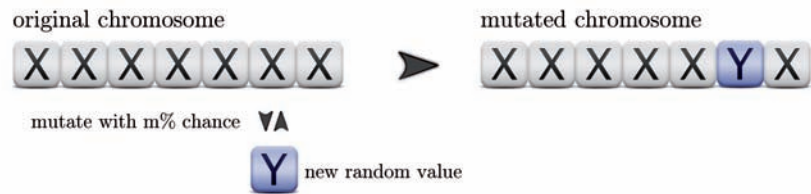
Figure 4.6: Uniform Crossover

### 4.1.4 Mutation

Sometimes the diversity enabled by the crossing of individuals is not enough to explore the whole search space, and the optimization process is in danger of easily converging to some local minimum or maximum. In order to strengthen the diversity of the population, an additional factor is introduced, known as *mutation*. It is just a simple imitation of the mutation that happens in Nature all the time, with a relatively small probability.

Mutation is a change, or an *error* in a genome, whose role in natural genetics is the cause of some confusion when it comes to reasons of its existence. The best explanation is the introduction of

diversity, in search of the optimal design. In artificial genetics it has a *secondary role*, and it represents a random alteration of the single chromosome alleles. We implement the method by changing some random gene value, replacing it with a random generated value. The chances of mutation happening are regulated with the *mutation probability factor*. It is usually very small, around 0.01, thus leaving a 1% chance for some gene to be changed, and it should be appropriately tuned, depending on the length of the chromosome and the size of the population. The simple mutation process is depicted in Figure 4.7, where it can be seen how one gene value in a chromosome is randomly picked and then switched with a randomly generated value to result in a mutated chromosome. The



**Figure 4.7:** Mutation

mutation probability factor, together with the crossover probability factor, can be used to control the convergence and search space exploration, and experience can lead to the best tuning for a specific problem. The tunings mostly used in the research are shown in Appendix A.

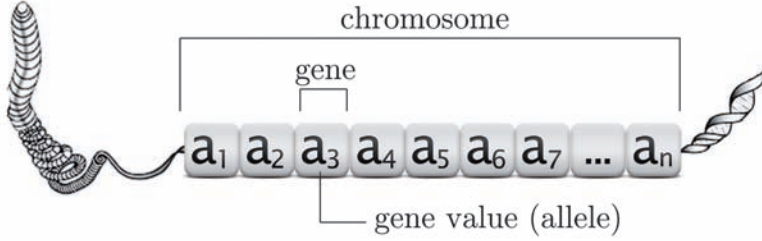
## 4.2 Grid Shell Genotype and Phenotype

### 4.2.1 Chromosome

Humans are *diploid* organisms, having always two homologous copies of the same chromosome. That, considering quick adaptation of organisms to environmental conditions, is the characteristic that can lead to many advantages in life. However, in structural optimization, there is no need for that kind of complexity for now, and in this research all the individuals, i.e., grid shells, are represented with a single chromosome, therefore acting as *haploid* organisms.

In artificial genetics, the chromosome is constructed as a string, an array of numbers. In Genetic Algorithms, a binary representation of individuals is normally assumed, building the chromosomes only out of 0s and 1s. But this is the point where we use the chromosome construction typical for Evolution Strategies and represent them as an array of real numbers  $A = \{a_1, a_2, \dots, a_n\}$ , as shown in Figure 4.8. For  $i = \{1, \dots, n\}$ , each gene  $a_i$  of the chromosome has a value in a specific range  $x = [0, m]$ . Here,  $n$  represents the size

of the chromosome and  $x$  the range of the gene, where the upper limit,  $m$ , is in our examples set to 1.



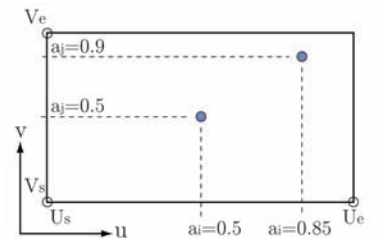
**Figure 4.8:** Chromosome

There are possibilities to represent a grid shell structure with the binary chromosome. In the application of GAs for the optimization of truss structures that is mainly done with the use of incident matrices [56]. However, for our purposes, a more convenient way is to use real number alleles and in that way *borrow* the principle from the Evolution Strategies.

Setting a real number value for every gene raises the question of their range and the method of their application. This is where the parametric representation of NURBS surfaces shows its advantages. In order to represent a grid shell as a function of  $u$  and  $v$  parameters of the surface, the logical thing to do is to use the chromosome as a string of these parameters. However, the domain of NURBS surfaces, i.e., their  $UV$  domain, can differ according to the method of their design. So, to make the method generally applicable, a value between 0 and 1 is set for each allele of the chromosome. Taking two alleles  $a_i$  and  $a_j$  from a chromosome  $A = \{a_1, a_2, \dots, a_n\}$ , we can represent any point on the surface over  $u$  and  $v$  parameters, scaling them to a value between 0 and 1, as shown in Figure 4.9 and following equations:

$$u = a_i(U_e - U_s) , \quad v = a_j(V_e - V_s) , \quad 0 \leq a_i, a_j \leq 1 \quad (4.2)$$

where  $U_s, U_e$  and  $V_s, V_e$  are the surface start and end domains. With the use of different *decoding functions* the numbers in a chromosome are used as  $u$  and  $v$  parameters to generate points on the surface. This points are later used for the construction of Voronoi diagrams. In this research the size of the chromosome was always fixed. That means that the number of generated points on the surface is the same for every individual in one optimization process. The purpose of this was to enable easy comparison between grid shells, since the solutions had relatively the same number of structural members. This can be seen as the limitation of the process, but the software can be easily changed to work with different chromosome sizes. For the purpose of this research, which is to prove the efficiency of the proposed method, this was not necessary, but it is one of the possibilities that will be investigated in the future.



**Figure 4.9:**  $UV$  coordinates on the NURBS surface

### 4.2.2 Decoding Functions

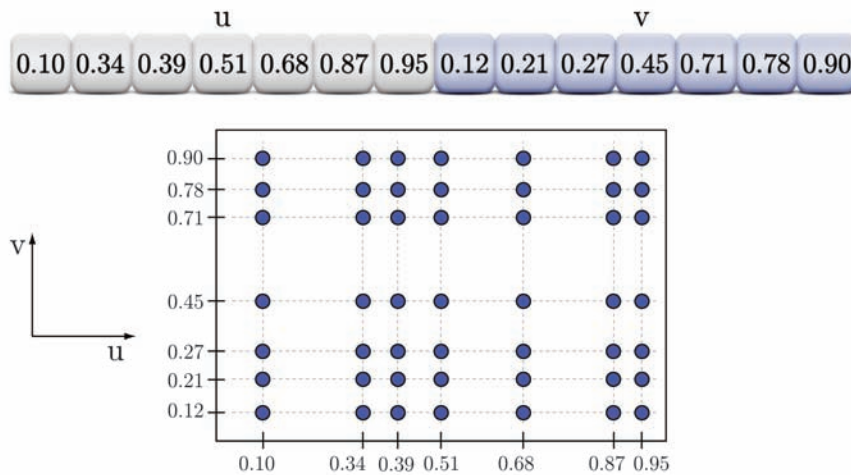
In Genetic Algorithms, the process of creating a chromosome representation is called *coding*, since any kind of individual solution is represented in the form of a code. It was shown how the system of coding basically refers to the creation of arrays of real numbers, the chromosomes. And those are only numbers until some decoding process is performed, thus making *phenotype* out of the *genotype*. In order to create an individual and eventually evaluate it, we have to read the chromosome and use a set of defined rules to make an *organism* out of it.

The decoding function is something that can be very creatively approached and it therefore has an immense number of possible solutions. It represents a bridge between chromosome and an individual and it is a set of rules regulating that transformation. Since the chromosome is a simplification of an organism, so it can make the inheritance of information and breeding relatively simple, it is decoding that holds the key to what will come out of it, and it is decoding that decides what kind of an organism we will have at the end. In our case, decoding transforms a chromosome string into a grid shell. In this research, the focus is on the grid shells generated with the help of Voronoi diagrams, and in that sense, the decoding process can be divided into two main parts: 1. From chromosome to Voronoi seed and 2. From Voronoi seed to grid shell.

#### From Chromosome to Voronoi Seed

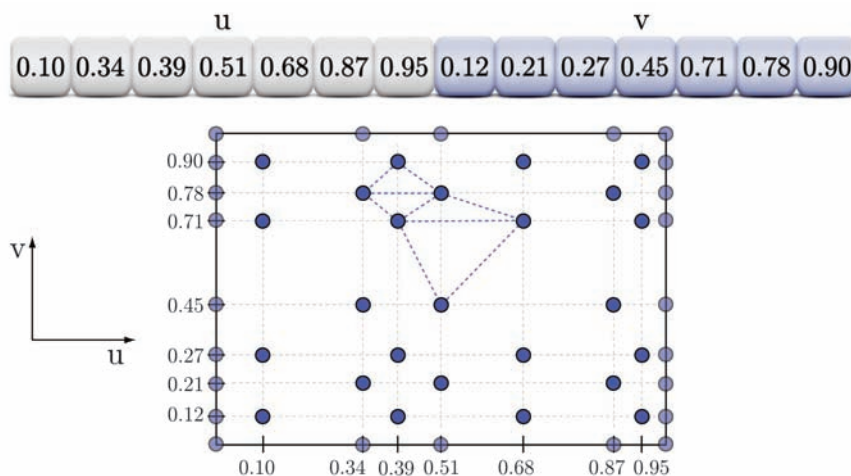
The first step is to convert chromosome alleles into Voronoi seed scattered on the surface. This can be done in a number of ways, producing different kinds of grids. Some of the decoding functions applied in the research will now be presented.

**Quadrangular decoding** It is possible to generate a quadrangular grid shell over any NURBS surface by *planting* Voronoi seed in the center of every quad. Figure 4.10 shows the creation of 49 points out of the chromosome depicted in the upper part. The input data needed to do the decoding is the size of the chromosome and the division point, regulating the number of seed in  $u$  and  $v$  direction. With that information the chromosome can be easily read and developed into a group of Voronoi seed on the surface, as shown in Figure 4.10. By dividing the chromosome in two sections, we can make a rectangular net, thus enabling a grid shell structure with 91 structural members and 96 points to be represented with a small chromosome with only 14 genes. The creation of the grid shell from Voronoi seed will be explained in the next step.



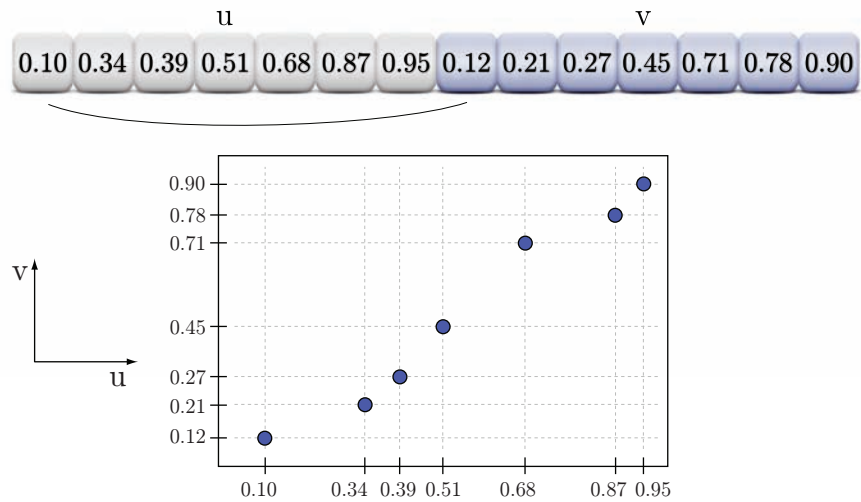
**Figure 4.10:** Quadrangular Voronoi Disposition

**Triangular and Hexagonal Decoding** In Figure 4.11 we can see how the same chromosome from the previous example can be used to develop a different pattern. A blue dashed line is used to indicate the triangular disposition of Voronoi seed in this case. What is interesting is that the same disposition of points can be used to create triangular and hexagonal meshes only by making different choices in the second step, i.e., choices between Delaunay triangulation and Voronoi diagram. This is depicted in Figures 4.19 and 4.15, but it will be clearer after the second step is described. However, the partition of the chromosome is more complicated in this case. It can be seen that additional points are added on the edges to ensure that structural members cover the whole area. Like in the quadrangular decoding system, input data consist of the size of the chromosome and the division point regulating the number of seed in the  $u$  and  $v$  direction.



**Figure 4.11:** Triangular Voronoi Disposition

**Voronoi and Voronax Decoding** Voronoi and Voronax share the same decoding system where the partitioning of a chromosome can be done as shown in Figure 4.12. Actually, that is the simplified system, where the chromosome is partitioned to  $u$  and  $v$  coordinates, like in the previous examples. Each Voronoi seed on the surface is represented with two coordinates and therefore needs two separate genes. This system gives us much larger freedom, since every point is independent from the others, but at the cost of computer memory and speed, since the chromosomes have to be much longer.



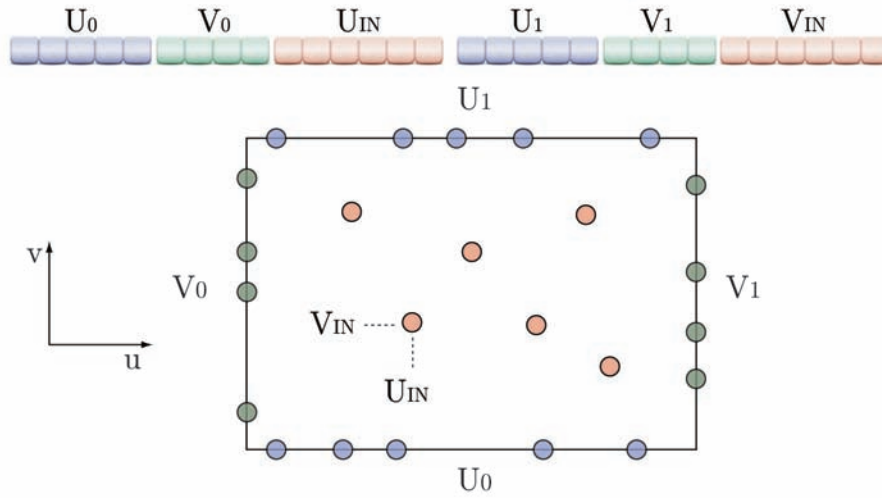
**Figure 4.12:** Irregular Voronoi disposition

It is shown in Appendix A that in order to gain more control of the generated grid shells, some additional information is needed, which requires alteration of the chromosome. Namely, the user has to provide the number of points on the  $U$  edge ( $i$ ), on the  $V$  edge ( $j$ ), and the number of points *within* the surface ( $k$ ). The size of the chromosome structured in that way is  $2(i + j + k)$ , since every point has to be represented with two parameters. This, more complex, chromosome and its decoding are shown in Figure 4.13. It can be seen how it is partitioned into genes that refer to the points on the  $U$  edge ( $U_0, U_1$ ), points on the  $V$  edge ( $V_0, V_1$ ) and points on the surface with their  $u$  coordinates ( $IN_u$ ) and  $v$  coordinates ( $IN_v$ ). Edge points are necessary when the Delaunay triangulation is used, but their use in Voronoi and Voronax structures is also welcomed since in that way we can set a specific number of members on each of the boundary edges.

### From Voronoi Seed to Grid Shell

In the second step, every combination of seed can be turned into a grid shell in three different ways, resulting in different types of grid shells. We can easily calculate the Voronoi diagram using the points



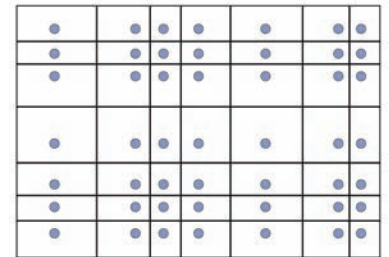


**Figure 4.13:** Irregular Voronoi disposition with edge points

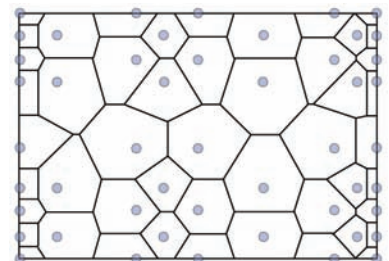
from the first step with Fortune's 2D algorithm since all points on the surface are defined with two  $uv$  coordinates. But we can also go further and relax the Voronoi structure to gain Voronax. The third possibility is to go with the Voronoi diagram's dual graph - Delaunay triangulation, and it will be shown how the same set of points on the surface can be used to generate those three different types of grid shells.

**Voronoi Diagram** Converting a set of points into a Voronoi diagram is described in Section 3.2. Therefore Voronoi seed gained from the quadrangular decoding function, shown in Figure 4.10, yield a grid shell presented in Figure 4.14. Also, the Voronoi seed used in Figure 4.11 can be transformed into a hexagonal grid shell by applying the Voronoi diagram algorithm, as shown in Figure 4.15. In the aforementioned Figure 3.11, in the section where the Voronoi diagram properties are presented, we have seen an example of a regular hexagonal grid. Here however, a deliberately distorted hexagon grid is shown to illustrate the idea of diversity that can be achieved with Voronoi in a polygon structure. This diversity is necessary in the search of the best solution. One of the strong points that can be extracted further from this figure is the senselessness of restricting ourselves to one type of polygon in a pattern. Voronoi and Voronax structures represent the liberation from that uniform thinking, and allow a huge variety of polygons to be combined in one grid. Eventually, in Figure 4.16, a *random* Voronoi structure can be seen, achieved with the application of Voronoi calculation in this second step.

**Voronax Diagram** The same set of points converted into a Voronoi diagram can be *relaxed* as shown in Section 3.3. Therefore by applying this type of decoding, in the second part of the process,

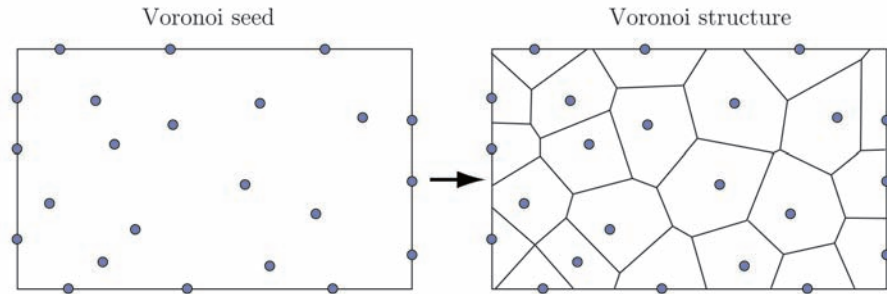


**Figure 4.14:** Quadrangular grid

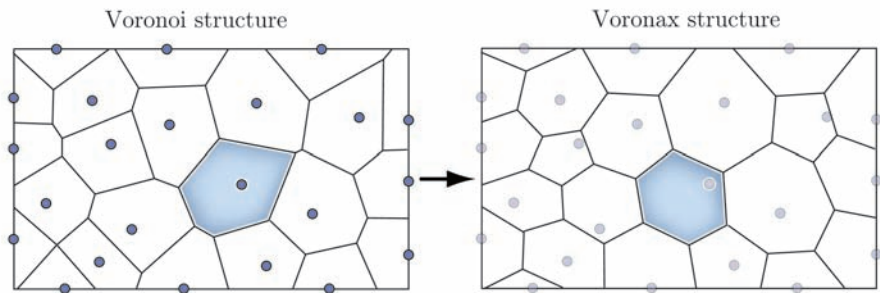


**Figure 4.15:** Hexagonal grid

instead of Voronoi derived grid shells in Figure 4.16 we can get *relaxed*, Voronax grid shells, as shown in Figure 4.17.



**Figure 4.16:** Voronoi structure

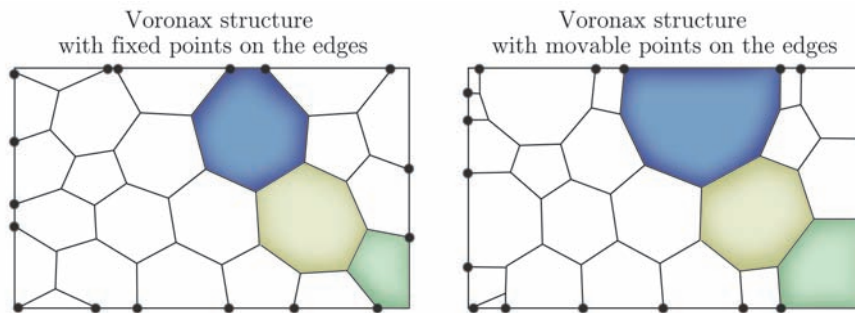


**Figure 4.17:** Voronax structure

It was mentioned that there is a lot of creative freedom in the use of relaxation techniques. On the example of Shanghai EXPO it was demonstrated how the manipulation of tension factors can redistribute grid density to fit some of the statical or optical conditions. It was also explained how the constrained method process works by keeping the points on the surface, or *linking* them to it. That is a matter of choice, and in the applied Force-Density method we can chose to set any point of the structure to be fixed, free in space or linked. If the point is *fixed* it simply doesn't change its position during the whole relaxation process. If it is *free in space* then it can move freely in all three axes during the process. Finally points can be *linked* to any kind of curve or surface (as explained in Section 3.3.2), allowing the point to move only along that curve or surface during the process. Since this research is constrained to predefined shapes, the grid nodes were always linked to the surface or edge curves. A possibility was left for the user to constrain the boundary nodes of the Voronax structure to the edge curves, instead of fixing them. In that way the edge nodes of the structure can move along the surface edges in the relaxation process. In Figure 4.18 it is demonstrated how the same structure looks when it is relaxed with fixed and movable points on the edges. The comparison of the nodes on the edges (marked with black circles)



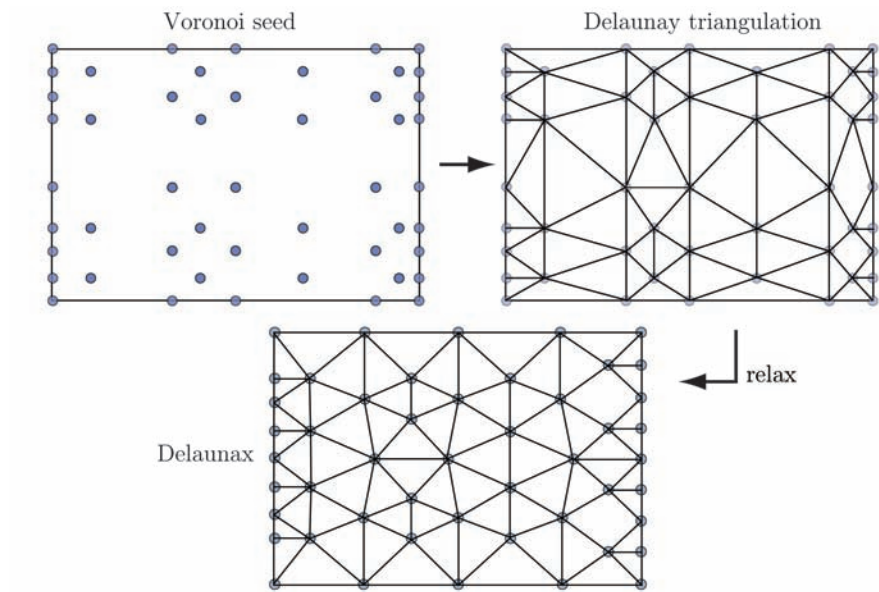
shows us the different disposition of the structural members on the boundaries. Colored fields show how the same polygons look in both versions. One of the main advantages is that structural members on the borders now tend to define the right angle with the surface edge, which is the statically favorable disposition. Other advantages belong to the uniformly distributed angles in polygons and uniform member lengths, what comes from the less restrained relaxation process. For the sake of clarity, all the methods are depicted in 2D, but it is clear that using the parametric nature of NURBS surfaces, everything described can and does happen in 3D, i.e., over some spatial free form surfaces. That will be obvious when the results are presented and grid shells are optimized over given free form surfaces. For now, there are a few more steps left in the explanation of the entire method.



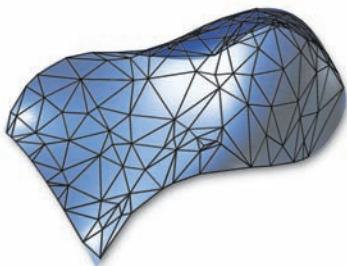
**Figure 4.18:** Differences in Voronax structures

**Delaunay Triangulation** Any set of points in plane, or on a parametrically defined surface in our case, can be connected in a triangulated system as shown in 4.19. In the figure we can see the example where the points from Figure 4.11 are turned into a triangular grid using the Delaunay triangulation method. Some of the characteristics of Delaunay, like the circumcircle condition, are the cause of its irregularity, i.e., of the switching between vertical and horizontal lines, but that can be useful for the diversity of triangulated grid solutions. Back in Figure 3.11, a triangular grid generated by a Voronoi diagram (not Delaunay) was showed, but it makes more sense to use Delaunay for automatic generation. One way to improve Delaunay triangulated grid is to relax it. In the same way that Voronoi can be relaxed to create Voronax, in the lower part of Figure 4.19 there is an example of the relaxed Delaunay grid, which will be referred to as *Delaunax*. It has more equal triangles, i.e., triangles with similar edge lengths and similar angles.

Even if the straightforward creation of triangular grids in this way looks ideal for the optimization of triangulated grid shells, the optimization algorithm with Delaunay and Delaunax triangulation



**Figure 4.19:** Delaunay triangulation and Delaunax grid



**Figure 4.20:** An example of a statically efficient but optically unacceptable and *uninterpretable* solution of an optimization process done with the Delaunay decoding function

often converges to a solution that is optically not acceptable and can hardly be interpreted. The problem is that the optimizations cannot be taken for granted. Their intention has to be recognized in order to abstract the principle and apply it according to many other conditions (like aesthetic) that have to be respected in an architectural project. In Section 5.2.2 this will be emphasized once more, and it will be shown how sometimes Voronax pattern can be more efficiently used to abstract the principle from the optimal result that the Genetic Algorithms offer. In the future more research has to be done to get an optically acceptable grid with the statical optimization of triangular grid shells working with the Delaunay (and Delaunax) decoding directly. Figure 4.20 illustrates a grid shell generated over a free form surface that shows one of the effective solutions generated by the Genetic Algorithms with the use of Delaunay decoding function. The grid is optically unacceptable and cannot be used to extract some principle that can be used to design a statically effective triangulated grid shell.

### 4.2.3 FEM Setup

When the grid shell's geometrical structure is defined and fixed, it has to be prepared for statical analysis and evaluation. For that to be possible, a set of physical properties has to be defined, in order to transform the initial linear representation into a spatial steel structure.

## Material and Section

This research is restricted to steel structures, and the material is therefore not taken as a design variable but as a constant. However, the purpose of the whole system is to be easily expanded and the material can easily be changed for any optimization process. Hence, with very little effort, the material can become another design variable, if its choice is not definite at the beginning. Material constants used in optimization processes are the standard ones for steel. Young's modulus was set to  $E = 210GPa$  with the standard Poisson's ratio of 0.3 and specific weight of  $78.5KN/m^3$ .

For most experiments, cross-sections of structural members were fixed and equal for all the members. The Dialog in Appendix A shows that different types of sections can be defined. In the research, that choice was always made in a way that the material is exploited enough without going over the yield limits. That was estimated according to the load combinations, span of the whole structure and number of structural members, i.e., average length of one member. The choice of making the member sections fixed was also made due to the fact that the main point of the research is geometrical and topological optimization. The choice of different sections also does not affect the solution in great manner, as long as all member sections are the same. This was confirmed with several experiments where the sections were changed but the results of the optimization processes were the same. Nevertheless, the method can be very easily expanded by inclusion of different cross-sections as design variables. In that way, we are multiplying the size of the search space with the number of possible cross-sections. It is certainly something that will be implemented in future research, but for the successful proof of efficiency of the entire method described it wasn't necessary.

In modern practice, the most common section used in free form grid shell design is a hollow rectangular one, mostly composed of 4 welded plates. The hollow rectangle section is used in the research, with different sizes, chosen according to how the load, span and number of elements, i.e., maximal stresses affect them. In Figure 4.21, there is a list of section types, with their properties, that were mainly used in the research, although there were also a lot of experiments done with I-shaped and circular profiles. Again, it has to be emphasized that experience showed that the choice of cross-section does not play an important role in geometrical and topological optimization, as long as they are all set to be equal. If they all vary, that is a different story, with an enormously enlarged search space.

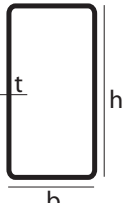
Steel Rectangular	nr	h (mm)	b (mm)	t (mm)	A (m <sup>2</sup> )	I <sub>y</sub> (m <sup>4</sup> )	I <sub>z</sub> (m <sup>4</sup> )	J (m <sup>4</sup> )
	1	200	70	10	0.005	2.23E-005	3.84E-006	1.03E-005
	2	170	70	10	0.0042	1.45E-005	3.29E-006	8.37E-006
	3	140	70	10	0.0038	8.80E-006	2.75E-006	6.40E-006
	4	110	70	10	0.0032	4.72E-006	2.20E-006	4.50E-006
	5	70	70	10	0.0024	1.48E-006	1.48E-006	2.16E-006
	...	...	...	...	...	...	...	...

Figure 4.21: Sections

### Support combination

For static analysis, certain nodes in a grid shell have to be restrained from moving. The user can therefore define a specific *combination* of support, a set of functions that define which nodes will be restrained and the degree of their restraint. In Appendix A it can be seen that the possibility of defining a support combination type exists in the Dialog, and in Figure 4.22, several support combinations used in the research are categorized. The single letters ( $x, y, z$ ) rep-

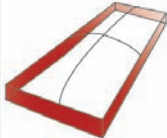
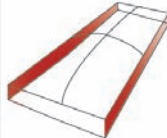
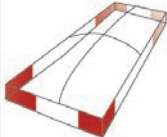
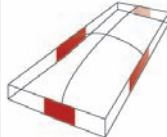
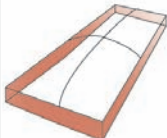
type	abstraction	restriction	type	abstraction	restriction
1		all 4 edges x ● xx● y ● yy● z ● zz●	2		2 edges x ● xx● y ● yy● z ● zz●
3		4 corners x ● xx● y ● yy● z ● zz●	4		4 middle points x ● xx● y ● yy● z ● zz●
5		all 4 edges x ● xx○ y ● yy○ z ● zz○	...	...	...

Figure 4.22: Support combinations

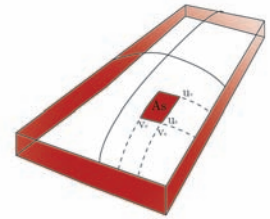
resent the movement, i.e., the restriction of the movement along those axes in space, and the double letters ( $xx, yy, zz$ ) represent the restriction of the rotation around those axes. As it can be seen, most of the types restrict the nodes completely, but that can easily be changed. The table illustrates the diversity that can be achieved with the arrangement of the supported areas, but new definitions of support combinations can be easily imported with new function definitions, according to the specific project requirements.

In the experiments, the selection of nodes that will be restrained was mainly limited to the ones on the edge of the surface (type 1),

since that is mainly the case in practice. Besides that one, several other support combinations are examined, like the ones where only the nodes on the  $V$  side or on the  $U$  side of a surface are restrained, or only the corners are fixed, etc. as it can be seen in Figure 4.22.

In practice, it often happens that the roof structure has columns or walls at specific points of the surface that it can use for support. Algorithmically, that can be solved with the definition of some area ( $A_s$ ) on the surface, over its  $u$  and  $v$  parameters, where the structure can be supported as it is depicted in Figure 4.23. Should a node in a generated solution fall into that area it will be automatically restrained, with a predefined degree. Solutions with additional support will then be favorable since they will have a better fitness. That induces a *magnetic* effect, and the optimization process will be pulled toward the area of search space with that type of solution.

Apart from the fact that we can choose which nodes will be restrained, the degree of restraint is of great importance as well. It is possible to neutralize the movement and the rotation of each node in all  $(x, y, z)$  directions, but also to partially restrain them in one or two directions. This can change the results significantly, since the distribution of forces in the entire structure can be shifted. There are a number of reasons why this often has to be considered in practice, sometimes only for the sake of optimization, as playing with different restraint types and force distributions can lead to a better solution. But sometimes it is allowed to transfer only part of the forces to some supporting columns or walls, as it was the case in the roof above the Great Court in the British Museum designed by Foster & Partners (Figure 4.24).



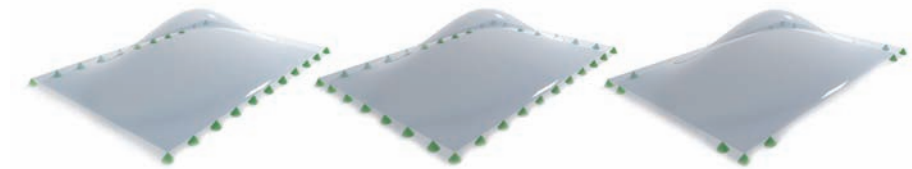
**Figure 4.23:** Support combination with the column area in the middle of the surface



**Figure 4.24:** Roof over the Great Court of the British Museum, London, Foster+Partners, 2000.

On the perimeter (marked green), the building could take the vertical forces from the roof structure, but not the horizontal ones. Therefore, a steel girder was developed along the edges of the Great Court, which was *movable*, i.e., wasn't restrained in the  $x$  and  $y$  directions. In that way only the vertical loads were transferred and the grid shell had to take over the additional stress. Support type 5, for example, shown in Figure 4.22, was made in such a way that

the nodes are only restrained along the vertical direction (rotations around  $x$ ,  $y$  and  $z$  axis are allowed) therefore letting the structure take over moments and horizontal forces, simulating a similar situation as in the British Museum. The results of the experiments with that support type will be shown in the next chapter, and we will see how the choice of joint restrain yields different solutions.



**Figure 4.25:** Different support combinations

The implementation of the restrictions is relatively easy. The presented algorithm works by evaluating each joint in the structure and determining its parametric position on the NURBS surface. If the joint falls into the area defined to be restrained by the support functions, it will be restrained according to the support type that is selected.

### Loading Combination

Combination of loads acting upon a structure can also be defined by the user. Dead and live loads can affect a structure with different force magnitude, direction and different spatial distribution. Therefore a set of functions is applied that can define the load type. There is an infinite number of load combinations for which a structure can be tested. In engineering practice, experience leads to the right choice of unfavorable combinations, thus trying to test the structure for the worst possible scenarios. That is always strongly linked not only to the structure itself, but to the environmental conditions too. Several different load combinations were made to show how an optimization process behaves according to them, and to prove that it is logically influenced by them. In that way, we have proof that this part of the process works, and that it can always be easily expanded, i.e., branched, resulting in more complicated and more sophisticated input data. The precision of the geometrical distribution of load, its magnitude or type can always be improved. In the research, several load types like gravity, snow and horizontal load as well as their combination were used, as it will be seen when the results of the experiments are presented.

**Gravity Loading** Applying a load automatically to some generated grid shell structure can be a complex task. The gravitational load in grid shell structures usually takes the weight of structural members (steel in our case) and the cover material. For covering, as



most ubiquitous in roof structures, glass is chosen with the weight of  $0.6KN/m^2$ , which corresponds to double glazing with two 12mm glass plates.

When grid structures are generated automatically, especially irregular ones like Voronoi or Voronax, all the polygons have to be recognized somehow in order to simulate their glazing and apply the loads on the surface. The surface load is then transferred to the joints, according to the shape of each polygon and its centroid as depicted in Figure 4.26. More about the details of different load combinations will be shown in Section 5.3.

The Fortune's Sweep Line Algorithm used to generate our structures does not provide cell information and therefore a separate algorithm had to be developed to *recognize* each structural polygon, so that the loads can be appropriately applied. This turned out to be an interesting problem, the solution to which is described in Appendix C, as its explanation would be out of context here and probably a confusing digression.

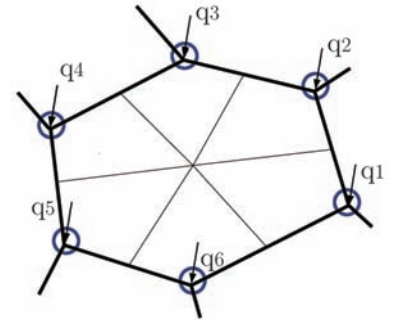
Nevertheless, after the algorithm gathers all the information needed, the weight of the steel members and glass surfaces is calculated automatically and additional load is added according to the chosen load type. The surface of the Voronax polygons is always approximated, since they are usually non-flat, i.e., double-curved. Some of the load types introduce nodal loading, but mostly they are restrained to standard vertical (gravity, snow) and horizontal loads. Loading combinations are something that always has to be tested in a number of different ways according to the specific characteristic of the structure and its environment. Hence, the software is open for an easy upgrade of any kind of load combination, and the results of the ones used in experiments will show how the optimization process converges differently according to the load type, magnitude and direction.

## Evaluation

After the geometry of the grid shell is defined and prepared for static analysis, its evaluation can be made. Evaluation in GAs is done with the help of *fitness functions*. They are a set of instructions used to estimate an individual solution. The composition of a fitness function is the actual process of goal definition and it decides what the objective of the whole optimization process is. As it is the most important part of GAs, a thorough description follows.

## 4.3 Fitness Functions

What is the *value* of a grid shell structure? Which of several solutions is *better*? The answers to those questions depend on the



**Figure 4.26:** Surface load is divided and transferred to joints, punctually

perspective, i.e., on the criteria chosen to evaluate a given structure. One specifically defined criterion leads to a single-objective optimization. As the name suggests, each solution in the process gets evaluated according to one aspect, which can be a function of several different variables. An objective is then usually expressed in terms of minimization or maximization of that value. Single-objective optimization will be addressed in Sections 4.3.1 and 4.3.2. Sometimes the solution has to be evaluated according to several criteria. In those cases, different objectives are set, which very often collide with each other. That leads to a multi-objective optimization, common in structural engineering. How multi-objective optimization is applied will be shown in Section 4.5. One more, very important subject also has to be discussed, as it is inevitable in structural design. Namely, in structural optimization, single-objective optimization alone does not make much sense, as there are always lots of restrictions that have to be taken into account, such as maximum stress allowed or maximal cross section of the member, etc. Those restrictions in GAs come in the form of *penalty functions* and they will be explained in detail in Section 4.4.

Fitness functions in our grid shell optimization are divided into optical and statical ones. The first, smaller, group of functions is made to satisfy geometrical conditions, like lengths of the members, or size of the cells. The latter one is represented by functions which evaluate each grid shell according to the results of the FEM static analysis.

### 4.3.1 Geometrical (Optical) Functions

Here, the functions that mainly take the geometry of the grid into consideration will be discussed. The geometry is important for the architectural expression but it also has to satisfy conditions imposed by fabrication possibilities. That is why the functions that are trying to optimize a structure according to different geometrical conditions and restrictions can be defined.

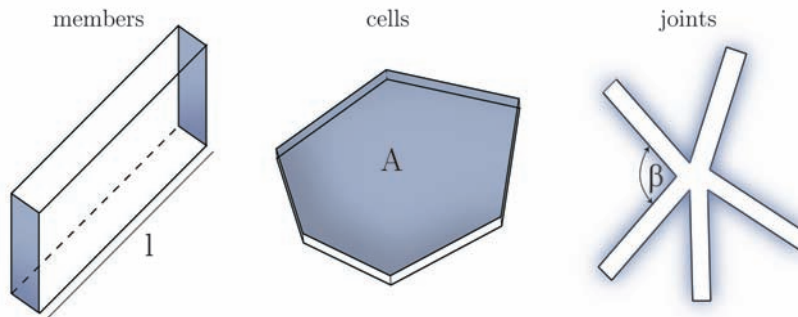
Every structural design is a *simplification* of an architectural form, trying to combine the shape and supporting elements, thus enabling the whole object to resist forces acting upon it. In that manner, a free form grid shell is a *discretization* of a surface into finite elements, i.e., structural members. Some variables that are chosen to participate in the optimization will first be presented and then one of the optical fitness functions will be explained in detail.

#### Possible Variables

There is always great freedom in the definition of the geometrical properties of a structure that we want to be observe and evaluate. Creativity can come from different project requirements. For now,



some of the standard geometrical characteristics will be addressed, since their use is sufficient to show how the method works and to prove its validity.



**Figure 4.27:** Grid shell structural characteristics

**Structural Members** The structural members of a grid shell can, and in free form structures mostly do, have different lengths and cross sections. Great differences between them often leave an unpleasant visual impression and are therefore something that has been mainly avoided so far. A Genetic Algorithms based tool can set some constraints, or even define the whole fitness functions to evaluate different optical aspects of the structure. It can be a perfect guide, by proposing a number of solutions that can be used for further processing.

Two main characteristics of a structural member are its length and its section. A huge number of functions can be defined, that regulate their size and proportion. For example, if cross sections are one of the design variables, we can use a function that would allow the connection of two members only when their sections are *connectible*, i.e., when the difference in their shape and size is in predefined boundaries. Much more interesting for the whole structure are the lengths. Namely, we can easily set constraints in form of a penalty function and define the minimum and maximum member length allowed. The final solution will then be within those boundaries, as it will be shown with the explanation of penalty functions and in the results of the experiments where those restrictions were used. Similarly, the relationship between member sizes can be regulated as we will see in the *Average Length Deviation* fitness function.

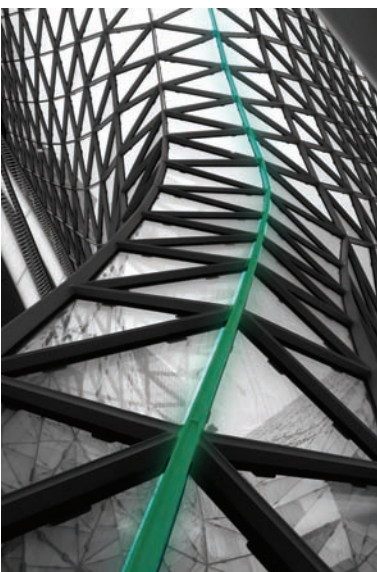
**Cells** In Appendix C a method, developed in this research, is described that recognizes all the polygons in an irregular structure, such as Voronoi or Voronax. That information can be used to set fitness functions or penalty functions to control those polygons. One of the first and most obvious applications is to set the limitation

on the polygon size, i.e., cell's area. We can easily define the minimal or maximal surface area that one polygon can have, and that is one of the main penalty functions used in the research experiments. However, more sophisticated evaluations can be used to set objectives or restrictions. The number of sides in polygons, maximal or minimal angles, are also something that can be controlled. There is always the possibility of combining all those characteristics into some complex functions if the project requires it.

**Joints** The number of members connecting in one joint can differ from 3 to  $n$  in a grid shell. With triangular, quadrangular, hexagonal, etc. structures, where the number of connections in one joint is always the same, joints that don't fit the pattern are quickly noticed. The human eye has a remarkable ability to recognize patterns and a single deviation from an established pattern is always a problem. Sometimes, due to design and static problems, joints like that are not avoidable. The shape of the joints can therefore be controlled with functions not only according to the number of members they connect, but also according to the angles at which they connect. Angles between the members in a joint are a huge factor for an optically acceptable design. Additionally, if the sections differ, complex functions can be made that restrict combinations of angles and sections that are not physically possible. A set of manufacturing conditions, which are generally strict for joints, can be imposed in that way.

**Guide Lines** Voronoi structures look extremely *non-regular*. One can not extract continuous lines or paths and therefore see such a structure as a collection of cells scattered all over the surface. A similar effect can be achieved with triangular, quadrangular or hexagonal structures when they form a grid with irregular shapes and connections, i.e., when cells have different sizes and angles.

On the other hand, we are used to the *regulated* triangular and quadrangular structures where the network is formed with distinctive *guide lines* or *paths*. In such grid shells, the angles between cells and members become the most important factor, as the smoothness of those paths contributes the most to the structure's visual appearance. In Section 3.3.5, the advantage of *relaxed* grids is shown resulting in greater smoothness. Therefore *kinks* in grid structures can be resolved with the application of the Force-Density method or with complicated constraints, controlling the *paths* formed. In Figure 4.28 a part of the MyZeil roof structure is shown where it can be clearly seen how the relaxation process kept the guide lines smooth. The constraint control can get very complex and is therefore not thoroughly researched and addressed. Instead, the focus is more on the structures where the smooth *paths* don't exist, and don't present a problem, like Voronoi structures. Nevertheless, the



**Figure 4.28:** MyZeil, Frankfurt, 2008

subject of pattern *beauty*, the rules in its proportions, cell distribution, polygon shape etc. is extremely interesting and deserves more exploration in the future.

### Average Length Deviation - Fitness function

In one of the most common grid shell design approaches, the focus is on trying to keep the shape and size of structural members as similar as possible. Using this approach in the research, a geometrical fitness function that considers the length of structural members is developed. It is however not intended to serve as a main fitness function but as a *shared* one in a multi-objective optimization and, much more important, to be applied as a penalty function constraint as it will be explained in Chapter 4.4. If a certain project only required optical optimization, this would be an easy task, since optical functions are much simpler and work much faster than the statical ones.

The total length of  $n$  members in a grid structure, where  $l_i$  is the single length of the  $i$ th member, can be presented as a function:

$$f^l(l_i) = \sum_{i=0}^n l_i \quad (4.3)$$

Therefore an average member length  $f_{avg}^l(l_i)$  can be calculated as:

$$f_{avg}^l(l_i) = \frac{f^l}{n} = \frac{\sum_{i=0}^n l_i}{n} \quad (4.4)$$

Now, an ideal structure, based on the criteria of the described fitness function, would be the one where all members have lengths equal to  $f_{avg}^l$ . Since that is not possible with free form shapes, the deviation factor  $D$ , that represents the total amount of difference between the member lengths and the average length, can be defined. Respectively, if the grid shell structure is a vector of parameters  $x$ , the fitness function is defined as:

Minimize:

$$f(x) = D = \sum_{i=0}^n |f^l - f_{avg}^l| \quad (4.5)$$

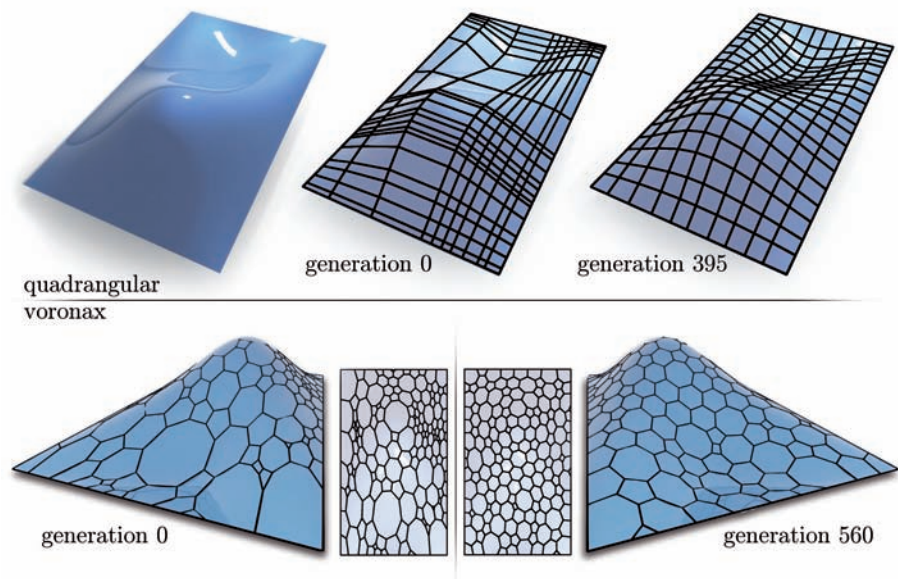
Optical functions are easy to implement and very fast algorithms, since they don't communicate with other programs. An optimization process with 10000 individuals, e.g., 200 generations with 50 individuals in each generation, can be executed in a few seconds.

Although a number of different optical evaluations is possible, the Average Length Deviation is the only one implemented as a fitness function. In Section 4.4 its implementation as a constraint will be addressed as well as some other optical constraints. Primary

functions should be, and in this research are, the ones that evaluate grid shells according to their static eligibility.

### Results

In Figure 4.29 there is an example of an optimization process performed using Average Length Deviation as a single fitness function, without any constraints. Different patterns are explored and the



**Figure 4.29:** Optimization with quadrangular and Voronax pattern, showing the convergence toward the solution in which all members have the same length

difference between one representative solution from the first generation and one from the advanced generations is demonstrated. Each generation consisted of 50 individuals. With the rectangular pattern (up), the restriction of movement of structural members is much greater, and therefore the solution is generally found quickly. With Voronax (down), there are more possibilities and the optimization process needs more generations, but it is clear that it converges toward the good solution, i.e., toward the solution where all members have similar length. Just as a short remark, one should not be confused by different cell sizes in the Voronax example. The fitness function doesn't regulate their size, only the length of the members. So two cells can differ very much in size, but if one has 5 and the other 9 bounding members, they can still have the same length. The graphs and detailed analysis is left out here, because this is only a small introduction for the elaborate analysis of different experiments which follows in the next chapter.

### Geometrical (Optical) Function as a Proof of GAs Efficiency

In order to demonstrate the credibility and efficiency of a GAs optimization process, two assumptions have to be verified. First of all, it has to be shown that, given a fitness function, the process converges toward the best solution, taking that fitness criteria into consideration. Secondly, it has to be proven that the fitness function considered is defined adequately, i.e., that the goal set is the right one.

In optical functions the second requirement is fulfilled, as the results can be seen optically and the conclusion can be drawn easily. If the Average Length Deviation function produces results showed in Figure 4.29 after only 500-600 generations, it can be *seen* that it converges properly. After a number of experiments, with the same convergence results, the conclusion is that the first condition, credibility, is satisfied and that the optimization process does converge to an optimal solution. With static functions, the only concern left then is if the objective functions are good, i.e., do serve the purpose of obtaining lighter and more stable grid shells.

#### 4.3.2 Statical Functions

Since the optical functions have been demonstrated, as well as the effectiveness of the GAs optimization, it is time to discuss the core of the whole research. The core is composed of functions that use FEM static analysis to calculate displacement and stresses in the structure, as well as its stability, i.e., buckling resistance. The results of those functions will be used in our grid shell optimizations over a given free form surface.

#### Objective

It was already mentioned that the distinction between design and optimization doesn't really exist. Every design process follows an optimization philosophy, and similarly, a structural design is by itself always an optimization procedure. At the beginning, the engineer faces a number of different, usually randomly created, solutions and by picking and modifying some of them, a good solution eventually emerges. The evaluation of a structure in every step of that process follows a set of goals and those standard goals will be used to create fitness functions in our research.

A statical fitness function has to define a statical objective that will be followed in the grid shell design. One of the basic structural behaviors in Nature are the minimization of material and the minimization of potential energy. The same objectives are also used in structural design. Trying to minimize the use of material and

overcome bigger spans and heights at the same time, has to be expressed in different terms. Stress generated in the structure by external forces is a first and obvious choice for minimization, since the reduction of stress directly leads to material reduction. Of course, there are lots of other factors that have to be considered, like displacements or buckling of a structure for example. All that can be used to creatively construct a fitness function according to the requirements of the design goal. Some of the basic fitness functions will now be demonstrated and their application will be explained. The results of the different optimization processes, that use the functions described here, will be presented in the next chapter.

### Minimizing Von Mises stresses

If we want to design a structure that can resist all the external and internal forces in the best possible way, we have to reduce stress in the structure, which leads to the effective and economical use of material. A steel beam can resist forces that act axially upon it with much less material than forces that cause bending. This simple fact leads to the conclusion that an arrangement of structural elements should be made in such a way that it minimizes the bending moment, and that is a form of geometrical optimization. If the position of a member can affect the stress induced, first we have to see how to calculate that stress and then how to find the best position.

Given the free form surface, the chromosome, a specific decoding function and all other design parameters, a grid shell is generated over that surface. The support and load combination design variables together with sections and material information lead to linear FEM static analysis of the grid shell. Due to the constrained joints (not pin-jointed like in truss structures) every node has six degrees of freedom, i.e ,transfers three axial forces and three moments. Hence, Figure 4.30 shows a beam element that experiences different distributions of all three forces and three moments throughout its volume.

The beam resists the moments  $M_x, M_y, M_z$  with its moments of inertia  $I_x, I_y, I_z$  and the axial forces with the corresponding cross-sectional surfaces. The trick is to determine the minimal cross-section that has the moment of inertia to resist the largest moments and enough cross-sectional surface to resist the largest axial force. That usually leads to an optimization process known as *sensitivity analysis*. However, in the optimization techniques, the stress that combines all stresses into a single value is used to simplify things. That value is called Von Mises stress and it defines the critical value of the material before it starts to yield. It is therefore also known as the Von Mises yield criterion and it states that although none of the individual stresses are large enough to cause yielding, their

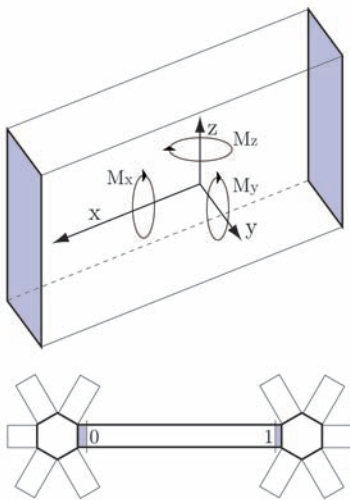


Figure 4.30: Beam Forces



combination can. Yielding therefore occurs when the *elastic limit* is reached, i.e., it is the lowest stress at which permanent (plastic) deformation can be measured.

$$\sigma_v = \sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2 - \sigma_x\sigma_y - \sigma_x\sigma_z - \sigma_y\sigma_z + 3\tau_{xy}^2 + 3\tau_{xz}^2 + 3\tau_{yz}^2} \quad (4.6)$$

The definition of Von Mises stress and its precise calculation can get complex and confusing, which can be seen from its general expression in Equation 4.6. In this research a simpler way to calculate Von Mises stress is applied, and with the intention of making things as clear as possible, only that version will be explained. For steel beam elements, with symmetrical cross-sections, some simplifications are possible when we want to rationalize the approach. Namely, the Von Mises stress depends on the axial forces, bending moments, shear forces and the torsional moment. The shear stresses can be combined through thickness shear stresses  $\tau_1$  and  $\tau_2$  and the axial forces and bending moments are combined to form  $\sigma_{xx}$ :

$$\tau_1 = V_z/A_z \quad , \quad \tau_2 = V_y/A_y \quad (4.7)$$

$$\sigma_{xx} = \left| \frac{N}{A} \pm \frac{M_y}{W_y} \pm \frac{M_z}{W_z} \right| \quad (4.8)$$

The Von Mises stress is then calculated as:

$$\sigma_v = \sqrt{\sigma_{xx}^2 + 3\tau_1^2 + 3\tau_2^2} \quad (4.9)$$

Now, the Von Mises stresses change along the beam, but due to the nature of a grid shell structure and concentration of load in the nodes, the extremes are mainly at the beam ends. Therefore in the research, for each beam,  $\sigma_{v,0}$  and  $\sigma_{v,1}$  are calculated, representing the Von Mises stress in the endpoints of the beam (shown in Figure 4.30 down). The aim of the fitness function is therefore to minimize the total amount of stress and the developed function is represented like this:

Minimize:

$$f(x) = \sum_{i=0}^n [\sigma_{v,i,0} + \sigma_{v,i,1}] \quad (4.10)$$

where  $\sigma_{v,i,0}$  and  $\sigma_{v,i,1}$  represent Von Mises stress in two end points of an  $i$ th beam in the grid shell structure  $x$  with  $n$  beam elements.

For the calculation of Von Mises stress the orientation of the member, i.e., the rotation of its cross-section has to be considered. In Section 5.5 an algorithm that rotates the structural members

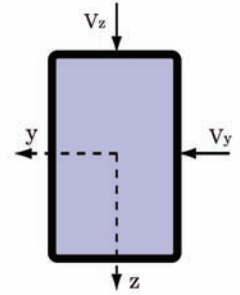
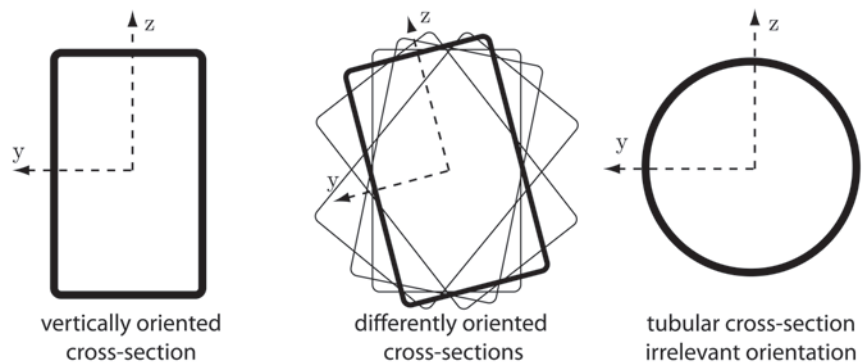


Figure 4.31: Shear forces

to orient them correctly is described. However, in most of the optimizations that was not implemented and they are performed with all members vertically oriented (in the positive  $z$  direction) or with tubular sections where the rotation makes no difference.

This was done to simplify the process. First of all, it was mentioned that the main goal of this research is to propose an efficient method for the design and optimization of free form grid shells. The experiments shown in the following chapters are there to prove that efficiency. Hence, it makes no difference if the members are oriented or not, because the optimization process can offer an optimal solution according to either of those two options. The process is concerned with converging to an optimal solution regardless of the input settings. Therefore the nature of those settings should not and does not affect the efficiency of the optimization process. Additionally, the orientation of members in a Voronax grid (as explained in Section 5.5) cannot be taken for granted because the Voronax cells are not planar, and the rotation of the members is only approximative and the *proper* orientation is arguable.



**Figure 4.32:** Three options for the cross-section of structural members used in the research

Another reason to simplify the process (and save a lot of computing time) was a very small difference in the evaluations of grid shells used in the experiment when their members are oriented or not. In Section 5.5 it is demonstrated that the differences in fitness values are between 0%-3% for the Minimize Von Mises stresses fitness function used in experiments, which basically means that the orientation wouldn't affect the process substantially.

### Deviation From Average Stress

Klaus Mattheck defines the *axiom of uniform stress* like this, “...there can be only one good mechanical design, namely the one in which there are neither weak places (locally high stresses) nor underloaded zones (useless ballast). In the final analysis this means that for a given operating load the stresses must be completely uniform everywhere in the component, i.e., the load is fairly distributed.” [39].



A fitness function based on that idea is developed to compare the results with the Minimize Von Mises Stresses fitness function, to prove the axiom of uniform stress, and therefore strengthen the choice of one solution. Since we defined the Von Mises stress in a beam element in Equation 4.9, then there is an average Von Mises stress in the structure:

$$\sigma_v^{avg} = \frac{\sum_{i=0}^n [\sigma_{v,i,0} + \sigma_{v,i,1}]}{2n} \quad (4.11)$$

where the average stress refers to the stress in one end of the beam. That is why the number of beam elements in the structure  $n$  is doubled.

From there a deviation  $D$  is defined that represents the difference between stresses in one end of each beam element and an average stress. Respectively the fitness function is defined as follows:

Minimize:

$$f(x) = D = \sum_{i=0}^n \sum_{j=0}^1 |\sigma_{i,j} - \sigma_v^{avg}| \quad (4.12)$$

Most of the time, the results are similar to the ones from the Minimization of Von Mises Stress fitness function, as it will be briefly presented in the next chapter.

### Maximize Load Factor

Grid shells belong to the class of lightweight structures. That means that they are able to overcome wide spans with a relatively small dead load. One of the goals of geometry and topology optimization is to enable the structure to carry loads by normal forces mainly, i.e, with an optimal use of material and the cross-sectional surface of the structural member. Since the loads create mainly compression forces in the grid shell, and the members are very slim, one of the greatest problems is their stability and buckling is one of the key factors in such realizations [54]. Even when the stresses in the elements are far below the limit and there is no danger of buckling of the single members, the whole structure can still collapse. Hence, a grid shell shows stability failure similar to slab structures or to continuum shells even when the two modes are combined [4]. In this research, the focus was maintained on *global instability*, but other problems considering local instability and member buckling can easily be examined in the form of similar fitness functions. In this fitness function the effect of geometrical imperfections has also been left out. They play an important role in structural design of shell-like structures since buckling loads are much reduced in many cases due to it [54]. However, imperfections have greater influence

on dome-like structures where the compression forces are high. In this research we are dealing with free form surfaces where the percent of bending induced stress is much bigger than the tension or compression induced stresses. Additionally, the consideration of aspects like this represents an expansion of the method, but it is not necessary to prove its efficiency. The optimizing process can work as well regardless of the input settings, i.e., whether the geometrical imperfections are included in the static analysis or not. The algorithmic expansions of this kind are therefore left for some future research and possible application on specific projects with the requirement that the imperfections are taken into consideration.

The buckling analysis for the whole structure refers to the estimation of a *critical buckling load*, an Eulerian buckling load, at which it becomes unstable and deforms into different *buckled mode shapes*, or *eigenmodes*. So we can use the *automatic load increment analysis* to calculate the *buckling load factor* (BLF)  $\lambda$ , that represents a ratio of the buckling loads and applied loads. To be more specific, its value multiplied by applied loads results in a buckling load, i.e., the load under which the structure becomes unstable. In order to do this, it is assumed that the differential stiffness matrix is a linear function of the applied load. At the point of buckling, the determinant of the sum of the elastic stiffness  $K_e$  and critical differential (geometrical) stiffness  $K_g$  is zero [42].

$$[K_e] + \lambda[K_g] = 0 \tag{4.13}$$

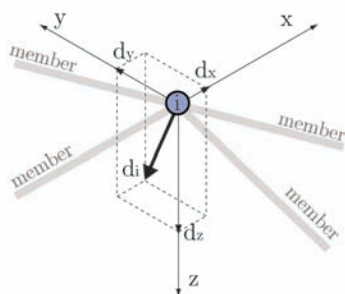
It is then logical that the fitness function should try to maximize  $\lambda$  in order to obtain the most stable solution. However, the Load factor is best used in Penalty functions as will be explained in the next chapter.

### Minimize Displacements

The stability of the structure is strongly connected to its *stiffness*. Namely, the stiffness can be determined according to the displacements of the structural joints. The bigger the displacements are, the weaker the structure is, and the stresses in members, trying to resist those transformations, are higher. From this perspective, in search of the structure with the smallest deflection, another fitness function is developed. If the structure has  $n$  joints, and each joint can move, i.e., be displaced in three directions  $x, y, z$  (Figure 4.33), then the displacement vector  $d_i$  for each joint is calculated as:

$$d_i = \sqrt{x_i^2 + y_i^2 + z_i^2} \tag{4.14}$$

and the fitness function is constructed here as a minimization of the sum of all displacement vectors in the complete grid structure  $x$ :



**Figure 4.33:** Displacement vector

Minimize:

$$f(x) = \sum_{i=0}^n d_i \quad (4.15)$$

### Other Fitness Functions

All of the fitness functions are developed and programmed within the research, although the basic idea of using the minimization of stress, deformation, etc. is a known concept in structural optimization. The four described static fitness functions are the ones used extensively in the research. Each one of them can be extended and altered easily to pursue some different objectives. Some experiments were done with other parameters, like *eigenfrequency* and *modal stiffness*, trying to find an optimal structure considering its dynamic behavior. However, instead of describing each of the used and possible techniques, the results will show that the whole GAs optimization method leads to interesting and important results and that it can be expanded easily to fit any objective. The goal in the research is to perform optimizations with a proper number of different and representative fitness functions in order to prove that the method works. When that is proven, it means that any kind of fitness function, i.e., any kind of input data combination will converge to an optimal solution according to those settings.

#### 4.3.3 Fitness Scaling

Depending on the different fitness functions, different values are obtained, and in a multi-objective optimization, for example, some system has to be made to compare those evaluations. More importantly, they have to be *scaled* in order to enable the selection method to be effective. The reason for scaling, what its usual forms are and how it is applied in the research will now be described.

#### Why Scaling?

Every individual in one generation has a chance of being reproduced according to its own fitness. The better its fitness is, the more chances it has of being reproduced, therefore enabling the *survival of the fittest* principle to be applied. For example, we can imagine that we have a generation of 5 individuals with fitness values: 1100,1200,1300,1400,1500 (without going into the fitness function specifications, since it is not relevant for this explanation). If no scaling was performed, we would add all values and calculate the share of each one in the generation. The percentages, i.e., chances of their survival would then respectively be : 17%, 18.5%, 20%, 21.5%, 23%. If we let the roulette selection method pick the individuals, they would all have more or less the same chance, because

the differences are not substantial. And the bigger the real value of the fitness function is, the smaller the differences become.

The second problem can occur when solutions are extraordinarily good or extraordinarily bad. They can totally disturb the balance and assign chances of survival that do not contribute to *fair competition*. If left to the normal selection rule (Equation 4.1), the extraordinary individuals would take over a significant proportion of the finite population and this is undesirable, a leading cause of premature convergence [16]. Namely, let us consider a generation with 7 individuals with following fitness values: 58,2,3,3,5,6,6. There is one extraordinary big value (58) in comparison to the others, and that individual will have 70% chance of being selected. One can say that is acceptable, since that individual is much better than the others. That is true, but it causes the differences between the other individuals to be very small (2.4%, 3.6%, 3.6%, 6%, 7.2%, 7.2%), more importantly, it *kills* them immediately by selecting only the fittest individual. That is not good, since it leads to *elitist* selection, therefore preventing diversity and, as already mentioned, leading to premature convergence. Experience showed that the introduction of diversity is one of the most important factors in GAs (as well as in Natural Selection), while searching for the global optimum [16]. That is why the thorough exploration of the search space should be supported as much as possible. A similar thing happens when one individual has a very small fitness value and its chances of survival should be enhanced somehow.

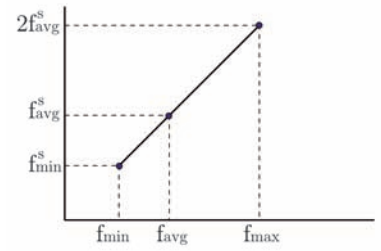
### Standard Fitness Scaling Methods

There are different ways to transform the values obtained from the fitness function to some reasonable numbers, which can be used to determine each individual's chances of survival [47]. Those methods can be constructed in various ways according to the coding method and the range of values coming from functions. For example, one of the simple ones, and not really effective is *linear rank scaling* [3]. The individuals are *ranked* according to their fitness value and then assigned a value that corresponds to their rank. The shortcoming of this method is that it is not precise and it doesn't assign chances proportional to fitness as it should. Additionally it doesn't make the corrections needed to solve the aforementioned problem of extraordinary individuals. There are also other ranking methods [47, 3] which will not be discussed here, since they were not found suitable for the purposes of this research.

One of the standard fitness scaling methods in GAs and most ubiquitous is *linear scaling* [16], that can express the relation between scaled fitness  $f^s$  and raw fitness  $f$  as:

$$f^s = af + b \tag{4.16}$$

The point here is to make average scaled fitness  $f_{avg}^s$  and average raw fitness  $f_{avg}$  equal, and to scale the maximal and minimal fitness to some reasonable value. Usually that value, for maximal fitness, is set to 2, as it can be seen in Figure 4.34. This method has its own advantages and faults, and it usually gets expanded by a number of factors that influence the values of  $a$  and  $b$ , so that the optimization converges properly. However, since this exact method is not used in the research, it will not be explained in detail. For the optimization processes in the presented research a simplified version of fitness scaling was developed, which was totally satisfactory and resulted in an acceptable convergence.



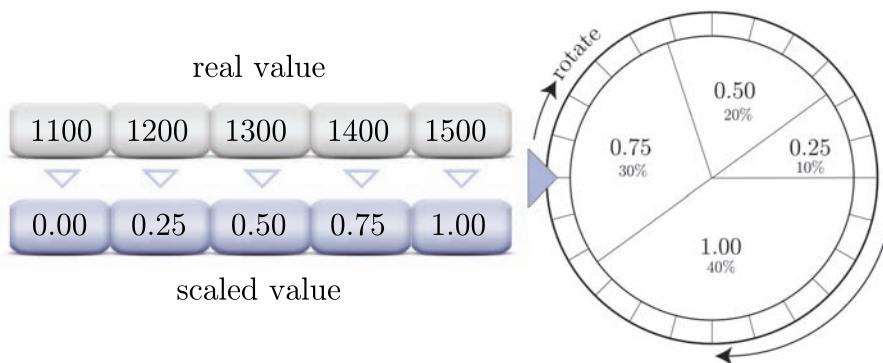
**Figure 4.34:** Linear scaling

### Simplified Method Developed For the Research

To avoid the problem of large real values, the idea was to scale all the solution fitnesses to a value between 0 and 1. If we have a fitness value as a result of fitness function  $f(x_i)$ , for  $n$  individuals in one generation, we can denote the maximal fitness and minimal fitness as  $f_{max}$  and  $f_{min}$ , and the difference between them as  $d = f_{max} - f_{min}$ . In order to transform and calculate the scaled value  $f^s(x_i)$  for each individual, the next operation has to be performed:

$$f^s(x_i) = \begin{cases} \frac{f(x_i) - f_{min}(x_i)}{d} & \text{if } f(x_i) \neq f_{min}(x_i) \\ 0 & \text{otherwise} \end{cases}, \text{ for } i = 0 \text{ to } n \quad (4.17)$$

In this way the worst individual will have value 0, the best one will have 1, and the others will be scaled to values between 0 and 1. With the values from the example above, scaled fitnesses would be as shown in Figure 4.35. The worst has zero chance of being selected for reproduction, and the other individuals are added up for the roulette selection.



**Figure 4.35:** Fitness scaling

This system works when the fittest individual is the one with the largest real value. However in most cases in structural design

we are dealing with minimization and therefore some changes have to be made to invert the process and set the individual with the lowest value to be the best (1), and the individual with the highest value to be the worst (have 0 for a scaled value). This is done in a very simple manner, by subtracting the scaled value from 1:  $f_{inv}^s(x) = 1 - f^s(x)$ . In that way everything is reversed and in the next section, when penalty functions are addressed, it will be explained how fitness scaling is expanded when the solutions are divided into feasible and infeasible ones.

In grid shell design, and with the fitness functions used, there was no need for complex fitness scaling procedures. The first problem of large real values had to be solved, but the second problem of extraordinary individuals could hardly occur, as such large variations in values were rarely possible. Even when they were, the elitist selection that would occur at that point couldn't hurt the exploration of the search space that much, since diversity was supported by a clever adjustment of mutation and crossover probabilities. Fitness scaling, as everything else, can be easily altered and improved, but for the sake of this discussion and the proof of the efficiency of the method, the fitness scaling used was more than satisfactory.

## 4.4 Penalty Functions

Unconstrained optimization is something that can hardly be applied in structural design. Whatever the objective is, there is always a number of constraints that need to be imposed, considering material properties and production capabilities. In Genetic Algorithms constraints are applied in the form of *penalty functions*. The name comes from penalization, introduced into the evaluation system and used to reduce the fitness values of infeasible solutions, proportionally to the degree of the constraint violation. The implementation of penalization can have the form of *equality* and *inequality* constraints [19] and therefore an optimization problem is usually expressed as:

Minimize (or maximize):

$$f(x) , \quad x = (x_1, x_2, \dots, x_D) \in \mathbf{R}^D \quad (4.18)$$

under constraints:

$$g_i(x) \geq 0 \quad , \quad h_i(x) = 0 \quad (4.19)$$

As mentioned before,  $x$  represents a design vector, with design variables of  $D$  structural components, and  $g_i(x)$  and  $h_i(x)$  are the *equality* and *inequality* constraints of an individual solution. In simpler terms,  $x$  is our individual that is always represented as a collection of different structural components, like: number of elements,

number of nodes, material, section type, etc. This information can be predefined or vary during the optimization. Either way, they are the design parameters or variables, or vector of components that together define our individual solution. Equality constraints demand that solutions have some exact value. For example, the constraint for horizontal displacement in specific nodes has to be zero. This type of restriction is rarely used in structural optimization. Most commonly used are inequality constraints that set an upper or lower limit of the value of some specific penalty function and it will now be shown how they are implemented in our GAs optimization process.

#### 4.4.1 Method

In structural design there are limitations that are rigidly defined. The material has limitations, often revised using the safety coefficients and resulting in a value that cannot be exceeded. For example, the usual yielding limit for construction steel is around 240 *MPa* and that is one of the limitations that can be introduced. The method is constructed in such a way that the user can choose any of the available penalty functions or a combination of them, and define the limit value for any of them. This can be seen in the explanation of the typical user dialog in Appendix A. Individual penalty functions will be described a little bit later. First, the general structure of the penalization algorithm developed within this research has to be addressed.

**Feasibility** If one or several penalty functions are defined, each individual solution can be marked as *feasible* or *infeasible*. If the solution doesn't violate any constraints it is feasible, and if it does it is infeasible. Graphically it can be represented as in Figure 4.36, where the attempt is made to illustrate that the feasible solutions usually represent only a part of the whole search space. Of course, the penalty functions must be set in a reasonable way to be effective. If the restrictions are too loose, it can happen that the whole search space is feasible. And vice versa, if the constraints are too tight it might happen that no part of the search space is feasible. Neither of the two problems can cause difficulties in the developed software, since the optimization process can produce results in both cases as it will be shown, but for optimal results they should be avoided.

The constraint, i.e., limit value can represent some lower limit but it is usually set as an upper limit of stress, length, area, etc. A limit value can play the role of a *death penalty* [40] in a way that it simply eliminates the infeasible solutions from further reproduction by giving them no chance of survival. That is however bad for two reasons. First, the optimization becomes elitist, which, as mentioned before, results in poor exploration of the search space.

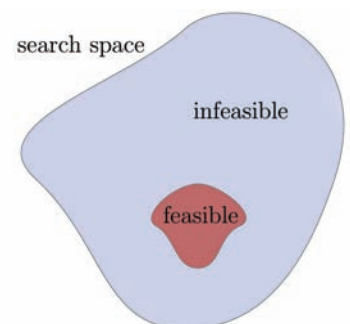
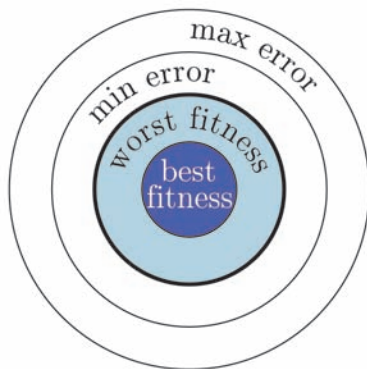


Figure 4.36: Search space

Second, infeasible solutions can be on the border of infeasibility and therefore have very good *genetic material*, that can be used to produce children with good fitness values. Keeping those solutions alive makes the optimization process more robust and faster. That is why in this research they are penalized and evaluated according to their *error*.

**Error** In Appendix B, where the data structure of an individual in our GAs optimization is showed, it can be seen that there is a piece of information about the individual’s feasibility, as well as additional information about the size of its error. If the solution is infeasible, it is important to measure how much it violates the constraints in order to evaluate it appropriately. That violation is referred to here as an *error*. Figure 4.37 is trying to illustrate the effect which we want to produce. We want to create a *magnetic effect* that attracts the solutions to the feasible search space area. That can be done by evaluating them in such a way that the feasible solution values depend on their fitness, and infeasible ones on the size of their error. The closer the solution is to the center, the higher it should be evaluated and the higher its survival chances should be.



**Figure 4.37:** Magnetic effect

The easiest way to demonstrate what an error is, is to use an actual example of constraint. Let us imagine that we defined that no member in a grid shell structure should exceed the length of  $5m$ . Let there be 5 members that exceed that limit and have the following lengths:  $6m, 7m, 8m, 9m, 10m$ . We can then add up the lengths that exceed the prescribed limit. So after calculating the error,  $1 + 2 + 3 + 4 + 5$ , we can say that the grid structure, i.e., individual in an optimization process, has an error value of 15. If several different constraints are introduced, they are simply added up. Sometimes when there are several constraints and their values differ too much, some type of scaling has to be performed to bring them to some proportional level. Usually that is not necessary, since our goal is to enter the feasible search space where there are no errors at all. Once the optimization process enters it, the magnetic effect keeps the optimization process inside and we don’t have to bother with infeasible solutions.

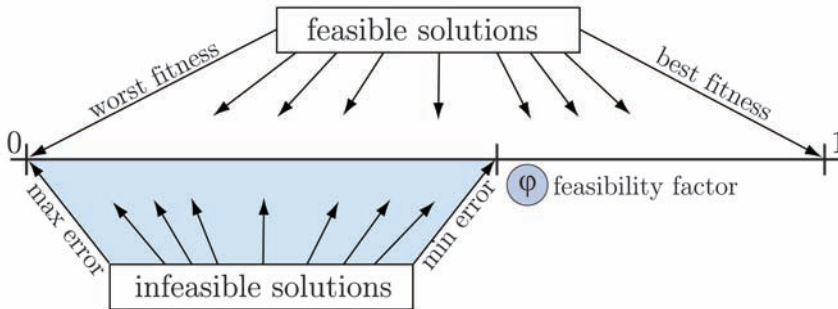
It can easily be seen that the error calculation is specific, and depends on the type of penalty function. Actually, the error calculation is, in a way, the algorithmic definition of a constraint. When the individual penalty functions used in the research are presented (Section 4.4.3), the methods of error calculation developed within this research will be additionally explained.

**Feasibility Factor** When we have the information for each individual in a generation, about its feasibility and its error, we can go to the next step of penalization. This next phase is integrated



into the fitness scaling functions and expands them to achieve the magnetic effect shown in Figure 4.37. Powell and Skolnick [8] remapped fitness values of both feasible and infeasible individuals in such a way that all feasible solutions have higher fitness than any infeasible solutions. This concept assumes the superiority of feasible solutions. The assumption rarely holds since it always happens that some infeasible individuals process very good genes that can be very valuable for later generations. That is why these individuals are preferable during the evolution than many low fitness feasible solutions. It is therefore necessary to allow some infeasible individuals to have higher fitness than some feasible solutions [43]. Taking this into consideration a following method is developed for this research.

Figure 4.38 shows the applied principle and introduction of the *feasibility factor*. It was explained how all the solutions are scaled to values between 0 and 1, so that the selection process can be effective. The feasibility factor  $\phi$  also has a value between 0 and 1 and it determines the space into which infeasible solutions will be scaled. To be more specific, the best infeasible solution (minimal error solution) will have the scaled value determined by the feasibility factor, as can be seen in the figure.



**Figure 4.38:** Expanded Fitness Scaling developed and used in the research

We'll denote the value of the  $i$ th individual in the generation, after the fitness scaling function is applied, as  $f^s(x_i)$ , and the *penalized* value, after the penalization is done, as  $f^p(x_i)$ . If the feasible part of the search space is  $\tilde{\mathbf{F}}$  and the infeasible one  $\tilde{\mathbf{I}}$ , then the revised function can be expressed as:

$$f^p(x_i) = \begin{cases} f^s(x_i) & \text{if } x \in \tilde{\mathbf{F}} \\ f_{inf}^s(x_i) & \text{if } x \in \tilde{\mathbf{I}} \end{cases} \quad (4.20)$$

where  $f_{inf}^s(x_i)$  represents the scaling function for infeasible solutions. This function depends on the error function  $f_{err}$ , made according to the type of penalty chosen to calculate the excess over the prescribed limits. After the error is calculated, the fitness scaling function is altered to distinct between feasible and infeasible

solutions. The application directly follows the form expressed in Equation 4.20. If the solution exceeds no limits it is feasible and scaled normally as described in Section 4.3.3. If the solution however violates restrictions and has a positive error value, it is marked as infeasible  $f_{inf}$  and scaled as follows:

$$f_{inf}^s(x_i) = \begin{cases} \phi(1 - (\frac{f_{err}(x_i) - f_{err}^{min}}{d_{err}})) & \text{if } f_{err}(x_i) \neq f_{err}^{min} \\ \phi & \text{otherwise} \end{cases} \quad (4.21)$$

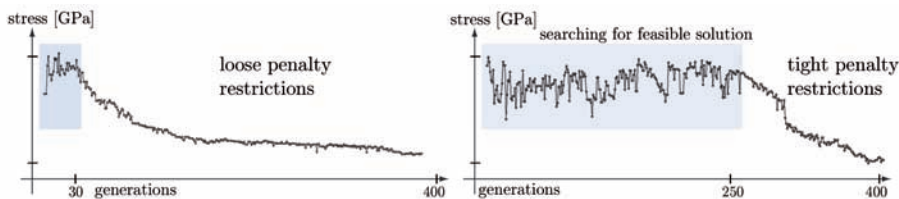
This is similar to the fitness scaling function used in Equation 4.17. The difference is that instead of the fitness value, an error value was used here. Respectively  $f_{err}^{min}(x_i)$  represents the infeasible individual with minimal error and  $d_{err}$  the difference between maximal and minimal error in the generation  $d_{err} = f_{err}^{max} - f_{err}^{min}$ . Since the minimal error solution is the best infeasible one, the value calculated was subtracted from 1 in order to make the solution with a smaller error have bigger values, thus bigger chances of survival. At the end, the value is multiplied by  $\phi$ , to limit the infeasible solutions and give them a smaller chance of survival than the feasible ones. In the case of  $f_{err}(x_i) = f_{err}^{min}(x_i)$ , we have our best infeasible solution and we assign it the maximal value, defined by feasibility factor  $\phi$ . This calculation is performed for all infeasible solutions.

#### 4.4.2 Application

As always, the process is not as complicated as it looks when expressed mathematically, and it will be demonstrated with a small example. Let us turn to the grid shell optimization again where the size of each member is limited to  $5m$ , and imagine we have a generation of 10 grid shell individuals from which 6 are feasible, and 4 are not. The 6 feasible ones will be scaled normally and will have scaled values between 0 and 1 as described before. We will define the error of each of the 4 infeasible grid shells as follows: 5,10,15,20. We have  $f_{err}^{min} = 5$  and  $f_{err}^{max} = 20$  and therefore  $d_{err} = 20 - 5 = 15$ . If we set the feasibility factor to be  $\phi = 0.6$ , for example, we can calculate the fitness values. The solution with error 5 is the best infeasible solution ( $f_{err}(x_i) = f_{err}^{min}(x_i)$ ) and will have a value of 0.6. The solution with value 20 is the worst infeasible solution and will have, after calculating  $0.6(1 - ((20 - 5)/15))$ , a value of 0, and therefore have no chance of survival. For the individual with an error of 10 the scaled value will be  $0.6(1 - ((10 - 5)/15)) = 0.396$ , and for the individual with an error of 15 the scaled value will be  $0.6(1 - ((15 - 5)/15)) = 0.198$ . It is clear how the solutions are ranked according to their error in a way that the individuals with smaller error have bigger chance of survival: 5(0.6), 10(0.396), 15(0.198), 20(0.00). Since the feasible solutions are scaled to values between 0 and 1, there will

be at least one solution with the scaled value of 1 that will have the most chances of survival, thus supporting the reproduction of feasible solutions.

Sometimes, when the restrictions are rigid, it can happen that all solutions in a generation are infeasible. Furthermore this can last for many generations, as it will be seen later in the examples. The optimization process then converges toward a minimal error solution until it generates a feasible solution. Once the feasible individuals are created, the solutions converge to an optimal solution inside that area. In Figure 4.39 two graphs are shown, that represent the progress of the average fitness value in a generation, in two optimization processes that try to minimize the stress in the structure. The graph on the left is an optimization process with very loose penalty functions and it converges almost immediately to a global optimum. The graph on the right however has rigid limitations and generates only the infeasible solutions for almost 250 generations. Then it finally finds the feasible part of the search space and converges inside of it. The irregularity of the graph in the first 250 generations shows how the algorithm doesn't allow the convergence, and *struggles* until it generates feasible solutions.



**Figure 4.39:** The magnetic effect of the penalty functions

**Adaptivity** The method applied in the research belongs to the *static penalty function* methods, meaning that the penalty settings are made at the beginning of the process and do not depend on the current generation number [40]. The feasibility factor is set at the beginning, to some reasonable value like 0.5 or 0.6. In that way it works satisfactorily, keeping the infeasible solutions below that limit and allowing the optimization process to converge fast inside the feasible area of the search space, once the solutions inside that area are generated. As with every part of GAs, this can be even more optimized, and one way to do that is to let the feasibility factor vary and adapt itself throughout the generations [40, 43, 28]. In *adaptive penalty function* methods information gathered from the search process is used to control the amount of penalty added to infeasible individuals.

One way to do this depends on the number of infeasible solutions in the generation, and it should set the feasibility factor to be proportional to the share of feasible solutions in the generation.

Namely, the more feasible solutions available, the bigger the feasibility factor should be, allowing the small number of infeasible solutions to be competitive. Therefore a following method is proposed here only in theory and will be experimented with in future research. If there are  $f + i$  individuals in a generation, where  $f$  represents the number of feasible ones and  $i$  the number of infeasible ones, then the feasibility factor can be expressed as:

$$\phi = \frac{f}{f+i} \quad (4.22)$$

Again, additional factors can be introduced that can set the lower and upper limits of  $\phi$ , but we are not going to go into that much detail. The effect of the adaptive factor on the whole optimization process is very small, as it has already been said that once the process gets inside the feasible search space, most of the solutions are feasible and the treatment of the infeasible ones is not important.

### 4.4.3 Examples

Logic dictates that the introduction of penalties must have some trade-off. Namely, the fitness of the best acceptable individual must lose some of its value, since the best solution without restrictions is usually out of the feasible part of the search space. This will be clear after some results of the experiments are shown and compared. In Chapter 5 the effect of the fitness functions, together with penalization, will be explained in much more detail. However, here we will jump ahead a little bit and try to show small examples of the application of some penalty functions together with their description.

#### Minimum and Maximum Member Length

Manufacturing conditions, as well as some design intentions, can lead to the limitation of the structural members' lengths. One grid shell individual can have  $n$  members, each one with the length  $l_i$ . We will introduce the penalty limit  $p = 5m$  that determines the feasibility of our solution. To calculate the error we check all the members and sum up their constraint violation  $e_i$ :

$$f_{err} = \sum_{i=0}^n e_i \quad (4.23)$$

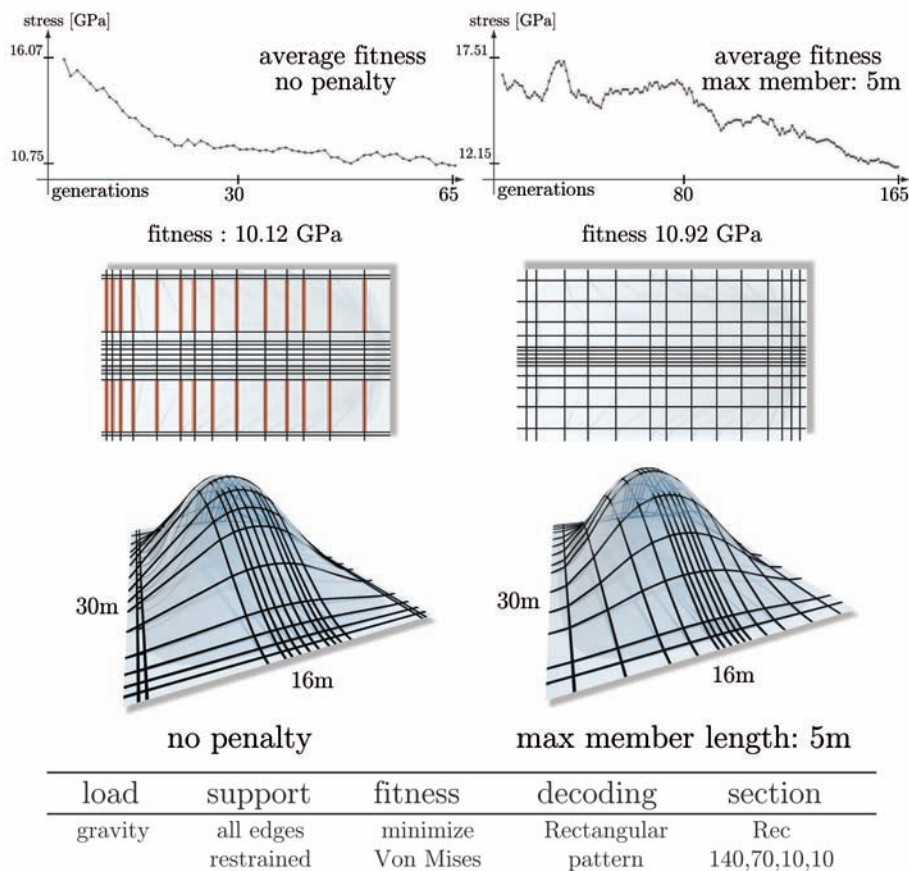
but we calculate  $e_i$  only if the member exceeds the prescribed limit:

$$e_i = \begin{cases} l_i - p & \text{if } l_i > p \\ 0 & \text{otherwise} \end{cases} \quad (4.24)$$

When we want the lower limit, i.e., for members not to be shorter than some length,  $e_i$  is calculated like this:

$$e_i = \begin{cases} p - l_i & \text{if } l_i < p \\ 0 & \text{otherwise} \end{cases} \quad (4.25)$$

In Figure 4.40 we can see a small example of an optimization process done without penalty functions (left) and the result of an optimization with limited member length (right). A free form



**Figure 4.40:** Effects of penalty functions, limited member length

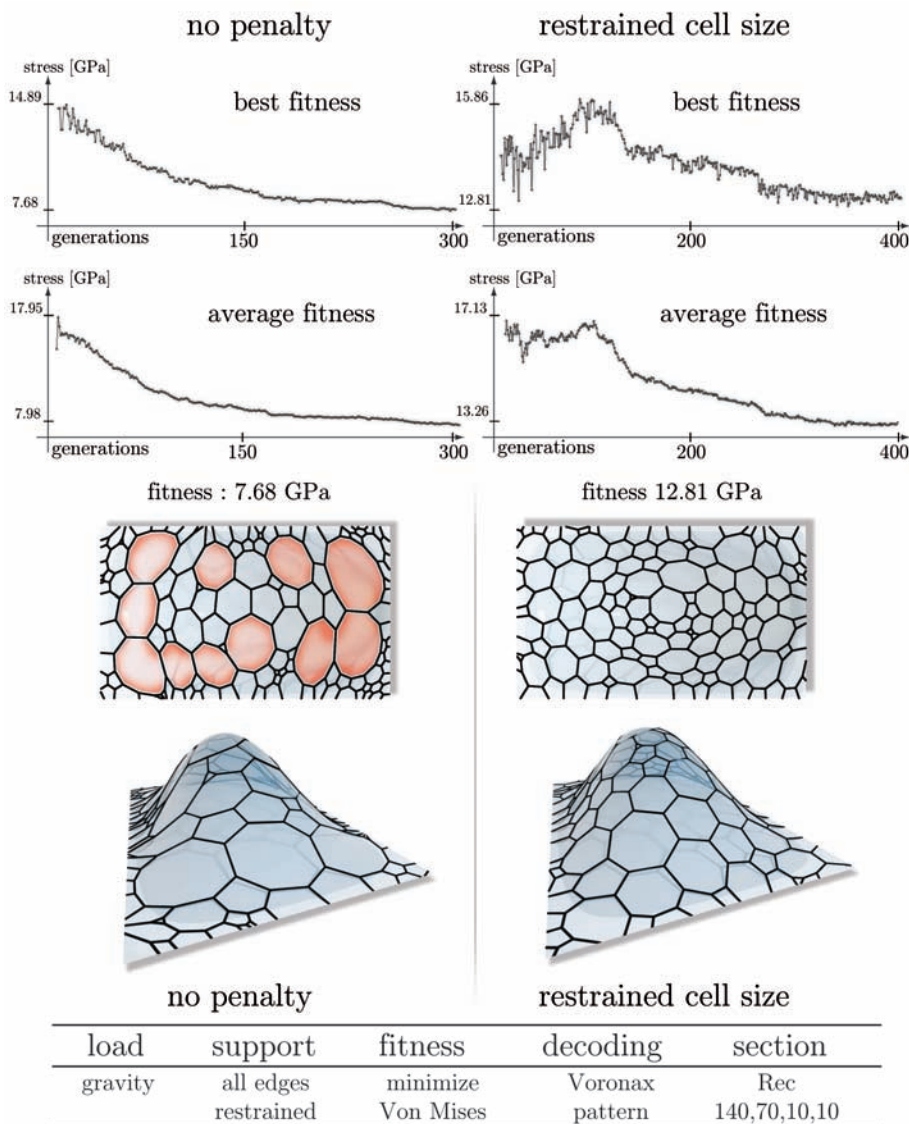
$30 \times 16m$  surface was used, with a 420-member grid shell generated over it, and the minimization of Von Mises stress is used as a fitness function. Other conditions, like load buckling factor, were not taken into consideration in this example since the idea is to show the effect of one constraint with single fitness function. The optimization is done over a rectangle pattern where the members can only move along the edges, thus forming a rigid, straight-line, grid. However, this is enough to show the effects of penalization. As shown in the table on the bottom of the figure, the applied load was self-weight of the structural members, all nodes on the edges were restrained from movement and rotation and members have a vertically oriented (parallel to the global  $z$  axis) rectangular cross

section. The graph on the left shows how after 65 generations (each consisting of 50 individuals) the optimization process converges to an optimal solution. Beneath the graph we can see its *intention* to stiffen the middle part, thus creating basically primary and secondary elements, i.e., a central beam and ribs. The ribs are very long (marked red), more than  $10m$ , and it is reasonable to *ask* the algorithm what would the result be if the length of each member was limited to  $5m$ . The graph on the right shows the results, and it can be clearly seen that we need more generations (165 in this case) for the algorithm to find a good feasible solution among the huge number of infeasible ones. The graph indicates the *struggle* for almost 100 generations until it finds the feasible search space and starts to go down. It is also clear that the convergence would soon be achieved in the next 50 or 100 generations. The best grid shell generated in the last (165<sup>th</sup>) generation is shown underneath and it can be seen that the limitations were respected and that the longest members do not exceed  $5m$ . As mentioned, there is a trade of in fitness, but very small one. Namely the fitness of the best feasible solution, i.e., the total amount of Von Mises stress is  $10.12GPa$  and the fitness of the best infeasible solution is  $10.92GPa$ . Those values would go down as the process continues but, as in most cases, the infeasible solution would have a slightly better, i.e., smaller value. The important thing to mention is that the restrained process still keeps the same intention of stiffening the middle bearer, but it does it within that one prescribed length limitation.

### Minimum and Maximum Cell Area

In Appendix C an algorithm, constructed to gain the information about cells (structural polygons), is explained. With that information we can control the size and shape of the cell. We can limit the number of its sides, or the maximal and minimal angle between its members, but the most ubiquitous constraint used is the limitation of the cell area. Similar to the last example, the error can be calculated as the sum of the differences ( $e_i$ ) between all cell areas and the limit value. If there are  $n$  cells we can refer to a single cell area as  $l_i$  and we don't have to write new equations since they are the same as 4.23, 4.24 and 4.25 if  $p$  represents the limit. In that way we can predetermine the minimal and maximal cell area.

With the same surface as in the previous example, in Figure 4.41 we can see two optimization processes with Voronax pattern and 110 Voronoi seed, resulting in 330 structural members in the two solutions presented. The applied load was self-weight of the structural members and surface load of  $1KN/m^2$  transferred over the cells to the structural joints. All nodes on the edges were restrained from movement and rotation and members have a vertically oriented rectangular cross section. Here, there are two addi-



**Figure 4.41:** Effects of penalty functions, limited cell area

tional graphs, that show the convergence of the best fitness solution in the generation. They show clearly the *struggle* of the optimization process when it comes to penalization. This comes from the newly created infeasible solutions that appear in each generation and disturb the convergence. Nevertheless, convergence does appear, it only needs more time, i.e., more generations. On the left, there is a nice convergence and we can see how the smaller cells move toward the supported edges to stiffen those parts that are under the biggest stress. Doing that, they relieve the middle convex part by generating extremely large cells there (marked red). On the right we can see what happens when we restrain the size of the cell. For around 100 generations the process mostly generates infeasible solutions, but then it finally finds the feasible search space and converges toward the best solution inside it. One of the best



solutions demonstrates the effects of imposed constraints and shows the trade-off in fitness value that comes from it.

### **Easily Expandable**

Penalty functions can be combined, and the dialog in Appendix A shows that, in our research, combinations of 3 functions were used. This is however not limited, and as many penalty functions as needed can be combined. Easily enough, new functions can be defined according to the specific demands of the project. All that has to be done is to define a new function that will calculate the error if the solution is marked as infeasible, and the expanded fitness scaling function will take care of the rest.

More effects of penalty functions will be demonstrated in the next chapter, together with various fitness functions. It will be shown how for a small trade-off in fitness we can direct the convergence and control the process easily.

## **4.5 Multi-Objective Optimization**

Penalty functions allow us to influence an optimization process with several factors. However, there is always one main objective, and different restrictions are used to set the boundaries of the search space where an optimal solution should be found. The process is therefore single-objective, but limited. In structural design it is sometimes interesting to see what would happen if more objectives were considered at the same time. That leads to a multi-objective (MO) optimization problem.

Sometimes the different goals that we are trying to achieve in structural optimization lead to the same tendency, i.e., the structure with better fitness according to one objective is better according to the second one too. This happens rarely, and therefore is not that interesting. What is much more challenging is to optimize some structure according to objectives that have different tendencies. Then, there is always a trade-off between them and the solution that satisfies all the objectives in the best possible way is impossible to achieve.

Speaking in general terms, our fitness function  $f(x)$  now becomes a vector of functions  $\mathbf{f} = (f_1(x), f_2(x), \dots, f_n(x))$  and our goal is to find the aforementioned design vector  $x = (x_1, x_2, \dots, x_D)$  that represents the solution which optimizes  $\mathbf{f}$ . Of course, penalty functions can be used normally to limit the search space even if the optimization is multi-objective. The constraints only have to be set more carefully.

In economics, there are lots of *players* with different goals (objectives). There is a theory, developed and used to mathematically capture behavior in strategic situations, in which an individual's

success in making choices depends on the choices of others, called *Game theory*. It is mainly used in economics, but the basic principles of it can be applied in other spheres of science like engineering, political sciences, international relations, computer science, philosophy, evolutionary biology, etc. One of the famous concepts of Game theory is the Nash equilibrium, developed by John Nash in 1951 at The Princeton University [41]. It is made with the exact intention of finding a solution that would be satisfactory for all players at the same time. However, the Nash equilibrium finds an equilibrium point where no player has interest to change its position, but that point doesn't necessarily represent the best solution for all the players. That is why another principle from Game theory is generally used in multi-objective structural optimization - the Pareto Optimum.

### 4.5.1 Pareto Optimum

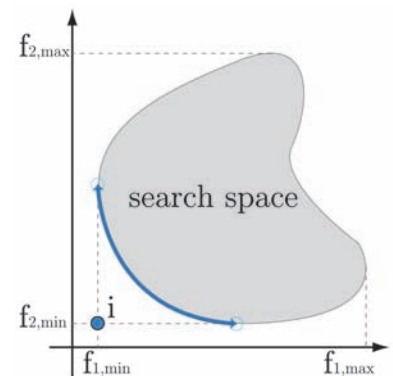
The method was developed by Vilfredo Federico Damaso Pareto, an Italian economist, at the beginning of the 20<sup>th</sup> century [1]. He used it mainly for studies in economic efficiency, but the general principle can be applied to any *conflict situation*. The method will now be explained for two objectives optimization along with its expansion for three objectives. The principle can be then followed to expand the algorithm for more than 3 goals.

#### Pareto Optimization With Two Objectives

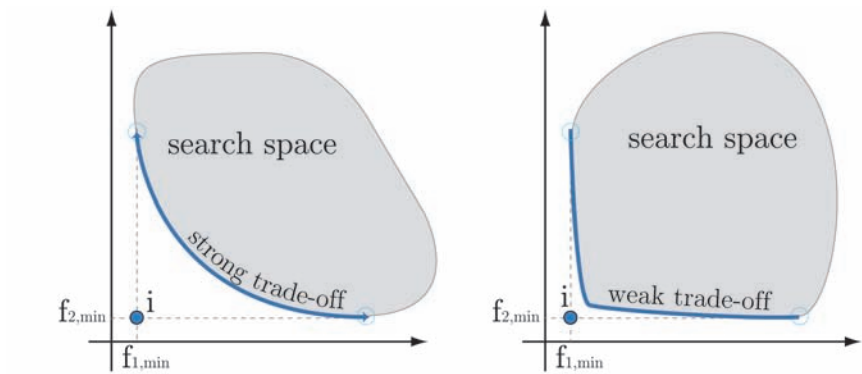
Let the first objective function (fitness function) be denoted as  $f_1$  and the second one as  $f_2$ . We can then construct a diagram as shown in Figure 4.42. For now, let both fitness functions represent a minimization problem. If the gray area represents our search space and the blue dot represents an ideal solution  $i(f_1, f_2)$ , we can mark the part of the search space where we would like to find our solution with the blue line.

The blue line represents the *Pareto frontier*, a line on which the set of choices would be *Pareto efficient*. Depending on the shape of the Pareto frontier we can make distinctions between solutions with a strong and a weak trade-off between the objectives. As it is depicted in Figure 4.43, it is obvious that the trade-off depends on the distance that the solutions will have from the ideal point. Naturally, the generated solutions are always inside the search space, and rarely on the border as shown in previous figures. So the Pareto frontier has to be formed out of available solutions.

**Pareto Frontier** The question of evaluating solutions, and determining a Pareto frontier from the set of available solutions, remains



**Figure 4.42:** Pareto frontier

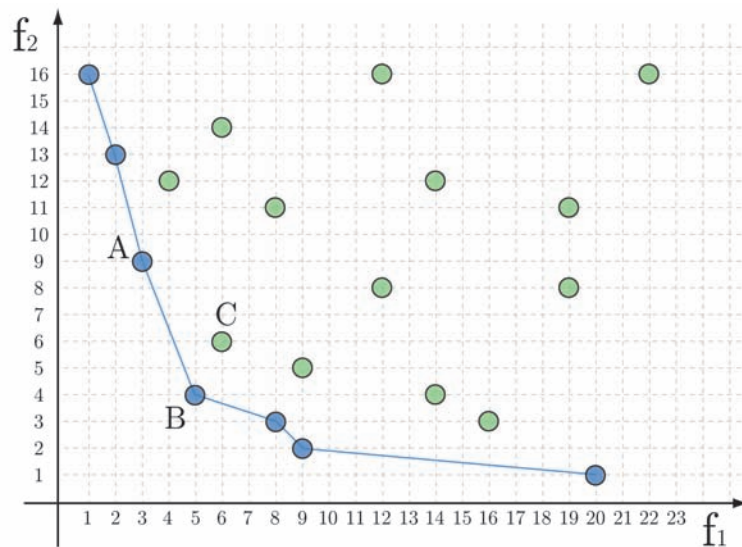


**Figure 4.43:** Strong and weak trade-off

unresolved. The easiest way to resolve it is to construct a simple example. Let us imagine a minimization process where one generation has 10 solutions with two values for each of the fitness functions, as depicted in the diagram in Figure 4.44. The position of each solution on the graph is determined by those two values. Now we introduce the notion of *domination* from the Game theory. It is stated that the solution is on the Pareto frontier if it is not strictly dominated by any other solution. If we compare two solutions, the first strictly dominates the second one if none of its values are inferior to the corresponding values of the second one, and if at least one of its fitness values is better (smaller in a minimization, bigger in a maximization) than the corresponding one in the second solution. For example, in Figure 4.44 we have marked solutions:  $A(3,9)$ ,  $B(5,4)$ ,  $C(6,6)$ , where the first number represents the value of the first fitness function and the second number the value of the second fitness function. If we compare solutions A and B we can see that A has a better fitness value resulting from the first fitness function, and solution B is better than A when evaluated by the second fitness function. Therefore it cannot be said that one solution strictly dominates the other. If we compare B and C it is clear that solution B dominates solution C, since it is better in both fitness functions, i.e., has smaller value and therefore is closer to the ideal solution. Since solution C is strictly dominated by at least one solution (B in this case) it is not on the Pareto frontier. All the solutions represented with blue points are not dominated by any other and do represent the Pareto frontier. All green solutions are, like solution C, dominated by at least one other solution.

### Application

The application of the method of determining the Pareto frontier can be creatively determined in a number of different ways. For the presented research a version of this method is developed which relies on scaling methods similar to the ones that we've seen in the



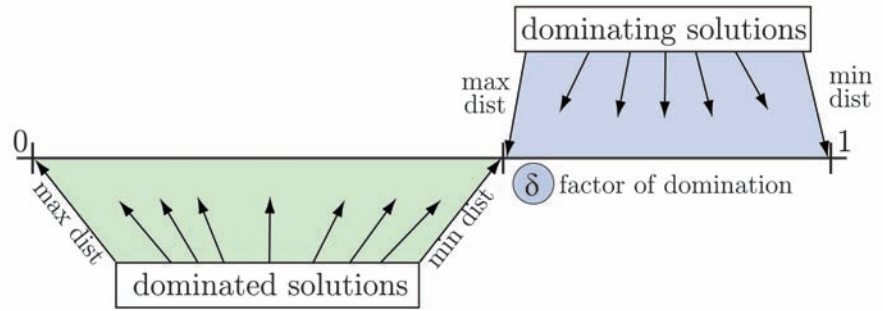
**Figure 4.44:** Pareto frontier (blue) and other solutions in the generation (green)

penalization process. In each generation the solutions were sorted out by different chances of survival being assigned to them.

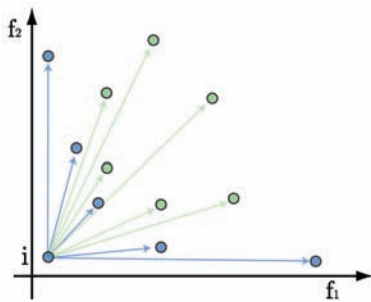
Initially, it is determined if the solution is dominated by any other solution or if it is on the Pareto frontier. Algorithmically that is an easy task that needs a maximum  $O(m(n^2))$  computing time, where  $m$  represents the number of objectives in the optimization and  $n$  the number of individuals in the observed generation. We simply check each individual and compare its fitness values with every other individual in the generation. If there is at least one solution that has all fitness values better, then the observed individual is dominated and it is not on the Pareto frontier. If there is no individual with overall better values, then the considered individual is not dominated and it is on the Pareto frontier.

Similar to the feasibility factor, the *factor of domination*  $\delta$  is introduced here. We know in advance that dominating solutions represent a minority in a generation, and therefore all of them need to have a bigger chance of survival than any dominated solution. Solutions on the frontier will be scaled to values between  $\delta$  and 1, whereas the dominated solutions will be scaled to values between 0 and  $\delta$ , as shown in Figure 4.45. In that way we can preserve the small number of good solutions by giving them higher chances of survival and forcing the optimization process to generate individuals closer to the ideal point, from one generation to the next generation.

In Appendix B it can be seen that the data structure of each individual in a MO optimization includes additional information about its domination. Namely, every individual is marked as dominated or not, and the distance between it and the ideal point  $i$  is



**Figure 4.45:** Dominating and dominated solutions



**Figure 4.46:** Distances from ideal point

also known. It was explained before that the ideal point represents an ideal solution which would represent the best possible fitness value for all objectives. The closer the point is to this ideal point, the better it should be evaluated, and the bigger its chances for survival should be. To visualize this, Figure 4.46 shows the distances from the ideal point to the solution. This idea is used to expand the fitness scaling function for MO optimization.

If there are  $n$  individuals in one generation, they can belong to the frontier  $\tilde{\mathbf{P}}$  or not, and therefore be scaled appropriately. If we denote the multi-objective scaling function as  $f_{mo}(x_i)$ , the scaling of solutions on the Pareto frontier as  $f_{pf}(x_i)$  and the ones that are dominated as  $f_{dom}(x_i)$  then:

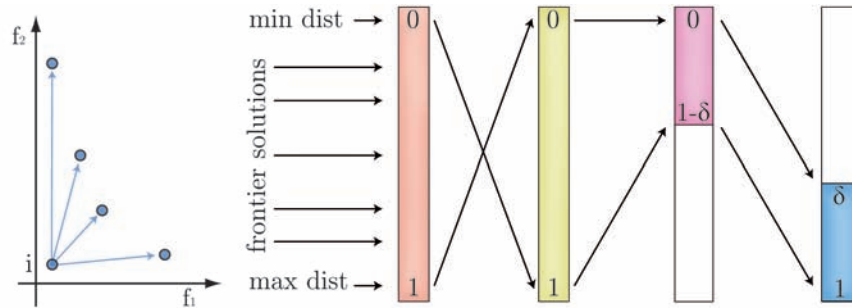
$$f_{mo}(x_i) = \begin{cases} f_{pf}(x_i) & \text{if } x \in \tilde{\mathbf{P}} \\ f_{dom}(x_i) & \text{otherwise} \end{cases} \quad (4.26)$$

The ideal point in a minimization problem is at the point where all objectives are theoretically minimal, i.e., have 0 value. In a two objective case, this means that the distances from the point of origin will be calculated and denoted as  $l_i$  for each individual. Like in the penalization,  $l_{max}$  and  $l_{min}$  are determined and their difference is calculated  $d = l_{max} - l_{min}$ . The Pareto frontier scaling function can then be expressed as:

$$f_{pf}(x_i) = \begin{cases} \delta + ((1 - \delta)(1 - (\frac{l_i - l_{min}}{d}))) & \text{if } l_i \neq l_{min} \\ 1 & \text{otherwise} \end{cases} \quad (4.27)$$

First, the solution is scaled to a value between 0 and 1 according to its distance from the ideal point:  $\frac{l_i - l_{min}}{d}$ . Then the value is subtracted from 1 because we want the solutions at smaller distances to have greater value:  $1 - (\frac{l_i - l_{min}}{d})$ . Now that we have values between 0 and 1, we scale them further to values between 0 and  $(1 - \delta)$  adding it to the equation:  $(1 - \delta)(1 - (\frac{l_i - l_{min}}{d}))$ . In the end we want to shift them to values between  $\delta$  and 1 and we get the final form:  $\delta + ((1 - \delta)(1 - (\frac{l_i - l_{min}}{d})))$ . If we have the best solution

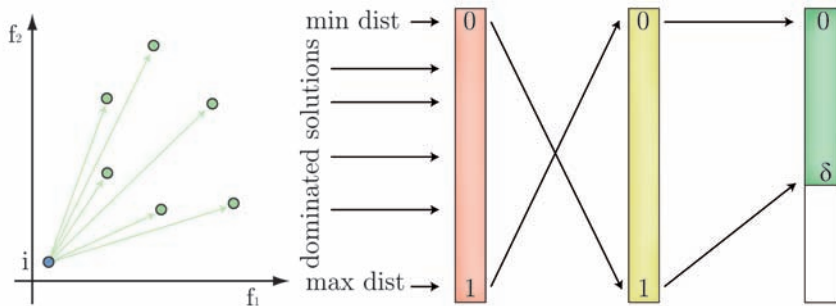
(minimal distance from ideal point), we scale it automatically to 1. The process is depicted in Figure 4.47.



**Figure 4.47:** Scaling of fitness values of the solutions on the Pareto frontier

With the dominated solutions we do the same thing, only without the last step and instead of  $1 - \delta$  we scale them to values between 0 and  $\delta$ :

$$f_{dom}(x_i) = \begin{cases} \delta(1 - (\frac{l_i - l_{min}}{d})) & \text{if } l_i \neq l_{min} \\ \delta & \text{otherwise} \end{cases} \quad (4.28)$$

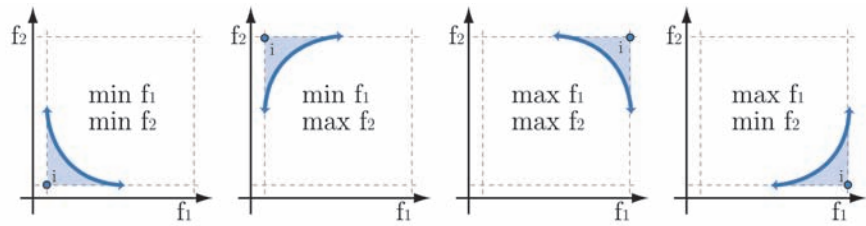


**Figure 4.48:** Scaling of dominated solutions

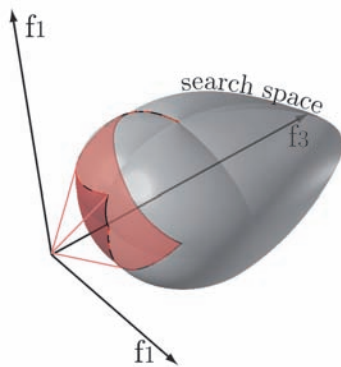
After scaling is performed, the optimization algorithm continues as usual. With the method described, it will always try to converge toward the ideal points and get as close to it as it can. Naturally, in two-objective optimization the position of the ideal point depends on the type of fitness function. Since each one can be minimized or maximized, there are 4 possible positions of the ideal point, as depicted in Figure 4.49.

### More Than Two Objectives

There can be more than two objectives, and for 3 fitness functions, where all of them are supposed to be minimized, the Pareto frontier can be depicted as in Figure 4.50. For a higher number of goals it is hard to analyze MO optimization graphically, but all the principles



**Figure 4.49:** Possible positions of an ideal solution



**Figure 4.50:** Pareto frontier with 3 objectives

remain the same. The results of the MO optimization experiments will be presented in the next chapter, together with all other results.

### Additional Scaling

The multi-objective optimization developed and used in the research is explained. However, this method, as every other, can be easily expanded and finely tuned to respond to specific requirements. For example, we can introduce additional factors that would prefer the solutions closer to one objective, if we find one fitness function more important than the other. That is however not important for the proof of the efficiency of the whole GAs optimization that is presented here, and will be left to future research.



# 5

## Results

The main structure of the algorithm has been presented so far, and most of its parts are explained in as much detail as seemed reasonable and required to understand the whole process. The really interesting part comes with the application of Genetic Algorithms and with the comparison of the results. The comparison of outcomes with different GAs settings will show how the optimization can be directed and controlled, but it will also supply an additional proof that the method works efficiently. It will be interesting to see how, after only a few experiments, simple conclusions can be made in order to predict the outcomes of other optimization processes. That is exactly what the goal was from the beginning - to develop a new type of intuition in free form structural design. Later on, the results will be compared to structures in Nature in order to demonstrate that with this method we are now *thinking* the same, and that once we are liberated from manufacturing restrictions we can come closer to building structures as beautiful and as efficient as natural ones.

The best way to show the effects of the GAs optimization is to change one of the parameters and keep all others fixed. Out of many components that can vary, several of the most important ones are chosen. Namely, the results of GAs structural optimizations will be presented, made according to the different: fitness functions (in single and multi-objective optimization), patterns, load and support combinations.

In each of the following sections there will be a set of input data presented at the beginning, i.e., the information about the most important parameters of the optimization process, with special emphasis on the study of its effects. The input data is basically divided into 3 categories as shown in Figure 5.1: GAs specific data, Pattern specific data and Fitness data. GAs specific data regulate the process with different probability factors, and determine the size of the population. Pattern specific data determine the size of the chromosome, i.e., the number of Voronoi seed and therefore the number of joints and structural members in the grid shell. Fit-

ness data holds the information about different settings (including Penalty functions) that are decisive in the process of evaluation.

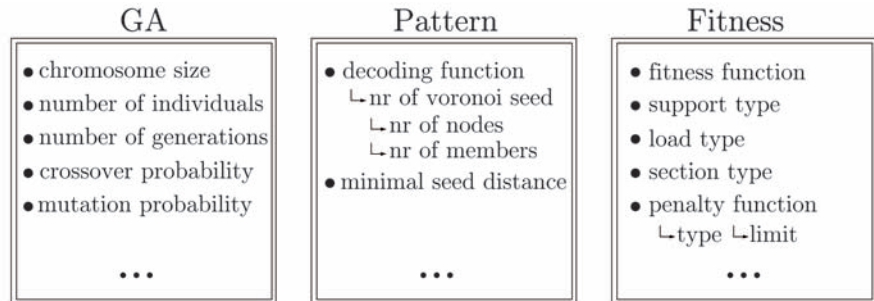


Figure 5.1: Input data

Every optimization process holds information that can be used afterward to recreate and draw the entire population of individual solutions or extract the graphs, that show the progress of the entire optimization process. The 4 graphs illustrated in Figure 5.2 represent the progress of the: 1. Maximal fitness value in generation, 2. Minimal fitness value in generation, 3. Average fitness value in generation, 4. Sum of all fitness values in generation. Out of those, two are usually more important, i.e., Minimal fitness value and Average fitness value. That is why in all experiments they will accompany the figures, to demonstrate the convergence of each GAs optimization process.

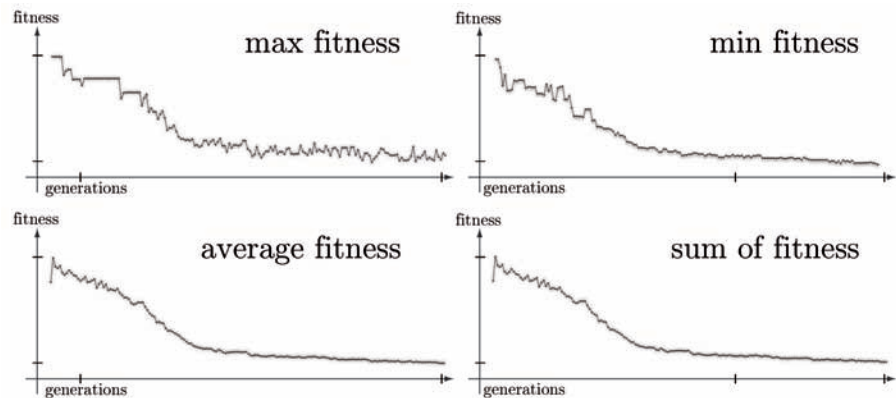


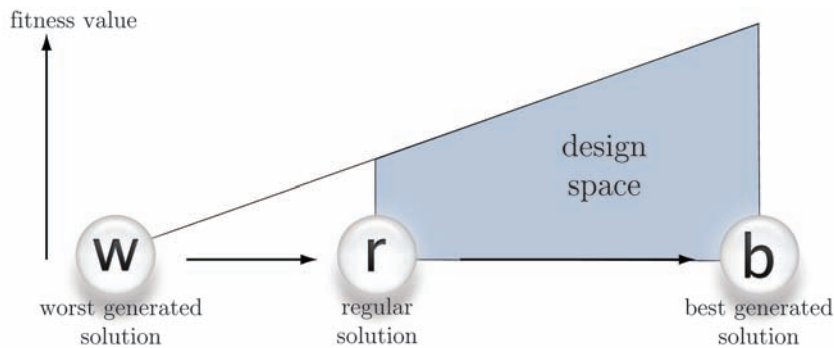
Figure 5.2: Optimization graphs

In order to demonstrate the innovations in this research, solutions will mainly be compared to the *regular* structure, i.e., the structure with uniformly distributed structural members, since that is the way they are usually designed nowadays, and that is something that can be improved. Since the polygons in Voronax structures have an average of 6 edges, they are usually compared to a regular hexagonal structure. The information obtained from a Voronax optimization will be used to optimize triangular and quad-

---

rangular structures as well, and they will also be compared to their uniform versions.

In each optimization process there are random solutions, generated at the beginning, with the bad fitness values, that naturally disappear after a few generations. Those solutions are not the worst possible ones, but simply the worst in the entire generated population. Generally, the *regular* solution will have a fitness value somewhere between the best and the worst, as depicted in Figure 5.3. In the same figure there is an area, marked blue, where we basically look for our design solution, and, as it will be shown in the following sections, we can decide upon one solution according to the different restrictions and our design aspirations.

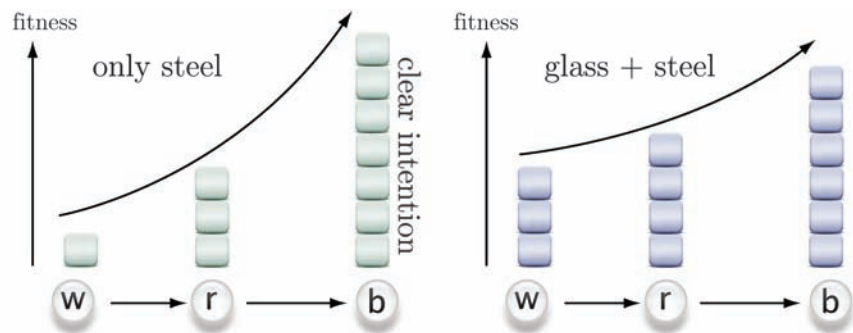


**Figure 5.3:** Optimal design is somewhere between the regular and the best fitness structure

Additionally, our goal is to extract a *pattern of behavior* that appears when we look at the entire population of solutions. In other words, we will try to read the *intention* of the optimization process, to see how the change in the geometrical disposition of structural members affects the statical behavior of the grid shell. Experimenting with different settings allowed us to determine the interesting ways of achieving that.

We can, for example, do the same experiment twice with different load settings. First, we apply normal gravitational load, applying the weight of the steel members and the weight of the glass (or any other material) that covers the cells. Then, we do the same experiment without the glass, i.e., only with the weight of the steel structural members. Since the glass is basically a constant surface load over the entire free form shape (regardless of the member disposition), it only *stabilizes* the grid, not allowing the creation of great differences in cell size or member lengths. With the experiments without the glass load, the solutions are geometrically *more extreme*, but the most important thing is that both experiments usually show the same *intention*, and when it is not clear what it is in the first optimization process, we can use the second one to confirm it. This will be clear when the actual experiments are shown, but it is important to give an example of the

creativity involved in the interpretation of the optimization results. Naturally, the gains with the experiments with only member load are bigger. For example, in some cases the best offered solution has a 5 times smaller total amount of Von Mises stress than the regular structure, and sometimes a 10 times smaller total amount of displacement. As depicted in Figure 5.4, the gains with the surface load are a bit smaller, but still substantial. It cannot be generally estimated how big the benefits of an optimization like this are, since it heavily depends on the free form shape, pattern and the fitness function. Experience showed that the more curved the form is, and the larger the freedom in geometrical distribution of members is, the more we can profit from the structural optimization. In some cases it comes as a surprise to see that, without changing the shape or the number of structural joints and members, but only by changing their geometrical disposition over that shape, we can achieve structures with several times less stress or displacement, or with multiple enhancement of stability. However, the optimization depends on choice of the pattern. Triangular structures, due to their rigidity, have the ability to remain stable and perform good according to different load cases. With n-gon patterns (like Voronax) the optimizations are done usually for one load combination and the effects of sudden changes in load magnitude and direction would have to be additionally investigated.



**Figure 5.4:** Different gains from different load settings

Every aspect of the optimization process presented so far can be constructed in a very complex manner. The choice is made however not to create confusion or prove the enormous possibilities of the method by demonstrating that complexity. Hopefully, from all the proofs and explanations offered, it will be clear what tremendous creativity can be expressed in the definition of every part of the code. We will however concentrate on restrained optimizations, with simplified surfaces and parameters, in order to prove, beyond any doubt, that the process gives an optimal solution directed by the user and the information provided by them. The examples will therefore show intuitively reasonable results and *convince* us to *trust* the method when we have complicated surfaces and complex

parameters where our intuition cannot help us. The development of new intuition will then come as a *side effect*.

## 5.1 Different Fitness Functions

We start with results of the fitness functions described in the previous chapter. In the following sections, most of the experiments were carried out with relatively simple free form shapes in order to clearly see the effects of the optimization process, thus proving its efficiency. However, in order to support the statement that a grid structure over *any NURBS surface* can be optimized, in this first section we introduce a slightly more complex surface (Figure 5.5), 21m wide and 33m long. Every NURBS surface is represented over its  $uv$  parameters and therefore can be optimized with the proposed method. The parameters used here will generally also be used in most of the experiments in the following sections, with differences in pattern, as well as load and support combinations, when those are the variables the effects of which are investigated. In the optimization procedures each generation had 50 individual solutions, and the number of generations varied according to the problem. All the GAs parameters, like mutation and crossover probabilities, were determined so that they can produce the best convergence of the optimization process, according to the conclusions drawn from many other testings. As it can be seen in the table in Figure 5.5, gravitational load is used in all experiments in this section, i.e., self-weight of the steel structural members (vertically oriented) and surface load of  $1\text{KN}/\text{m}^2$  in the vertical,  $-z$  direction, transferred over the cells to the structural joints. Joints on the 4 edges of the surface are restrained from movement and rotation in all directions.

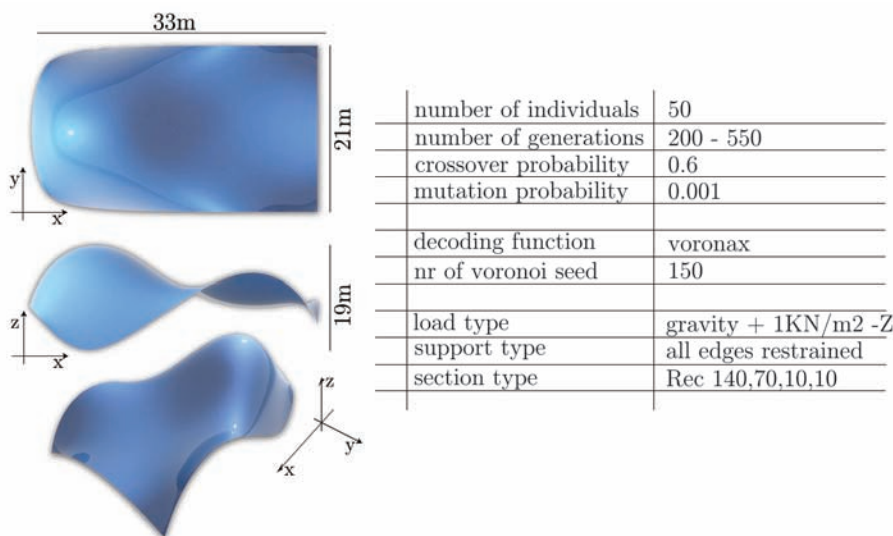


Figure 5.5: Surface and parameters

All members are assigned identically, vertically oriented, rectangular section, and 150 Voronoi seed produced grid shell solutions with 455-465 structural members. It is important to mention that in all of the experiments the self-weight of the grid structure remains approximately the same. The applied surface load is always uniformly dispersed over the entire shape, and the sum of all member lengths (hence weights) differs by maximum 5% for different solutions with the same number of structural members. Therefore, the optimization goal can be stated as - trying to minimize or maximize a specific value, while keeping the weight, i.e., the total mass of the structure, the same.

The goal here is to demonstrate the differences and similarities in solutions generated by different fitness functions. That is why in this section, we will concentrate mostly on the one grid structure type, namely the Voronax structure. The optimization of the other grid shell types, like triangular or quadrangular, will be addressed in more detail in the following sections. The main difference that has to be considered is that triangular grids are categorically different, since, as mentioned before, they are rigid and show good performance with different load combinations.

### 5.1.1 Sum of Von Mises Stresses

In Section 4.3.2 it was described how the fitness value of this particular fitness function is calculated. We determine the Von Mises stress at each end of every structural member in the grid shell, sum them all up, and then try to minimize that value.

In Figure 5.6 a set of different optimization results is shown, trying to depict how one simple analysis process, that will eventually lead to an optimal final solution, can look. This is however a simplified procedure, limited to a few tests that can be done in one or two days. For a project with a specific fixed free form, we can do many tests with different parameters, combined with thorough static analysis, to obtain an optimal grid structure. We can choose different patterns, a suitable number of structural members and their disposition. Here is where we start with the optimization process, depicted at the top of the figure (a), with 150 Voronoi seed and without any restrictions (penalty functions) in order to show clearly the effects of a single-objective optimization and extract the *intention* of the Genetic Algorithm easily. Initial observation of the graphs shows a steady convergence of the average solution and best solution in one generation, although it is clear that if we continued the process, the curve would continue to go down slowly. For us however, those 330 generations are enough to *read the intention* of the process and see where it strives to go. One look at the fitness values shows that the best generated solution (110GPa) has an almost 3 times smaller fitness value (sum of all Von Mises stresses)

than one of the randomly generated solutions from the first generation ( $304.1GPa$ ), and twice less stress than the solution with uniformly distributed members ( $204.9GPa$ ). All solutions have the same number of structural members - 460. Before we start to analyze the grid, we can look at the best solution from the experiment done with the weight of the structural members only (without the glass load), depicted in the framed area bellow. We can use this to clearly see the *intention* of the GAs process. When the best solution from that process is evaluated with the surface load, we see that it is very close to the optimal solution value ( $115.7GPa$ ), but here we can easily see why. As expected, cell density was increased near the supported edges, since those are the areas with the largest stresses. However, there is an additional stiffening of the area marked red. It seems to be a very important point, that stabilizes the convex part, with the help of additional increased density around its basis. If we analyze the deformation of the quadrangular structure (as seen in the figure), we can notice that it is exactly in that area where the greatest deformations occur, so the reason for which the algorithm stiffened that part is understandable. We also witness the stiffening of the corners, a pattern that will appear in almost all experiments, making it obvious that the GAs process uses the corner's naturally rigid shape to stabilise the structure when it has the opportunity. The logical conclusion is that the biggest deformations extend along the middle of the surface and therefore produce large stresses. Moving the members toward the edges results in smaller deformations and the formation of extremely big cells in that middle area.

The inclusion of the load buckling factor as a penalty function was avoided here. The cell sizes were used because they can vividly show how the penalty functions affect the optimization process. Nevertheless, three solutions have their Load Buckling Factor value (marked as LBF next to the solution) to show that the extreme grid is unstable (LBF:0.83) but it can be used to read the intention. Furthermore, the more regular grid, restricted with penalty function, shows that it naturally has a bigger Load Buckling Factor Value (LBF:1.35)

In the best solutions we can see some oversized cells that are generally not acceptable in grid shell design. So we can try to do the optimization with restricted cell size (Figure 5.6 (b)). The process with this particular penalty function displays a similar convergence to an optimal solution after 330 generations, and again the similar pattern emerges, where the density along the edges is increased to stabilize the entire structure and result in less stress. As with the unrestrained solution, we see the formation of a *belt* between the concave and the convex part, thus completing the circle around the basis of the convex part (this method of *surrounding* the convex parts, i.e., stiffening it with increased cell density, will be also recognized in other experiments in the following sections). To check



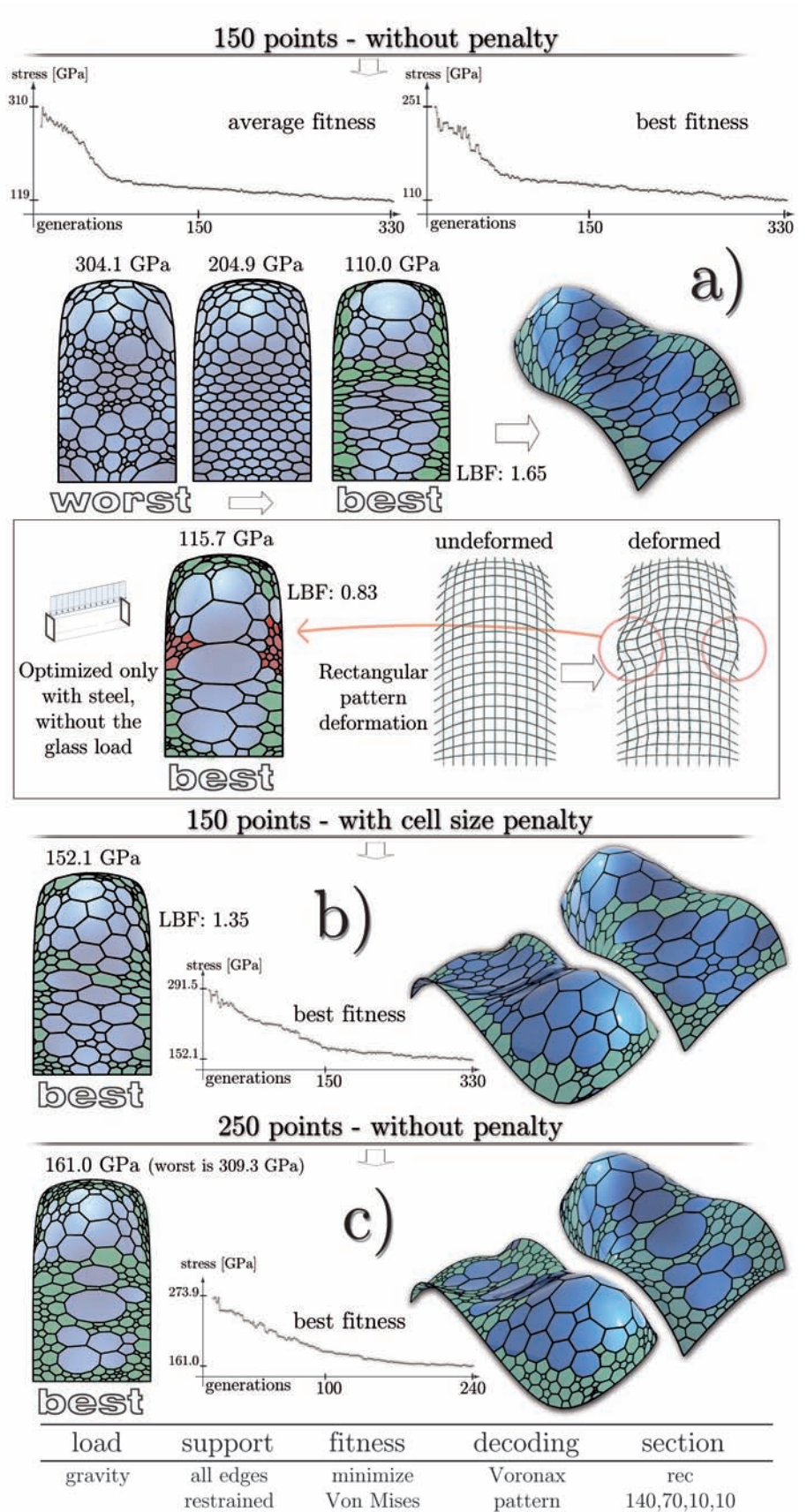
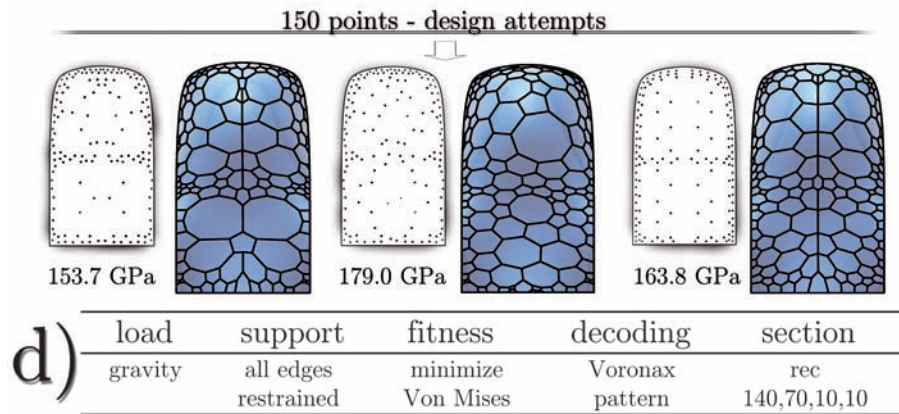


Figure 5.6: Analysis of a surface

this assumption, another optimization was carried out with more Voronoi seed - 250 (Figure 5.6 (c)). And looking again at the cells marked green, we can actually see that the creation of the *belt* zone repeats to stiffen up the structure. This belt is in this case a part of a *girder* formation, spanning the surface from left to right, making a sort of a secondary rib structure.

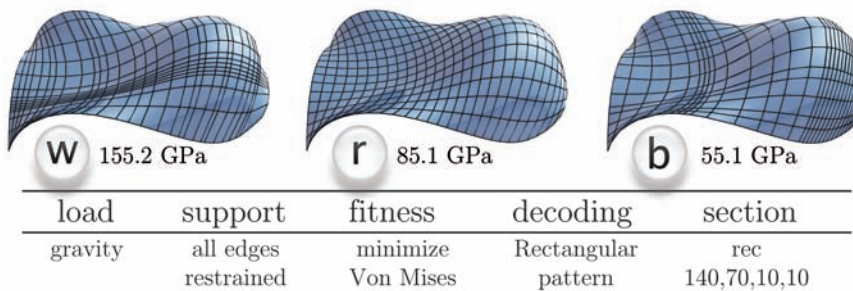
The information obtained can be used to create our own solutions, depicted in Figure 5.7, with Voronoi seed on the left and the resulting Voronax structure on the right of each solution. Designing



**Figure 5.7:** Voronax grids designed according to the *recommendations* from the optimization process

the structure, we can always bear in mind that we can go even below  $110GPa$  for the whole grid, and that we can (and usually must) deviate from this value in order to satisfy some other restrictions, like cell size or member lengths. Whatever the cell disposition is, when designing *manually* we always have to watch out for the direction of members, since they are best exploited when they are parallel to the forces acting upon them.

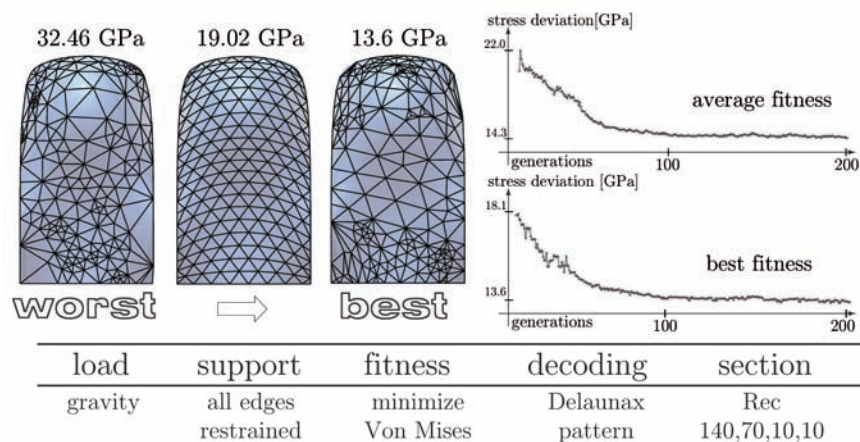
An interesting comparison can be made with the rectangle pattern (Figure 5.8) to show how the same principles which we obtained from the optimization process with the Voronax structure can be applied to different patterns. Even with very little rearrangement



**Figure 5.8:** Rectangle pattern, worst - regular - best solution

of structural members, while keeping the number of members the same, the gains are obvious, i.e., the stress in the entire structure decreases considerably. We can see that in the solution on the right the zone behind the convex part is stiffened up, as well as the crossover zone, where convex goes to concave. The longitudinal members are also rearranged and shifted a bit toward the edges, as the Voronax experiment *taught* us. And we got stress reduction in the entire structure from  $85.1GPa$  in a regular one to  $55.1GPa$  in the optimized one. The point here is that the movement of members in this quadrangular pattern is very restricted. Only complete lines could move, without the disturbance or rectangular member disposition (as described in 4.2.2). Still, this small alteration led to the 35% stress reduction, and the more freedom in pattern and member movement we have (like with the Voronax pattern) the greater the gains are.

If the optimization is performed with Delaunax pattern (relaxed Delaunay triangulation), we have the gains in statical efficiency, but the results are hardly interpretable. Figure 5.9 shows the optimization results with the same input settings (except for the pattern), where two Delaunax solutions have 525, vertically oriented, steel structural members, and the uniform triangulated one in the middle has 520 members. The optimization shows a good convergence, and the best solution that has 2.5 times less Von Mises stress then the worst generated solution and 50% less stress then the uniform grid. It shows larger cell density around the convex part and the concave part near the supported edge. However, as mentioned before and explained more in the next chapter, the Delaunay and Delaunax optimization results are hard to interpret, and that is one of the areas that should, and will be investigated in the future.



**Figure 5.9:** Delaunax pattern, worst - regular - best solution

**Deviation From Average Stress** In Section 4.3.2 a fitness function based on Mattheck’s *axiom of uniform stress* was explained.



There is no sense in going into the details of the experiments done with it, since the results are generally similar to the ones from the optimization process that minimizes stress. As mentioned, in most cases uniform distribution of stress leads to minimal stress solutions. In free form grid structures uniform distribution of stress shouldn't be confused with uniform distribution of structural members. With solid models (like in Klaus Mattheck's experiments), it is fairly easy to achieve a good design. A grid shell over a pre-defined free form surface is very restrained, and can never *evolve* into a state of uniform stress, and in most cases not even come close. That is why this fitness function is created to compare the results, and maybe to be used in some specific projects where uniformity is of particular importance. In Figure 5.10 there is a short optimization process that shows a similar tendency to the previous fitness function. All settings, except for the fitness function, are the same. Self-weight of the structural members and surface load of  $1KN/m^2$  are applied, nodes on all 4 edges are fully restrained and members have vertically oriented rectangular cross-sections: 140,70,10,10. The results show that density is enlarged near the edges (near the support) and around the convex part of the surface, as in the previous experiment. Load Buckling Factor of the best generated solution is 1.158.

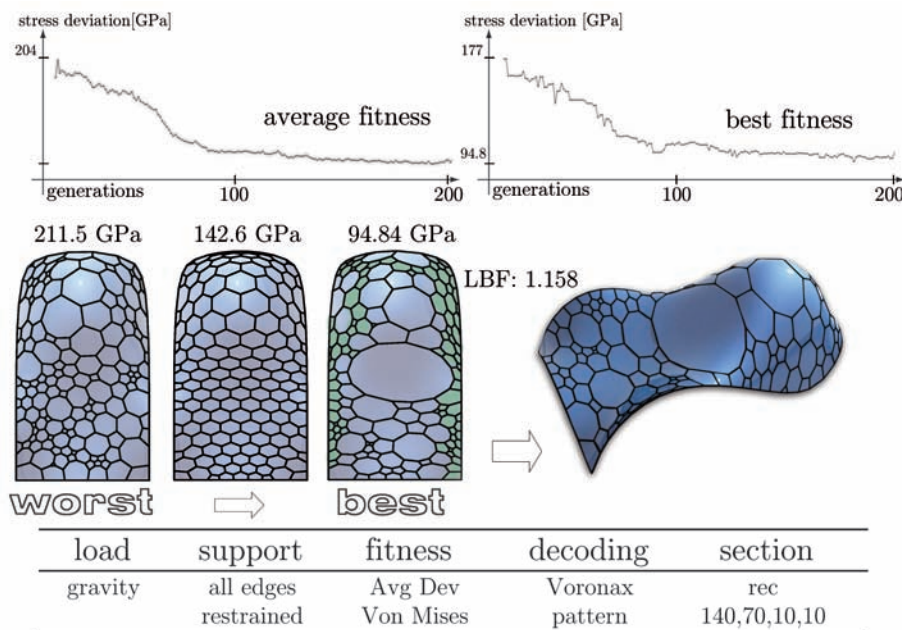


Figure 5.10: Deviation from average stress

### 5.1.2 Sum of Displacements

We can sum up all the displacements, from all joints, in one grid structure and try to minimize that value, as described in Section

4.3.2. In that way we can let the GAs optimization process search for the stiffest structure. Since stresses basically come from preventing deformations, the results from this fitness function are similar to the ones from the minimization of Von Mises stress. Figure 5.11 shows that clearly. The optimization process has the same settings, self-weight of the structural members and surface load of  $1KN/m^2$ , nodes on all 4 edges fully restrained and members with vertically oriented rectangular cross-sections. A similar pattern occurs, in which the structure is stiffened near the edges and around the convex part (marked green). The extremely large cell on the crossover from the convex to the concave part is a clear indication that the greatest displacements happen exactly at that point. We also learned from this and previous experiments, that we can solve the problem of large deformation areas through two different approaches. We can leave it as open as possible with the creation of bigger cells, like in this example. But, we can also stiffen up that area, with the creation of a *belt*, like in the example with the minimization of Von Mises stress.

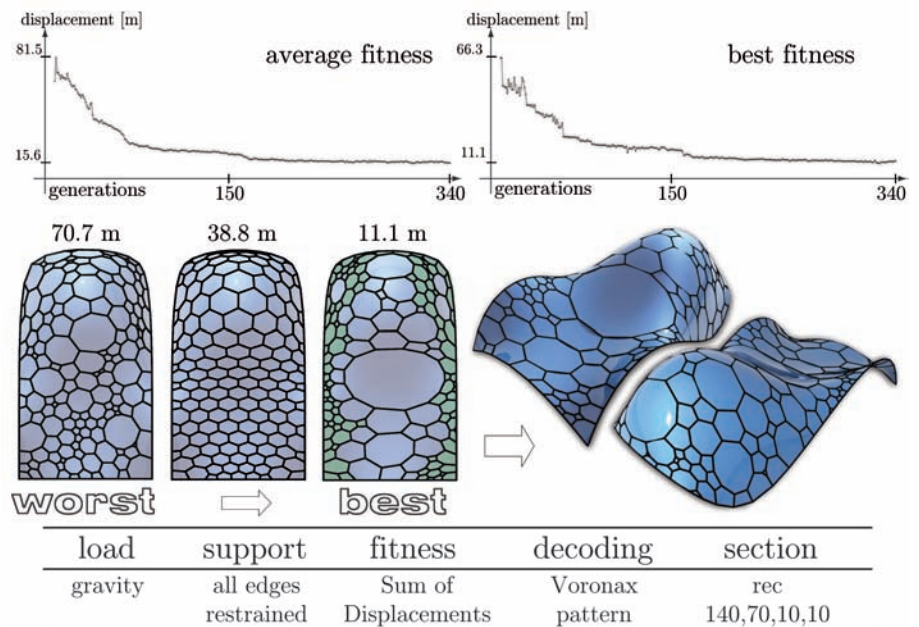
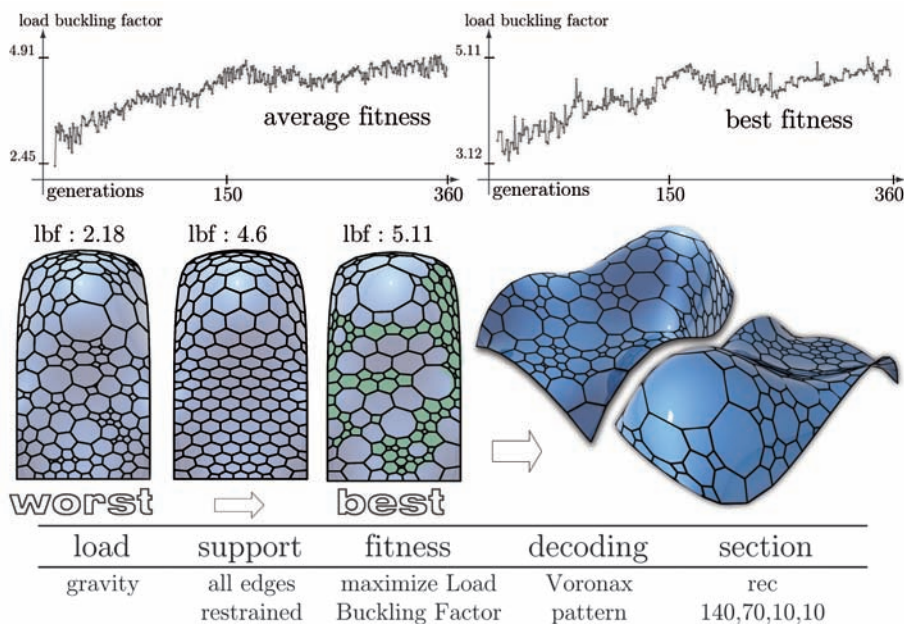


Figure 5.11: Minimize displacements

It is interesting to see the enormous differences in displacements, i.e., stiffness of the worst generated solution, regular solution and the best proposed one. The best offered grid shell has a more than six times smaller amount of displacements than one of the bad, randomly generated solutions, and almost four times less displacement than the regular structure. That shows that there is more than enough room in that design space to make a structure that is far more efficient than it is done nowadays.

### 5.1.3 Load Buckling Factor

The stability of grid structures is something more sensitive than other aspects, in terms of the geometry of the grid. Namely, very small differences in geometry can lead to very different load buckling factors. This means that small changes in member disposition can make the whole structure become unstable and buckle. The reason for this is that the buckling of one weak spot or even one single member can lead to a chain reaction that destabilizes the entire grid shell. In Figure 5.12 there are results from the optimization process with the same surface and parameters as in the previous examples, but with the fitness function that tries to maximize the Load Buckling Factor. Although a logical use for the Load Buckling



**Figure 5.12:** Maximize Load buckling factor

Factor is to set is as the penalty function in the stress or deformation minimization, here the opportunity is used to show that it can be used as a fitness function as well, especially for multi-objective optimization explained in the following section. The graphs in the figure show how *unstable* this process is. The goal is to achieve the largest factor possible (as explained in 4.3.2), and the curve in the graphs is expectedly rising, but its instability proves that very small differences in the generated solutions result in very different factor values. From this experiment (as from many others) it can be concluded that regularity is actually good for stability. In the best solution, the denser cells are marked in green to show that we can hardly detect any pattern or intention, as one does with other fitness functions. The differences between the worst and the best generated solution cannot be clearly stated, yet the latter has a load buckling factor which is twice as large and therefore a lot more stable. The

small difference in the fitness value (load buckling factor) between the regular structure and the best generated solution additionally proves that an uniform distribution of cells and members is good for stability, although the difference in values shows that the regular structure is not ideal and can be improved.

The settings in the FEM software used for this experiment were set to Modal Buckling. Since the fitness function is easily constructed and can include any setting that can be made in the FEM software, other options, like member buckling, can be incorporated into the optimization process and investigated. The analysis of fitness functions and the possibilities of their creation is, however, not the purpose of this research. The goal is to propose and describe an effective method for structural optimization of free form grid shells. Therefore, the construction of the fitness function is only a *branch* of that method and can be modified and constructed by the user according to the conditions of the specific architectural project.

In the results obtained from multi-objective optimization it will be shown how to combine this with other fitness functions. That would also be the advice on how to handle the results gained from the optimization with this fitness function - to combine it. In grid shell design we should determine some value, some indicator of stability (it doesn't have to be the load buckling factor, it can come from precise dynamic calculations), and then optimize the structure by keeping it stable, always above that value. From that we can easily conclude that the penalty function would be an ideal solution. Of course in order to achieve greater stability (not only keep it above some limit), we can turn to multi-objective optimization, the explanation of which follows.

#### 5.1.4 Multi-Objective Optimization

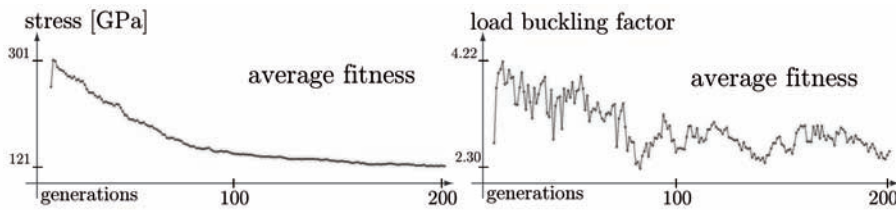
After the explanation of the Pareto optimum application in Section 4.5, some results will now be presented. We will see two examples of optimization processes with two objectives at once. In both cases it will be shown what happens to the second objective when we do a single-objective optimization and then how that changes with the application of the Pareto optimum.

##### Von Mises Stress and Load Buckling Factor

The combination of stress reduction and stability enhancement is probably the most important goal to achieve in structural design. What makes it a difficult task is that the two objectives in some cases do not go along with each other. To prove this statement, an experiment was made with the single-objective optimization where the goal was to minimize the Von Mises stresses, and at the same time to check the alterations in the load buckling factor throughout

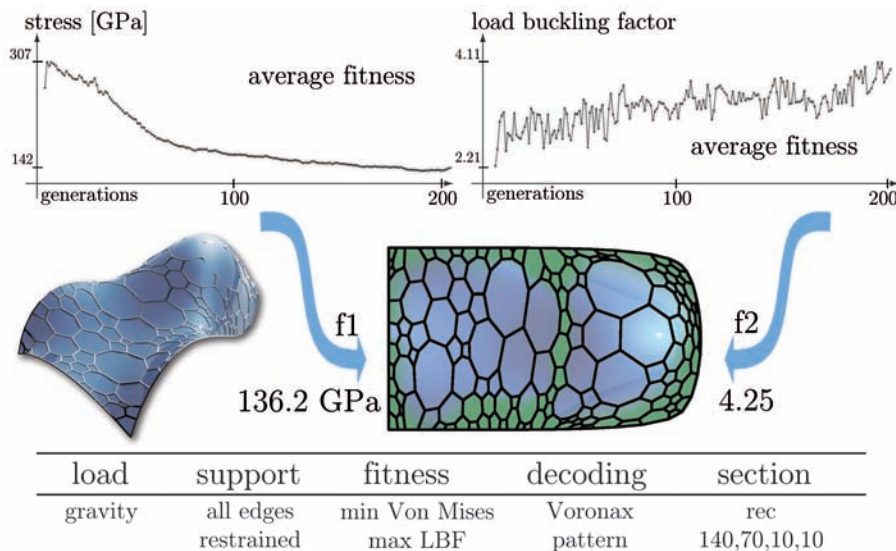


generations. The load consists again out of self-weight of the structural members and  $1KN/m^2$  surface load. Structural joints along the edges are restrained from all movement and rotation and steel members have vertical rectangular cross-section: 140,70,10,10. In Figure 5.13 there are two graphs, showing the progress of the GAs optimization process. As seen before, the Von Mises stress decays expectedly, with a good convergence, but on the second graph we see that the load buckling factor decays as well.



**Figure 5.13:** Progress of the average stress and average load factor in a single-objective optimization

This is exactly the opposite of what we want to achieve. In Figure 5.14 we can see what happens with the application of the Pareto method. As mentioned before, the load factor produces



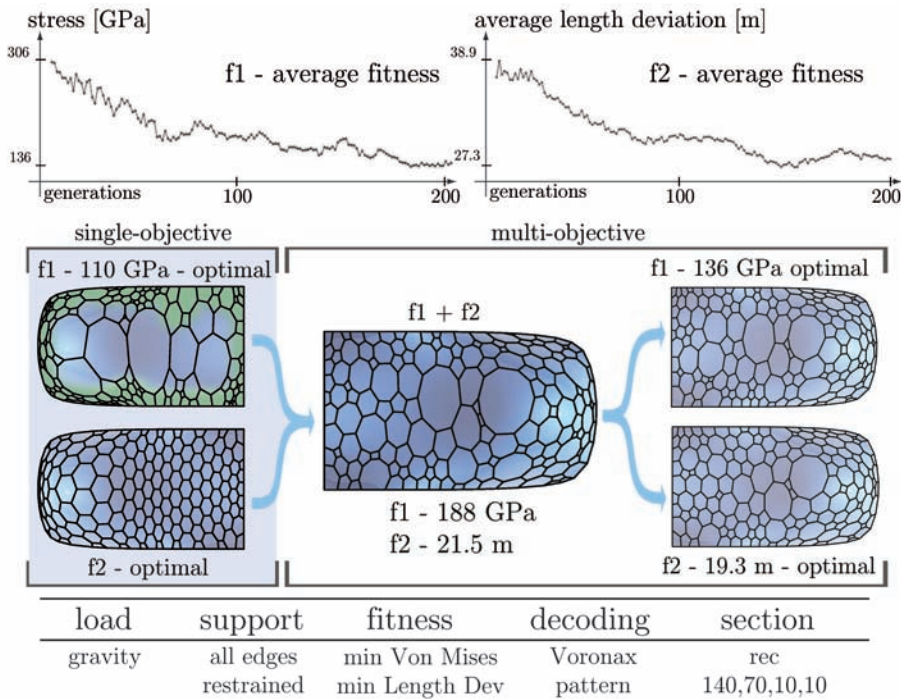
**Figure 5.14:** Two-objective optimization, Von Mises stress and Load buckling factor

instable diagrams anyway, since the slight changes in structure can result in big differences in load factors. However, in comparison to the single-objective optimization, the ascending progress of the load factor is clearly distinguishable here. Also, there is a noticeable trade-off in the minimal amount of stress after 200 generations, i.e., it goes from  $121.3GPa$  in the single-objective optimization to the  $136.2GPa$  in the multi-objective one. But it is a small price to pay

to gain a load factor rise from 2.2 to 4.25. It is always up to the designer to choose the trade-off between several objective functions. The important thing is that the automatized GAs method, with the help of Pareto optimum, offers a group of solutions that are good in both objectives. In Figure 5.14, one of the solutions is showed, that has one of the best fitness values considering both fitness functions. It has a distinguishable edge-stiffening pattern that we encountered in the experiments with the minimization of Von Mises stress, but with more regular cell sizes, that enhance stability and the load buckling factor. With the combination of single-objective and multi-objective optimization we can therefore prove the *intentions* of the process, by seeing it as a merger of two different optimal solutions. That leads us to the next experiment, where the merger is geometrically more obvious.

### Von Mises Stress and Average Length Deviation

How the geometrical restrictions, like member length, cell size, etc., are best applied with the use of penalty functions has been explained and demonstrated. However, we can use the Average Length Deviation fitness function in a multi-objective analysis to demonstrate the effectiveness of the Pareto method and to graphically see how the solution of a multi-objective optimization can be a merger of two single-objective optimums. In Figure 5.15, the graphs show a constant decay of both fitness values, which is good, since our goal is to minimize both fitness functions. Under the graphs, on the left, we see the results of single-objective optimization processes with the two fitness functions. We remember that the minimization of stress shifts smaller cells to the edges and around the convex part of the surface. The Average Length Deviation fitness function assigns the best fitness value to the regular (uniform) structure. After the process offers us thousands of solutions we can start the *design game*. On the right of the figure we see two solutions of our multi-objective optimization, each with a best (minimal) value for one of the two fitness functions. However, in both cases when the solution has optimal value according to one fitness function it is not satisfactory according to the other function. That is why we can chose a solution that tries to minimize that trade-off between functions, i.e., that is sufficiently good in both functions, like the solution shown in the middle of the figure. In this example, the individual was picked *manually* from one of the latest generations, by comparing the fitness values. The solution can be a compromise, and we get to decide how much of one objective we want to *sacrifice* in order for the other to profit.



**Figure 5.15:** Two-objective optimization, Von Mises stress and Average length deviation

## Conclusion

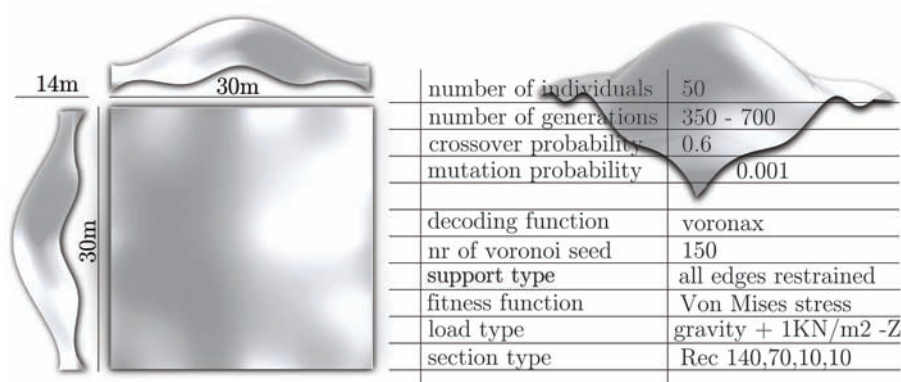
It is important not to observe these examples in terms of real building structures. Every real project will have many boundaries that have to be respected, starting with the grid pattern. The examples are here to show that within those boundaries there is always some space left for optimization. Within those boundaries, the multi-objective approach finds the solution that obviously does converge to an optimal structure according to more objectives. The trade-off depends on the compatibility of functions. In both of our examples, they were incompatible, therefore resulting in a strong trade off. In the first example we saw that stress increases from the optimal  $110\text{GPa}$  to  $136\text{GPa}$ , or even to  $188\text{GPa}$  in the second example. It was shown that the method works, and the specific project requirements can guide us to make choices that will finally influence our grid structure in the way we want it. The GAs are there to point us to the optimal solution according to the settings that we have chosen, and the engineer can then use the information he has to design an optimal structure.



## 5.2 Patterns

This section will focus on the explanation of the general use of Voronax structures. Namely, it will be demonstrated how Voronax can be used as a basic pattern in GAs, in order to show us what the optimal structure should look like. After the optimization process, we can use all the generated individuals to see how the grid shell evolved and to understand its behavior throughout generations. The geometrical development of the grid, i.e., the change in cell density over the surface in the course of *time* is meant by *behavior*. The density information can then be used as a guide to create individual solutions. Those solutions can follow the guide lines provided by the Genetic Algorithms, but how far we want to follow them is a personal choice, relative to fitness, i.e., material, i.e., cost. The pattern is something that is left to the designer to decide upon, since different density can be produced with infinite number of various pattern combinations. Additionally, relaxation process with controlled tension factors can be used to adjust the density as *advised* by the GAs optimization process.

It will be shown how the orientation of the connected structural members has a great influence on the grid structures with regular patterns. Some methods of the structural path generation will be proposed. It will be demonstrated how they can be combined with the density information we get from Voronax optimization to create statically efficient grid shells. All the examples and comparisons will be there to point out different possibilities. Since there is an infinite number of pattern combinations, there is an infinite number of design possibilities. However, Genetic Algorithms and the member orientation methods can show us the appropriate way to implement these possibilities.



**Figure 5.16:** Surface and parameters

First, we start with the  $30 \times 30m$  surface depicted in Figure 5.16, with standard parameters (shown in the table) that were used with most of the experiments. As in the previous section, a gravita-

tional (dead) load is used, i.e., the self-weight of the steel structural members and a  $1KN/m^2$  surface load (glass). All members have vertically oriented rectangular cross-section and all the joints on the surface edges are restrained from movement and rotation in all directions. Each generated grid shell has between 455 and 465 structural members. An optimization process offers its best solution in form of a Voronax structure. We will use that to create our own Voronax grid shells, and then use the same knowledge to develop grid shells with different patterns.

### 5.2.1 Start With Voronax

The results of the GAs optimization process with the Voronax pattern, gravitational load (members+glass) and joints on all four fully restrained edges are to be found in Figure 5.17. The graphs show a very nice convergence of the best and the average solution. Again, the best generated grid ( $83.6GPa$ ) has almost two times smaller amount of stress then one of the randomly generated grid shells in the first generation. It can also be seen in this case that the regular solution is very bad ( $138GPa$ ), and that it has a fitness value very close to the worst generated grid ( $148GPa$ ).

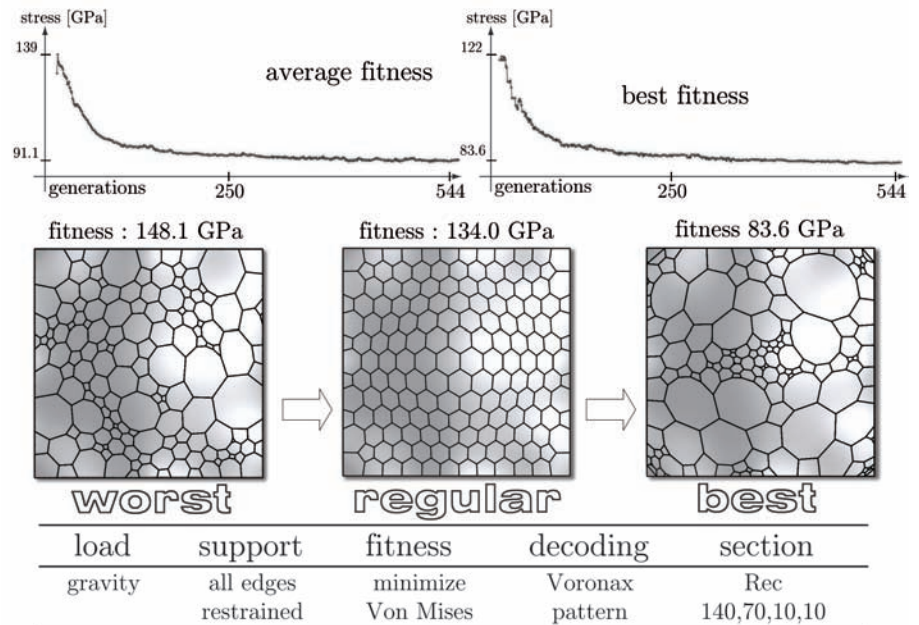


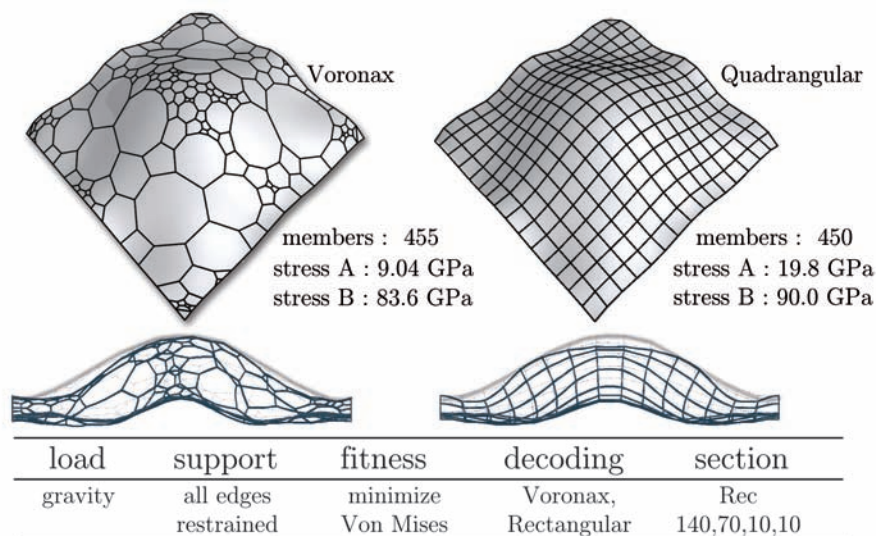
Figure 5.17: Voronax pattern

Intuition would imply that for a symmetrical shape and a symmetrical load the optimal structure should also be symmetrical. Genetic Algorithms in this experiment (repeated several times) show the intention of making a slightly asymmetrical solution. Namely, the *intention* of the algorithm will be discussed in the following



section, but here it has to be said that a fact that an asymmetrical solution was optimal cannot be explained with certainty here. One explanation is that the algorithm would eventually converge to symmetrical solution. However, when applied to different patterns (as shown in the following section with the triangular grid) making an asymmetrical density disposition, as advised by the algorithm, also generates less stress in the structure. This question is therefore an interesting subject for research and it is left open for future investigation.

The opportunity will be used here to compare this solution with a quadrangular grid shell. If we generate it in a way that it has similar number of structural members (as depicted in Figure 5.18), and we use the same load and restraints, we can compare the results.



**Figure 5.18:** Comparison of Voronax and quadrangular grid shell

First we will apply only the self-weight of the structural members and mark the results as *stress A*. In the lower part of the figure, deformations of both structures can be seen. In the regular structure, the largest deformations occur in the middle. In the Voronax structure, thanks to the greater cell density in the center and the formation of the cross-formed bearing zones, deformations in the middle are minimized. That leads to the reduction of the total amount of stress in all structural members from  $19.8\text{GPa}$  to  $9.04\text{GPa}$ . However, if we add the surface load of  $1\text{KN}/\text{m}^2$  to the load of the steel structural members, we see a small difference in the generated amount of Von Mises stress,  $83.6\text{GPa}$  and  $90.0\text{GPa}$  under *stress B*.

This experiment shows how it cannot easily be determined which pattern is better, i.e., it cannot be said that the Voronax pattern will always show better performance than some other, *regular* one, or vice versa. Experience showed that the greater the surface curva-

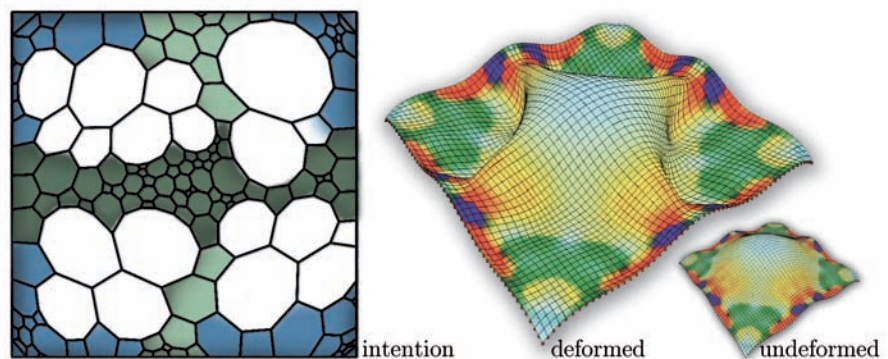


ture and the complexity is, the more Voronax is effective. It cannot be forgotten that the quadrangular grid has 4-member joints which makes it additionally stiffer than the Voronax. In spite of that, such as in this case, the change in density can make Voronax more efficient. If we had a flat rectangular surface, it makes no sense (besides the pure design justification) to apply Voronax, since the rectangular pattern would respond much better. However, with strong curvatures, regular patterns would have to be heavily distorted in order to be optimized, thus disturbing the smooth structural paths and being unacceptable optically. With the Voronax pattern and its polygonal *cell structure*, we have additional freedom to adjust the density of the cells to fit the demands.

This example demonstrates how the different load combinations can also influence the efficiency of the structure. As a matter of fact, all other factors, like support combination, material, cross-sections, etc. can influence the stability and the generated stress in the grid. The multiple connected small members in the Voronax structure can be combined into one member for example, which would be in its favor. This comparison should therefore not be taken for granted, but only as an example of how a comparison can look when the grid is designed. What is certain is that if we choose one pattern, we can use GAs to generate statically optimal geometry, and if we don't know which pattern to choose, we can use GAS to see what the optimal solutions with different patterns are, and make our decision according to the differences.

### 5.2.2 Recognizing The *Intention*

When we have the results, we have to realize what it is that makes the best solution better than the others. Instead of taking the results for granted, it is better to use them as a guide to alter the design to fit our aims. In Figure 5.19, on the left, we can see the basic *intention* that we can extract from the previous experiment. Here, as in the large number of examples with similar surfaces, we

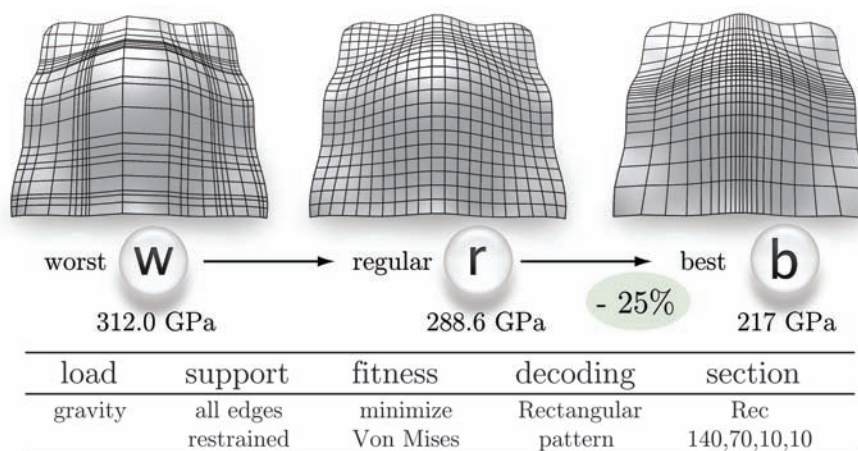


**Figure 5.19:** Intention of the optimization process

can recognize the stiffening of the central part. It is clear that the dense cells extend from the edge to the edge to create a central *spine* (marked dark green). Another set of dense cells extends from the center in two other directions to form a *cross* (light green), thus following the convex parts of the surface and making a *primary bearing zone*. Since the nodes on the edges are restrained, greater density is generated there (marked blue) with members perpendicular to the edges to take the load in the best possible way. Naturally, the GAs do not miss the chance to stiffen up the corners, since their naturally rigid shape is suitable for stiffening the whole structure. If we look at the deformation visualization (on the right side of the same figure), it is clear that the GAs process generated larger cells in the areas of the greatest deflection. Those areas should therefore be lighter and they should rely upon the *cross*, i.e., the primary bearing zone.

### Check With the Quadrangular Grid

In order to check if the intention is correct, an optimization process with the rectangular pattern is made. In Figure 5.20 we can see that it follows the similar logic. The self-weight of the steel struc-



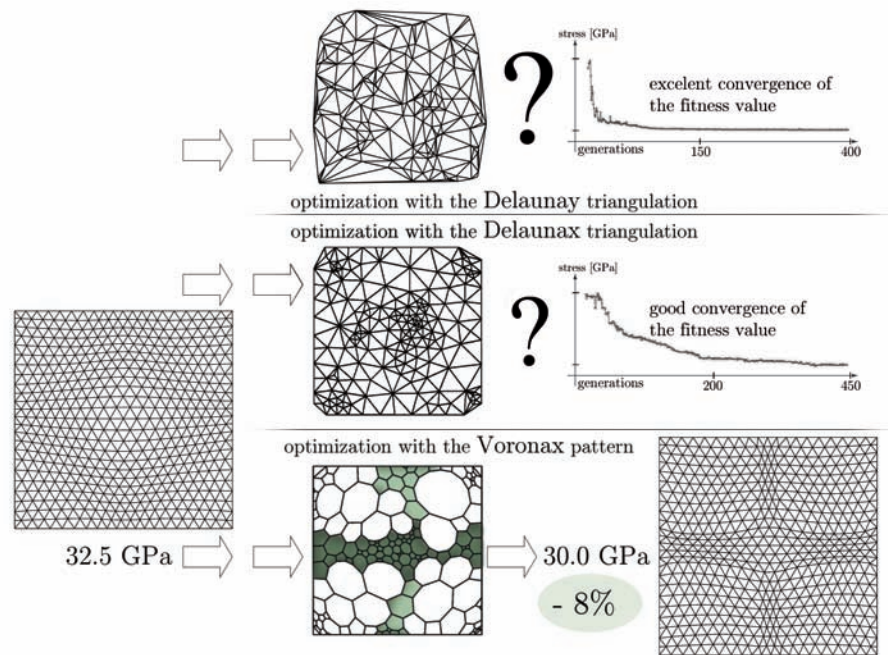
**Figure 5.20:** Rectangular pattern optimization

tural members and a  $1\text{KN}/\text{m}^2$  surface load (glass) is used again. All of the structural members have the same vertically oriented rectangular cross-section and all the joints on the surface edges are fully restrained. Since the quadrangular decoding is restrictive, the number of structural members in all generated solutions is the same. As it can be seen in the figure, the best generated solution stiffens up the middle convex part, making a *cross*. Even with the very restricted movement of the members (only whole rows can move, as explained in Section 4.2.2), substantial reduction of Von Mises stresses is achieved. It is consequently our decision, based on the aims and possibilities, how much to deviate from the regular struc-

ture, moving toward the optimal one. In any case, the direction that we ought to follow is clear.

### Check With the Triangular Grid

We can perform optimizations directly with triangulated grids, using Delaunay and Delaunax triangulation as explained in Section 4.2.2. However, they end up with a solution, the intention of which cannot be *read*. Although the performed experiments converge nicely, there is no logic which can be derived from the optimal solution. The optimization of this kind cannot give *ready to build* solutions. It serves as a support tool, a *recommendation*. We should be able to extract a principle from it that can be used to design an efficient structure. If that principle, or pattern of behavior, cannot be extracted and abstracted, then the optimal solutions from the optimization process cannot be properly used in architecture.



**Figure 5.21:** Influence of the member orientation on the total generated Von Mises stress

The upper part of Figure 5.21 is the result of an optimization process where the Delaunay triangulation and Delaunax were used as a chromosome decoding method (with all other settings staying the same as in the Voronax optimization). The graphs show the progress of the average fitness value for 400 and more generations, and it can be easily seen, for example, that with the Delaunay the optimization process converges very early. Almost 350 generations of mutation and crossover didn't change the optimal solution. The Delaunay grid shell, depicted from the top (next to the graph), is one of the best generated solutions. But the grid in the picture

obviously cannot be an acceptable solution from the production and the design point of view. Much more important, we cannot learn anything from it, i.e., we can't extract information that we could use for the grid shell design over that same NURBS surface. Delaunax solution would be more acceptable due to its relaxed geometry. Even though some pattern of behavior can be recognized, like the slight stiffening of the corners and the middle of the surface, it is not enough to abstract a principle. It is very important for triangular grids to control the design *manually* because the structural paths formed by the grid members have to be smooth to ensure a better distribution of forces. There is a possibility that a combination of penalty functions that control aesthetic aspects of the grid can be implemented to improve the optimization process, but that is a complex matter that will be explored in future research.

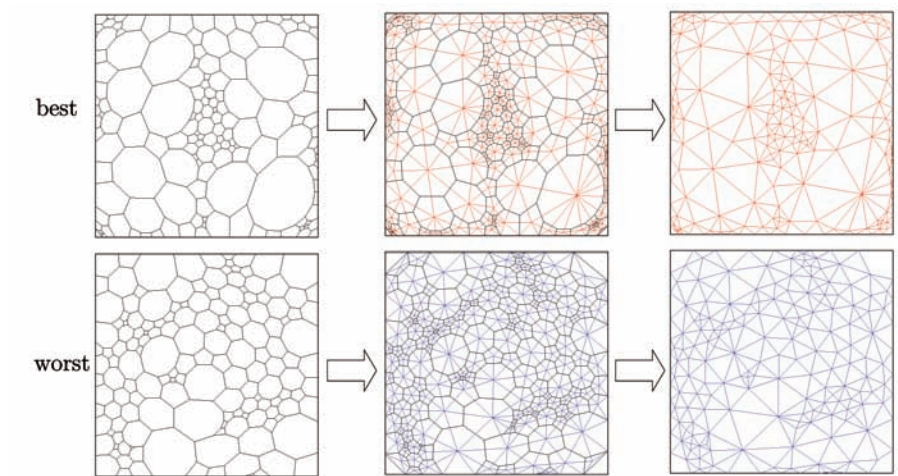
There is a better approach in the lower part of the Figure 5.21, using the Voronax structure and the intention we recognized from the best generated solution. If we use the relaxation techniques described earlier, and give the members in the middle a larger tension factor, the relaxation process will pull the triangles toward the middle to form a *cross*. This, very small rearrangement of the structural members, can decrease the total amount of generated Von Mises stress in the structure by 8%. Considering the shape of the surface, from the designer's point of view, it could be said that the optimized structure looks even more interesting. Considering the manufacturing possibilities, the solution has 1-3m long members which is absolutely acceptable. This optimization has a relatively small gain, but it is the minimal amount of the rearrangement that could be done, to take the regular grid structure and pull the members toward the middle a little. Additionally, we are distorting the rigid, straight structural paths that triangular structure has, and later it will be shown how that can influence the solution greatly.

At this point it has to be said that the idea of taking an optimal Voronax grid and making a triangular grid directly out of it also doesn't bring good results. Figure 5.22 illustrates the best and worst solution generated from the optimization described at the beginning of this section. Extracting a triangular grid from them directly gives neither optically acceptable nor statically efficient structure.

### 5.2.3 Design Our Own Grid Shell

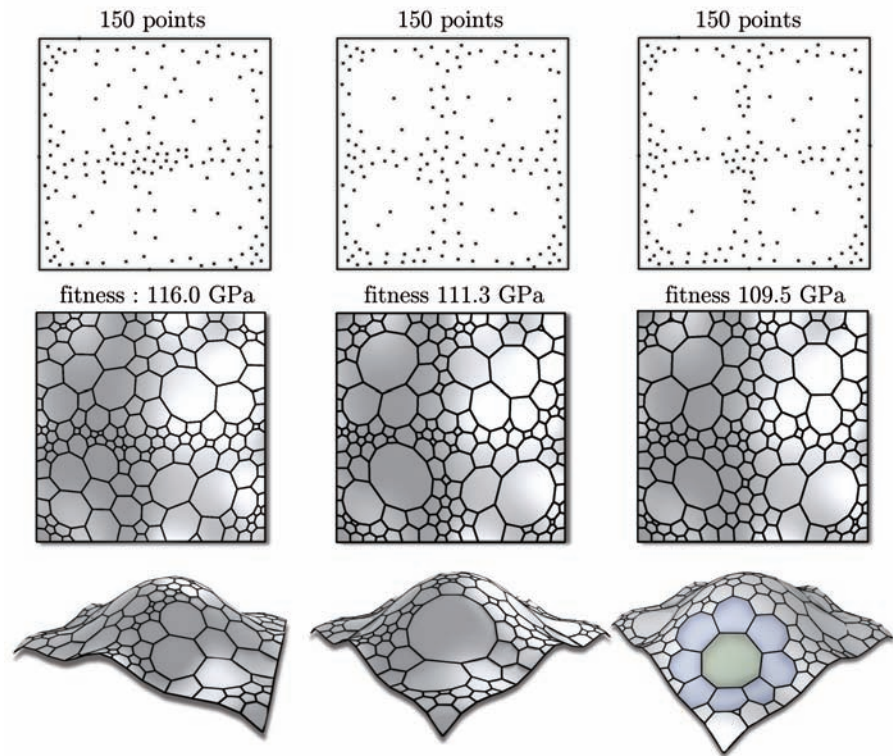
Now that we know the logic that we have to follow in order to gain efficiency, we can make our own Voronax structure. The examples in Figure 5.23 show three manually created structures, using the information gained from the previously described optimization process. The structures were generated by simple manual definition





**Figure 5.22:** Direct extraction of a triangular grid from the Voronoi solution doesn't give acceptable results. A principle has to be extracted and then applied

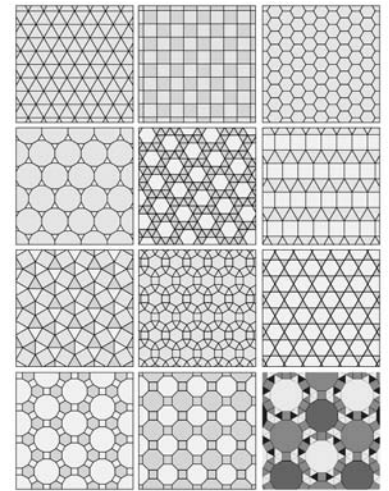
of the points on the surface, as shown at the top of the figure, keeping in mind that the density has to follow the *spine* and the *cross*. If we look at the fitness (total stress amount) values (109 – 116 GPa) we can see that it is still much better than the uniform solution presented at the beginning (134 GPa), but it is also a lot worse than the best offered structure (83.6 GPa). This shows that we are



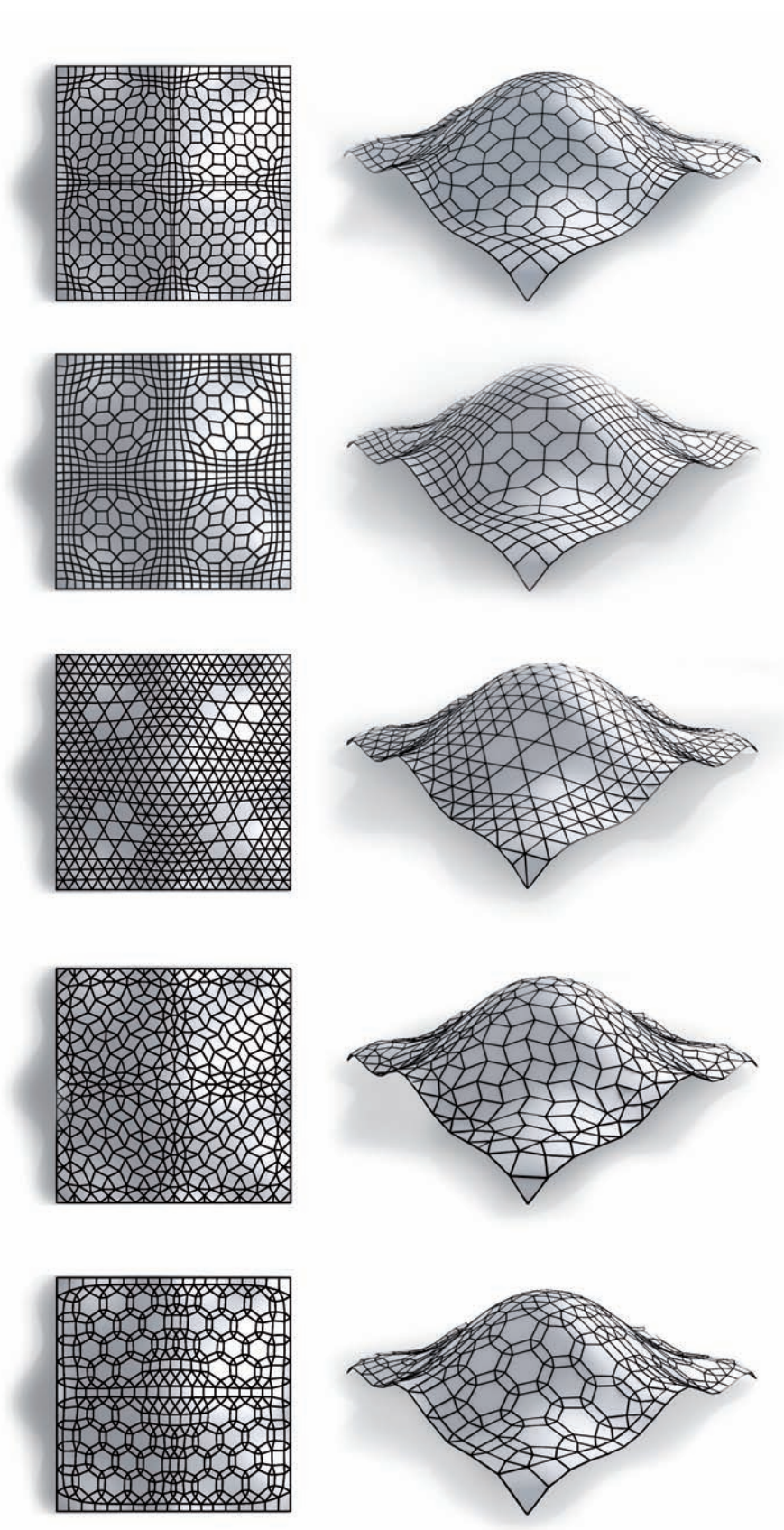
**Figure 5.23:** The *intention*

on the right track, i.e., that we *read* the intention properly, but that we can go a lot further in optimizing it. In any case, the conclusion remains the same. With the change in the disposition of structural members and adjustments of their density, we can optimize the structure greatly. Instead of using penalty functions to avoid oversized cells, we can also manually insert new points, or move existing ones, thus playing easily with the density (something barely possible with the regular patterns). Talking about patterns, there is basically an infinite number of polygon combinations that can be used to form a grid structure. Using the basic  $xy$  to  $uv$  transformation, we can apply any of the 2D tessellation techniques, like the ones depicted in Figure 5.24 for example.

Using the same information obtained from the optimization process, few examples will be used to try to recall the fact that the pattern is not a decisive part of a statical optimization. Whatever geometrical paradigm we choose, it can be optimized. This can be done by deletion or movement of the members, as well as with the controlled grid relaxation, i.e., the usage of different tension factors. In Figure 5.25 there are manually generated solutions with patterns the members of which are simply deleted, but also solutions where the relaxation is used to adjust the structural density so it can fit into our *spine* and *cross* paradigm.



**Figure 5.24:** Some examples of the large variety of 2d tessellation possibilities



**Figure 5.25:** Different patterns



### 5.2.4 About the Orientation of the Structural Members - Paths and Guide Lines

It is clear in the Voronax pattern that it is the density which is adjustable to get to the statically optimal structure. Furthermore, if we look at the regular patterns, it becomes clear that the direction of the structural members is another very important factor in the grid shell design. (That the direction of structural members can also play an important role in the Voronax structures will be demonstrated in the next section.) A very simple comparison of the two vertical triangular grids, depicted in Figure 5.26, shows that, in this example by changing the orientation of the triangles, we have 8% stress reduction.

Quadrangular and triangular grid shells have their cells usually connected in such a way that they form distinguishable *paths*. Those paths are basically a system of interconnected polygonal curves, that ensures the rigidity and greater stability of the structure. If we observe our grid shell from that perspective, we can think about new ways of increasing statical efficiency. We know that the bending of the grid shell structural members creates larger stresses than the axial forces. Therefore, our goal is to minimize the bending moments and we can do that by orienting the members to be parallel with the main forces. In the example of the triangular wall, the more efficient one has vertical *paths* that are parallel to the gravitational load (marked red). In this way, axial stress is enlarged at the cost of the bending stress, which then results in the smaller total amount of Von Mises stress.

With the vertical wall we can determine the direction of the members intuitively. With free form shapes it is not always clear how to define those basic paths in the grid.

The distribution of forces is too complex to be simplified easily, and it depends on many other factors (like the position of the bearing points and the distribution of loads, for example). But in order to optimize our grid structure by changing the orientation of the structural members we can use the principle stress trajectories offered by an FEM software to help us define our structural paths. To avoid a large digression, in order to obtain the trajectories we will analyze our surface as a shell structure, supported on all four edges. NURBS surface is first transformed into a mesh. Material is set to concrete, thickness to 10cm and the FEM analysis is performed in Sofistik (commercial FEM analysis software).

The entire experiment is depicted in Figure 5.27. In the upper half we can see the trajectories of the maximal Von Mises stress, calculated with Sofistik. It can be seen that a smooth paths can be extracted from the directions of the maximal Von Mises stress. This is depicted in the figure, on the lower right quarter of the



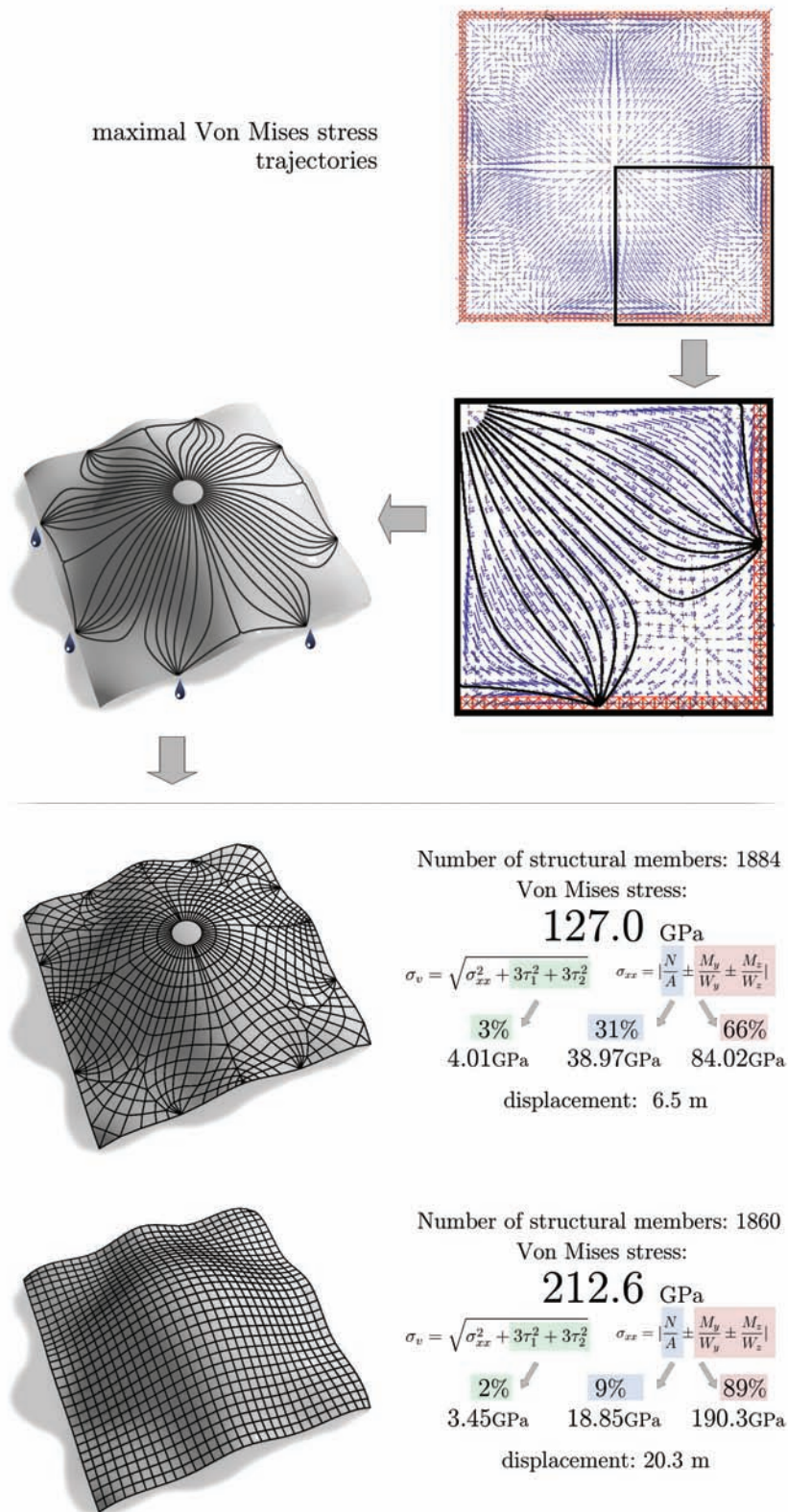
**Figure 5.26:** Influence of the member orientation on the total generated Von Mises stress

surface. Since the surface is symmetrical with respect to the  $x$  and  $y$  axis, this quarter is reflected twice in order to get a full path grid, as shown in the figure.

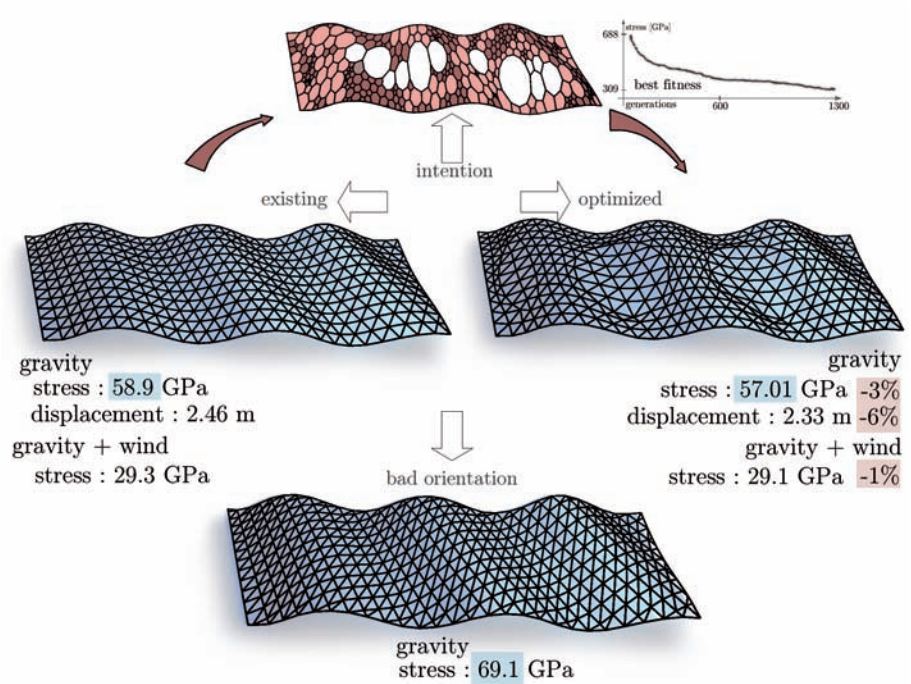
This information can be used to design the grid, as the one depicted in the lower part of the figure. We can then compare that structure with the uniform quadrangular one. For comparison purposes, the regular quadrangular grid was generated with the similar number of structural members and with the same, vertically oriented, rectangular cross-section (140,70,10,10). As in the previous experiments, the self-weight of structural members and  $1\text{KN}/\text{m}^2$  glass load is applied. All the structural joints on the edges are fully restrained from movement and rotation.

It can be seen that the total amount of Von Mises stress is almost twice smaller in the optimized structure. But, what was attempted here was to increase the stress generated by axial forces at the cost of the stress generated by bending, in order to decrease the total amount of Von Mises stress. We can test if this was the case by calculating the portion of the Von Mises stress generated by normal (axial) forces ( $N/A$ ) and the portion generated by the bending moments ( $M_y/W_y, M_z/W_z$ ). If we do that, we can see that, in the regular structure, the bending forces cause 89% of the total stress, and the axial forces only 9%. With our optimized structure, the portion of the stress generated with the axial forces rises to 31%, causing the total stress to be much smaller. Von Mises stress depends on the member cross-section and this distribution of stress can look differently. But since both grids are generated with a similar number of structural members, that have the same cross-section, this comparison between two structures can be made in order to show the increment of axially generated stress in comparison to bending stress.

This was an example of how the orientation of structural members can influence the statical efficiency of the grid shell. These methods can be combined with the different cell density in order to gain statically optimized structures. That has to be carefully done, and with regular structures there is usually a significant trade-off between those two aspects. This trade off can be nicely seen in the analysis of the portion of the Westfield Mall roof in London [22]. In Figure 5.28, on the left, there is a triangular structure as it was built in 2007. This orientation of triangles proved to be very good. Underneath the existing solution, marked as *bad orientation*, we can see how rotation of triangles can significantly increase the generated stress. The main problem we have in this case is that, if we want to change the density of a triangular structure, we have to distort the structural paths. In the same figure, at the top, we can see a result of the GAs optimization process with the Voronax pattern. Smaller cells are painted with 2 shades of red to mark the intention, extracted from the best offered solution after 1450 gener-



**Figure 5.27:** The use of principal stress trajectories for the statically efficient design of free form grid shells



**Figure 5.28:** Portion of the Westfield Mall roof in London, possible optimization

ations (72.500 generated individual grid shells). That information is used to relax the structure and get an optimized solution depicted on the right. In order to transfer the intention from the Voronax optimal grid to a triangular grid we have to be creative and combine the knowledge we have. If we simply made the triangles denser in the areas where the Voronax cells are smaller, that would heavily distort the triangular grid, and *destroy* the paths, thus increasing the generated stress instead of decreasing it. It can be seen that the bigger cells are generated in the middle and the smaller cells near the support and across the concave part of the *waves*. Having that in mind, a uniform triangular structure is relaxed and resulted in the grid shell depicted on the right side. The density disposition in this solution is good, but the paths are changed, and that is the reason why the gain is only 3%. However, the gain is 18% in comparison to the structure with differently oriented triangles. So, although the choice of the uniform triangular structure proved to be good for this particular example, it can be seen that in the design process the change of paths and density can lead to substantial differences in statical efficiency.

It is important to realize that it makes no difference if we got 3% or 10% less stress in this experiment. Since every project is unique, this doesn't mean that with some other surface and support combination we couldn't get 20% stress reduction. All the tests are here to show that the change in the density and the structural paths changes the overall statical efficiency, and more importantly, it is

shown *how* to change them, in order to optimize the structure. The proposed methods in this research are not there to give a ready, best of the best solution. They are there to help design the best grid shell by choosing and combining different patterns and arranging them in a statically efficient way. This research shows an efficient method for this purpose, keeping in mind that every project is unique, and that it can be done in many different ways, respecting different constraints.



### 5.3 Load

Our basic task in structural engineering is to design a structure that will resist all the forces acting upon it. Those influences come from the self-weight and the environment. Whether they represent the snow, wind, thermal dilatations, earthquake, etc. they can all be represented as spatial forces applied as *loads* in the FEM static analysis. Single structure has to be tested for the large number of different load combinations. According to the shape and environmental conditions, an engineer usually tries to think of the worst case scenario, i.e., the worst possible combination of loads that could occur at the same time. Using the knowledge and experience gained from many tests, an optimal structure that responds adequately can then be chosen. The comprehensive thinking behind this process is something that we still cannot simulate with the use of computer, but what we can do is offer an optimal solution for each of the load combinations that an engineer wants to consider. We can automatize something that they would generally do manually (one step at a time).

In Appendix A it can be seen that, in our GAs application, user has to define a load combination (Load Type) applied on the structure. For the research, large number of loads were tested, since their definition and expansion in the code is easy. Any combination of a dead and a live load can be applied, but in order to show the effects of the optimization process, we will try to keep it simple and restrained. It is important to see how the optimization process converges toward different results according to the single loads. All optimizations in this section are done over the  $30 \times 30m$  surface, shown in Figure 5.29, with the basic settings shown in the table. The only thing that will be changed is the load type, in order to see how it effects the optimization process. It can be seen that

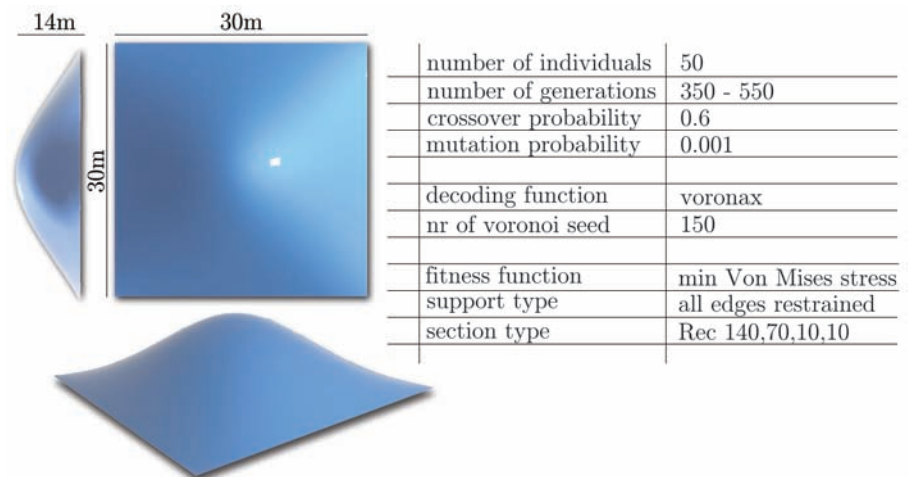


Figure 5.29: Surface and parameters

the fitness function used is the Sum of all Von Mises stresses in the structure. The cross-section used is rectangular and support is along the edges of the surface. In order to show only the effect of different loads, without the interference of other variables and factors, no restrictions (penalties) are used and therefore some of the solutions will have oversized and undersized members and cells. The curvature of the surface is very small, and proportionally, the profits from the optimizations will not be as big as in the other sections.

### 5.3.1 Case 1 : Gravitational Load

We start with the basic, vertical, gravitational load. Each member in the structure has its own weight, calculated with the help of its cross-sectional surface  $A$ , length  $l$  and material specific weight  $\rho$ , which is in this case  $\rho = 75KN/m^3$  for steel. The dead load of steel structural members is then  $q = \rho Al[KN/m]$ , and the simplified scheme of the member load is shown in Figure 5.30. In Appendix C the complex algorithm of cell definition is described, and that information is used to calculate the surface of each cell and apply load over it. In this first example, the load applied is  $1KN/m^2$ , and it is transferred to the nodes as shown in Figure 5.31. For each cell in the structure the center point is calculated. Part of the cell area is then assigned to each node and multiplied with the surface load.

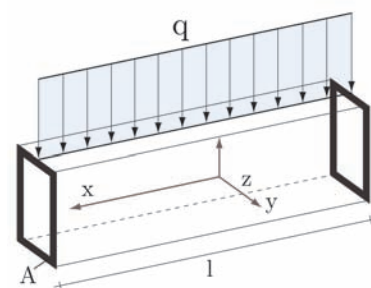


Figure 5.30: Linear load

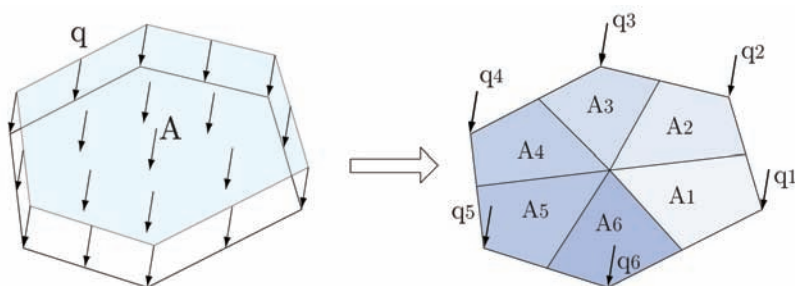
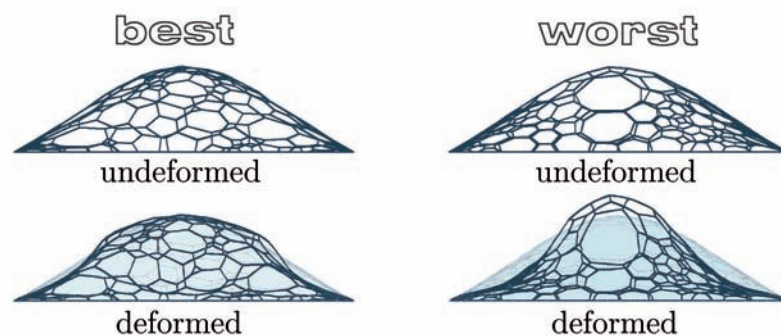


Figure 5.31: Surface load

**Results** Finally we get to the first results of the optimization process (shown in Figure 5.33), for the gravitational load applied on our predefined surface. We can see that there can be several different optimal solutions according to the input parameters that we set. As always, we start with one of the bad solutions and the regular one so we can compare their fitness values to the optimized structures. Beneath them, on the left, we can see the result of the first test, done only with the gravity load from steel members. One of the best offered structures shows a clear intention, i.e., dense cells



in the middle and in the corners. In this experiment, the best solution had 4 times lower stress than the uniform hexagonal solution. The solution is optimized only with the weight of the structural members, but here it is evaluated with the glass load in order to compare it with other solutions. Therefore, with the glass load applied the generated stress is  $42.7GPa$ , and that is a 30% reduction in comparison to the regular structure. We go further and do the GAs optimization with glass load applied (depicted in the middle), and we see a clear intention by leaving the middle part open, thus avoiding the great displacements in that area. This solution offers a 50% reduction of stress. And if we repeat the experiment, this time with limited cell size, we get a solution on the right side, that has 43% smaller amount of the total Von Mises stress in the grid than the uniform structure. When we compare all three optimizations, we can see that the intention can sometimes be easily recognized and sometimes vague. But all the solutions show better performances when compared to the regular structure and we can use them to create our own structure, following the guide lines they provided. The information obtained from the Voronax optimization can be used to optimize a quadrangular structure, for example, as depicted down in the same Figure 5.33. In the middle there is a uniform quadrangular grid shell. Left of it we see an optimized version, done with the help of stress trajectories, as explained in the previous section. Additionally, there is a small opening in the center, and the members around it are denser, an intention that can be extracted from the first two Voronax optimizations. That brought almost 20% reduction of the total Von Mises stress, and almost 40% reduction of total displacements. As expected, the percentage of the stress generated by axial forces goes from 21% in the regular grid to 34% in this optimized one. On the right there is a grid shell where the regular grid is only slightly changed, according to the first Voronax optimization. The paths remained straight, but the density was increased in the middle to create the  $X$  formation, and the corners were made a little bit denser. This led to 5% stress reduction, and a 20% smaller sum of displacements.



**Figure 5.32:** Deformations of the best and the worst solution

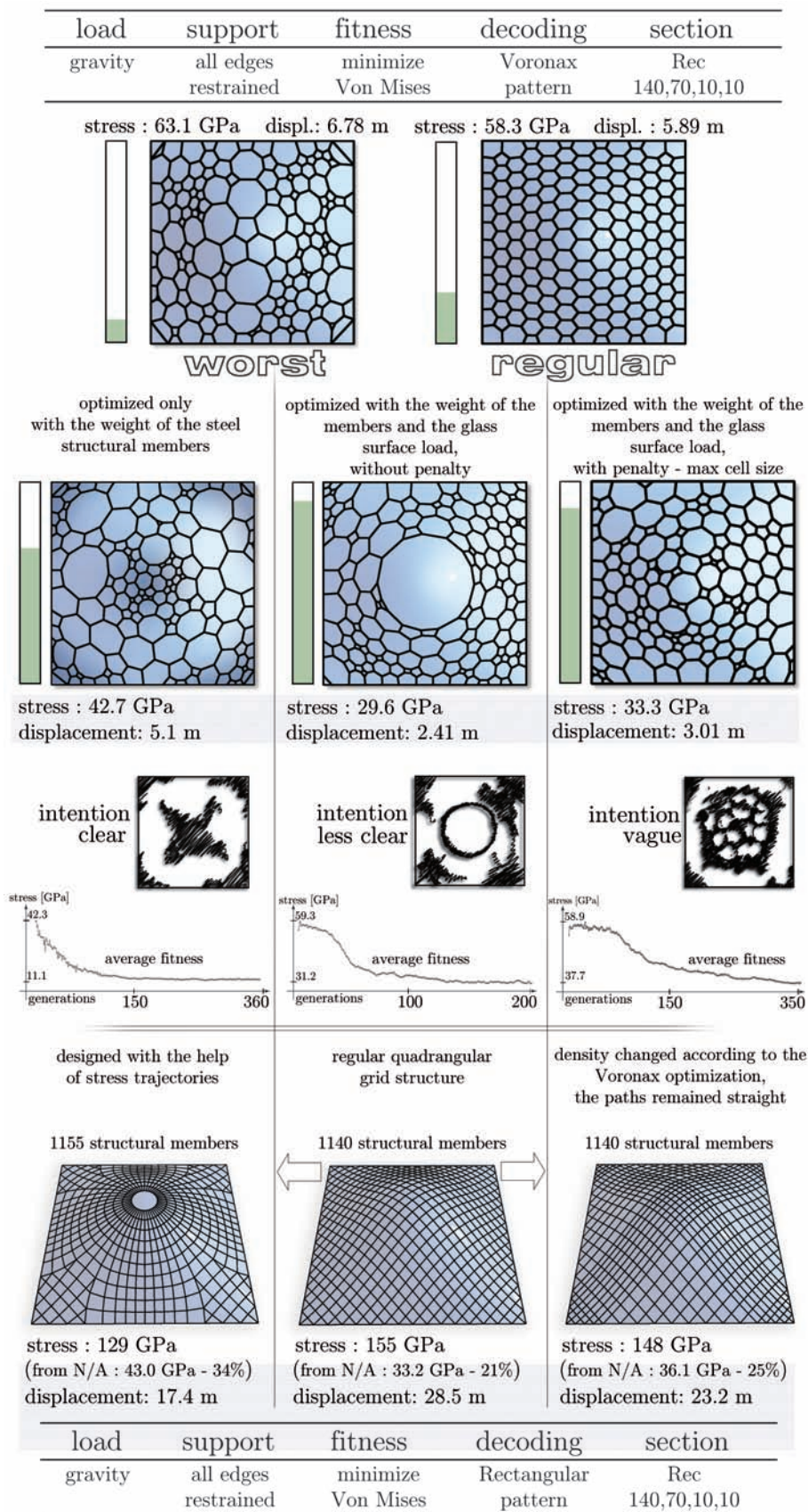
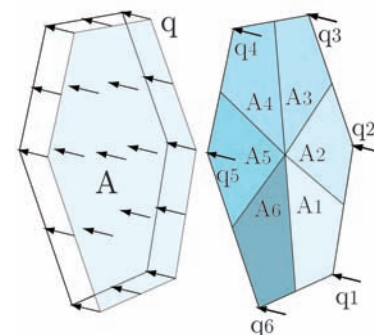


Figure 5.33: Gravitational load results

**Conclusion** It can be concluded, in a surface with corners, that for our predefined surface and gravitational load, the corners and the edges have to be stiffer (regardless of the pattern geometry that we use). (With different surfaces there are different GAs *recommendations* to be followed). Then we have to deal with the center part where the displacements are big. We can do this by stiffening that area, or leaving it open, according to the other conditions that we have. In the first case it is interesting to see how the weight in the middle area is enlarged (with greater density), but the stiffness it brings to the structure compensates for that and results with the smaller amount of stress. This can be nicely seen in Figure 5.32 where the comparison is made between the deformations in the optimal and the worst generated solution. The stiffened center in the better grid makes it deform very little in the middle, thus resulting in smaller total displacement. When there are more solutions possible, like in this case, testing different load combinations can make us decide which of the offered solutions is more acceptable.

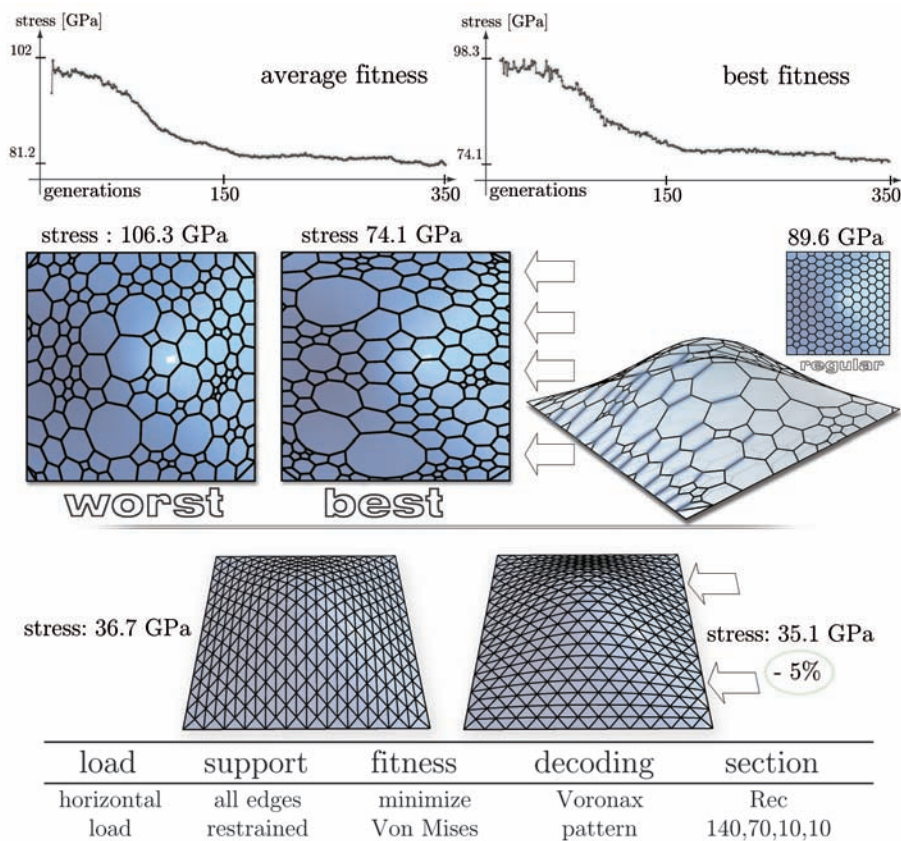
### 5.3.2 Case 2 : Horizontal Load

The second load type which should be examined is horizontal load. In this case we have gravitational load, applied using the weight of the structural members, and the horizontal load applied as the surface load, transferred from the cells to the structural joints like in the case of vertical load (as shown in Figure 5.34). The load was kept at  $1\text{ KN/m}^2$ , and it is applied on the structure from the right (observed from the top). This type of load is *artificially* created only for the purpose of showing the effects it produces, i.e., in order to prove that the GAs optimization converges toward a proper solution. All the generated individuals in the optimization have the joints on the edges fully restrained and steel structural members with the vertically oriented rectangular cross-sections (140,70,10,10).



**Figure 5.34:** Horizontal surface load

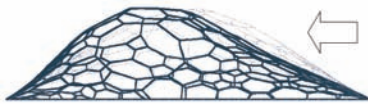
**Results** In Figure 5.35 the graphs show convergence of the average and the best fitness throughout 350 generations. The graphs indicate that the convergence did not achieve its minimum and it is certain that in the next 100-200 generations it would continue to find solutions with smaller amounts of stress. However, the best result obtained from generation 341 shows clearly what the tendency of the optimization process is. Underneath the graphs there



**Figure 5.35:** Horizontal load results



is the worst, randomly generated solution, obtained from generation 1 and the best solution from generation 341. We know that the best use of steel members can be achieved when we minimize the bending moment and maximize axial stress. That is exactly what the Genetic Algorithms have done in this case. The load is coming from the right (depicted with big arrows) and in order to resist it, we can see how a large number of members *answered* by positioning parallel to it. The whole structure *stretched* horizontally and thus prevented large horizontal deformations, i.e., minimized the total stress. In the same Figure 5.35 we see how the member orientation affects a simple triangular structure. By simply rotating the triangles, so that they create distinguishable structural paths parallel to the wind force, we get 5% stress reduction.



**Figure 5.36:** Horizontal load deformation

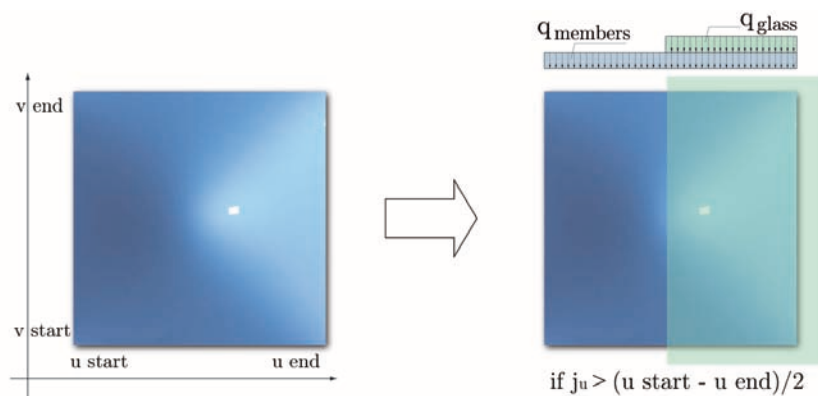
**Conclusion** The structure loaded with single one-sided load has to resist it with parallel member disposition. The members on the edges, where the stresses are the largest (with restrained joints), have to be dense and parallel to the load forces. The structure has to be stiffened in a way that it resist the horizontal deformation in the load force direction, as depicted in Figure 5.36.

Naturally, we don't need Genetic Algorithms to tell us that the optimal disposition of members is the one where the load direction is parallel to their longitudinal axis. But these experiments are made to prove that GAs provide efficient and logical results, so we can *believe* that they offer us the optimal solution, once we cannot determine it intuitively.

### 5.3.3 Case 3 : Partial Load

One more interesting load case is relatively standard in the definition of the worst case scenarios. Namely, the structure has to be able to resist uneven load. In order to define the load case that affects only a part of the surface, we can turn to the surface domain for help. It was already explained how every point on a NURBS surface has a 2-parameter definition. Figure 5.37 depicts the surface, and if we can simply impose the condition that the joint should be on the right half in order to be loaded, we can define our partial load. If we denote a single joint in the structure as  $j(u, v)$  and its  $u$  parameter on the surface as  $j_u$  then the condition for that joint to be loaded can be expressed as  $j_u > (ustart - uend)/2$ , where  $ustart$  and  $uend$  are the end values of the surface domain in  $U$  direction. With that condition we load only the joints that are in the green area of the surface, shown on the right in Figure 5.37.

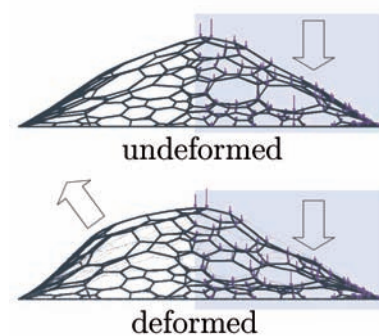
**Results** When the conditions are not so simple, and the geometrical solution of the grid shell doesn't have one strong optimum, we need more generations to achieve a definite convergence. In



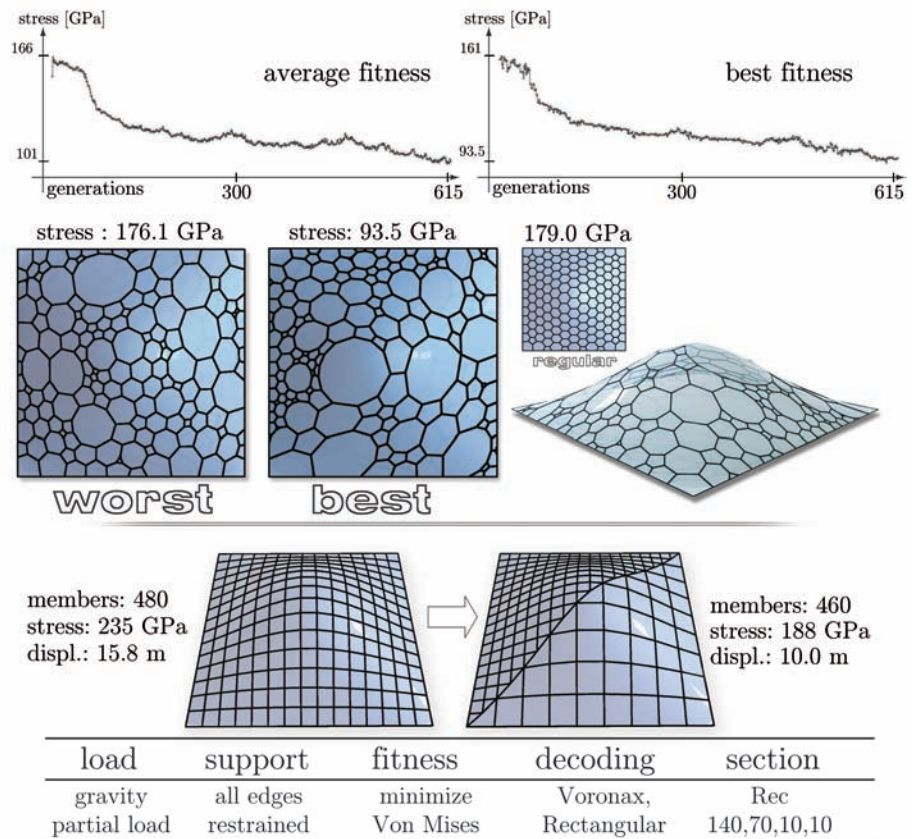
**Figure 5.37:** Definition of partial load

this example 615 generations were generated (30750 individual grid shells) and two graphs in Figure 5.39 show the convergence of the average and the best fitness solutions. From the graphs it is clear that the optimization process hasn't definitely converged and that it would continue to be reduced in search of the minimal stress structure. Nevertheless, we can take one of the best solutions and easily extract the pattern of behavior, i.e., the tendency that the algorithm shows. In Figure 5.38 our best solution is depicted in the undeformed state (with forces acting only upon the right half of the joints) and in the deformed state. Due to the shape of the surface and uneven load, it is obvious that the structure deflects on the loaded side and bulges on the left, the unloaded side. Using that fact, we can find an explanation for why the best solution offered in Figure 5.39 (obtained from generation 596) looks as it does. The Genetic Algorithms obviously tried to stiffen up the left side. In the process they *realized* that the corners are the stiffest part of the structure and shifted small cells to the upper left corner to stiffen up the whole structure, thus generating fitter individuals. The *intention* of this GAs optimization process is open for discussion, but the fact remains that among 30750 individuals generated, the process does converge toward an optimal solution, and that the offered grid shell structure does have minimal total stress in comparison to all of the others. In this case it is also very interesting to see that the uniform structure ( $179GPa$ ) shows bad performance even in comparison to the worst randomly generated structure ( $176GPa$ ). Thorough examination of whole generations can remarkably show the tendency, and the development of the grid shells toward one single optimum.

Just to show that this case is also transferable to a regular structure, in the same figure we see a uniform quadrangular grid on the left, and an optimized grid, according to the Voronax optimization intention, on the right. Naturally, the surface load over one half of the surface is practically never the main load to consider, but this



**Figure 5.38:** Partial load deformation



**Figure 5.39:** Optimizaion with partial load - surface load over the right half of the surface

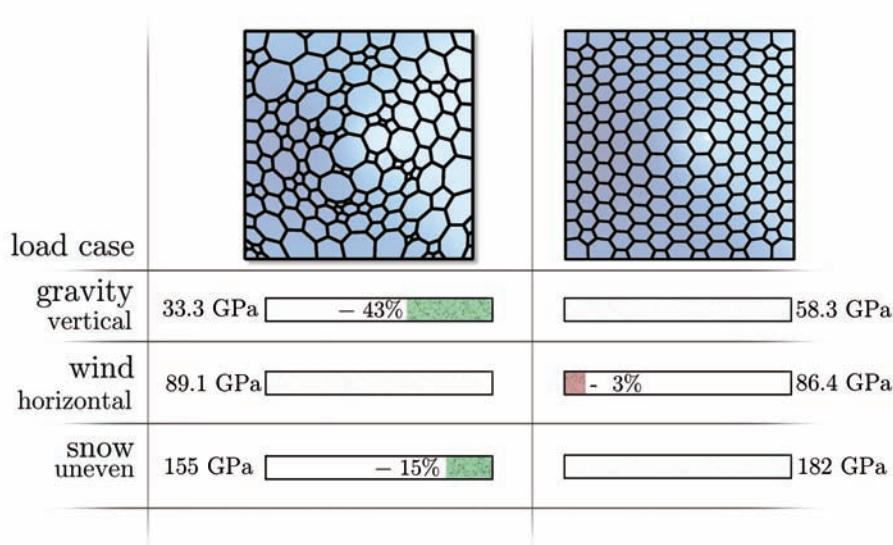
is another example that shows that the intention of the Voronax optimization process can be used to increase the statical efficiency of the regular grid shell.

**Conclusion** If we have a surface and we are supposed to design a grid shell structure over it that is unequally loaded, we have to observe the deformation it produces and stiffen it up in the areas where the stresses are the largest. With simple surfaces, like the one demonstrated in our example, that can be sometimes intuitively determined, but in complicated free form structures we can determine the right cell and member disposition with the help of Genetic Algorithms. As always, this information can be used to generate a structure with a different pattern, if the tendency is recognized and cleverly used. In symmetrical surfaces and loads like this, the algorithm converges toward one solution, but it is clear that the concentration of small cells in the lower left corner would have the same effect. That is why this should be used as a support tool by an engineer, since the machines do not have that ability of comprehensive *thinking* ... yet.



### 5.3.4 Multiple Load Cases Comparison

At the end of this section, it makes sense to analyze how a surface, optimized according to the one load case, behaves in respect to other load cases. One of the arguments in favor of regular structures is that their uniformity is good for that change of loads (snow, wind, etc.). The same surface will be used for this comparison, although its shape has a very small curvature. (It is important to keep this in mind, because the more deformed the free form shape is, the bigger the gains from the optimization are and the less suitable the uniform structure becomes) In Figure 5.40 there is a table with the grid shell optimized for gravitational load (with the restricted cell size), compared with the regular, uniform one. Besides the Von Mises stress reduction, two other load cases are examined here too - wind from the right and snow on the right half of the surface. The optimized structure performs a bit worse in the second case (because of the reduced members parallel to the force direction), but better in the third one. It is very hard to draw general conclusions out of experiments like this. The results will vary from project to project, depending on the shape, pattern, fitness function, etc. The more curved the surface is, the more we will profit from *irregular* structures. Already in the next section, where the experiments with a free form vertical wall are performed, we will see very large differences in stress values. But comparison depicted in Figure 5.40 is just another example of one possible step in the design process that can be performed in search of the optimal design.

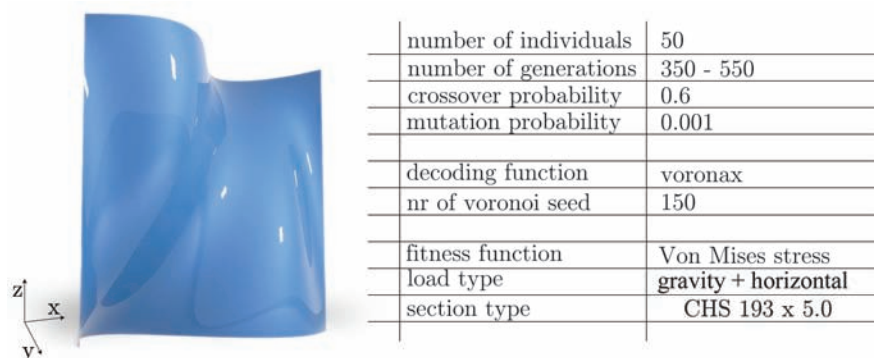


**Figure 5.40:** Comparison of 3 load cases for optimized and regular structure

## 5.4 Support

The principle of different supports was already mentioned in the last chapter, and some of the used support combinations were presented in Figure 4.22. Now we will see how the results of an optimization process with different restraints look like. For this purpose, a free form  $30 \times 30 \text{m}$  vertical wall was designed, depicted in Figure 5.41. Grids are generated with 150 Voronoi seed, i.e., they always have between 455 and 465 structural members that now have circular cross-sections. The total amount of Von Mises stress is used as the fitness function and the standard crossover and mutation probability factors are used - 0.6 and 0.001. Each generation has 50 individuals and up to 600 generations were generated for single optimization processes.

The load used was a combination of gravitational load (self-weight of the structure) and horizontal load of  $1 \text{KN}/\text{m}^2$  transferred to the structural joints (as in the previous section, Figure 5.34). This type of load comes from the intention to have a vertical position of the surface, which should remind of the fact that free from surfaces do not have to only be roof structures and that they are very suitable for facades or *envelopes* for the entire buildings. In order to apply the horizontal load, a simulation of the wind would be appropriate. But the wind acts perpendicular to the surface, and since the cells are not planar, it was difficult to accurately define the angle and the magnitude of the wind force. Therefore, the surface of the cell is calculated and load is transferred to the joints acting in the  $-y$  direction. The fact that the load is not one of the standard ones applied in the static analysis, doesn't affect the efficiency of the optimization process. The optimal solution given is the optimal solution for that load combination, and that is all that's needed to show how the method works and how the results can be analyzed. The purpose of the experiments in this chapter is to see how results differ with different support combinations, and the load type is not important as long it is the same for all examples, which it is.



**Figure 5.41:** Support surface and parameters

### 5.4.1 Case 1 : All Edges

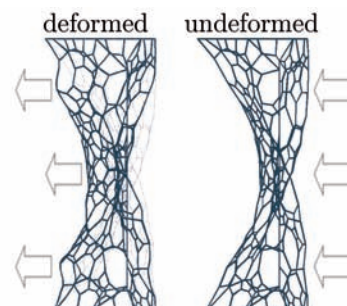
The standard *all edges* support combination (used in most of the other experiments) will be presented first, with its effects on our free form vertical wall. *All edges* simply means that all joints on each of the four surface edges are restrained from movement and rotation in all directions, as it can be seen in Figure 5.42, marked red. Gravity and horizontal load are applied as explained above and structural members have a circular hollow cross-section: CHS 193x5.0 .

**Results** In Figure 5.44, we start with the graphs that show nice convergence after 550 generations, i.e., 27500 generated individuals. The total amount of Von Mises stress in the worst, randomly created solution, is almost three times bigger then the amount in the best one. As always, that depends on the load intensity and the curvature. As the forces grow bigger so does the difference between offered solutions. Respectively, when compared to the results from the last section (where the gains were in the range of 10-50%), here we have much more deformed shape and therefore almost 300% difference between the worst and the best solution. In Figure 5.44 we also see two regular hexagonal solutions with fitness values of  $99GPa$  and  $101GPa$ . The difference between them is in the member orientation, but they have the same number of Voronoi seed and the same number of structural members. We can see how the best solution outperforms them (according to this fitness function) considerably with  $38.1GPa$  of total Von Mises stress amount, i.e., 62% stress reduction. The total displacement is six times smaller in the best generated solution ( $2.22m$ ) than in the worst generated one ( $13.43m$ ).

Since the joints on the edges are constrained, the largest deformations occur in the middle of the surface along the concave diagonal. Three distinctive characteristics can be detected on the best solution offered. First, convex parts (painted green) are *relieved* with the formation of big cells, i.e., lesser density. Second, in order to take the forces over from them and stabilize the whole structure, around both of the convex bulges there is a belt of dense cells (painted red). Third, the concave diagonal is covered with cells that follow its curvature (painted yellow). Since the upper left and lower right part are stabilized, the horizontal forces try to *stretch* the diagonal. The *yellow* cells are then also *stretched* in a way that enables them to take over those tension forces by following the curvature, thus setting the members parallel to the forces. We can check the percentage of the stress generated with the normal forces and the bending moments, to see if the orientation of structural members is better in the optimized solution. Indeed, in the worst generated solution axial stress participates in the total Von



**Figure 5.42:** Case 1: All edges restrained with horizontal load



**Figure 5.43:** Case 1: Deformation from horizontal load

Mises stress with 3%, whereas in the optimized structure, the stress generated from axial forces rises to 10%.

If we put one generation (50 individuals) in a row and look through the solutions, we gain a nice clue about the direction in which we should think. In the middle of Figure 5.44 it is shown what happens when we *look through* generations 1 and 523 of this optimization process. We cannot conclude much from the first, randomly generated generation, but we do have an idea of the optimal structure which could be reached when we look through the generation 523.

We can use this information and try to design a structure with the quadrangular grid, as depicted down in the same Figure 5.44. On the left is a standard uniform quadrangular structure with the 51GPa total amount of generated Von Mises stress (with the same load and support settings as in the experiment above). On the right we see what a structure can look like with the similar number of structural members, arranged according to the intention recognized in the Voronax optimization process. Underneath the grid shell visualizations we see reductions in stress and the displacements which take place. The optimized structure has a 13% smaller total Von Mises stress, 25% smaller displacements, and a slightly better load buckling factor (which makes it 18% more buckling resistant).

Now, it is hard to say that the grid on the right is *pretty*, and it can hardly be accepted as a final solution. But the *prettiness* was not the goal in this experiment. The optical beauty of the grid shell can be achieved with relaxation tools and careful member rearrangement. This experiment shows that the density and the path information obtained from a Voronax optimization can be used to find a statically efficient arrangement of structural members with different patterns. Considering the fact that we can combine triangles, quadrangles, hexagons, or any kind of n-gons in one grid shell, there are an infinite number of solutions that we can generate over some predefined NURBS surface.

**Conclusion** When designing a structure over a given surface with a clearly emphasized convex and concave parts, we have to observe the curvature carefully, in order to obtain an optimal design. We cannot know immediately what the compensations will be if we strengthen different parts. That is why we can use these optimization tools until we develop an intuition for that. Examining the best offered solution, or *looking through* the latest generation, we can extract the density and path information applicable to grid shells with various patterns.

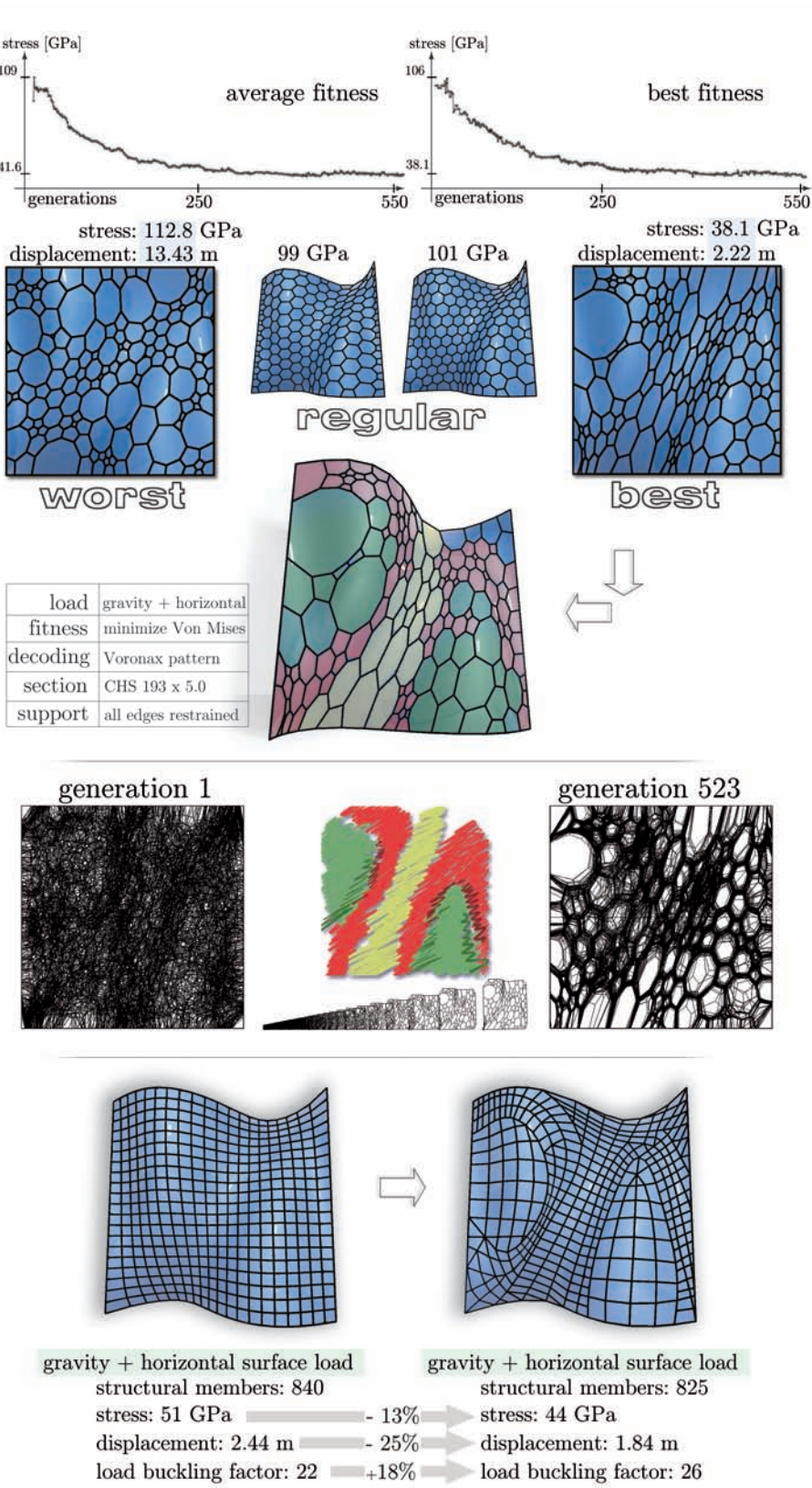
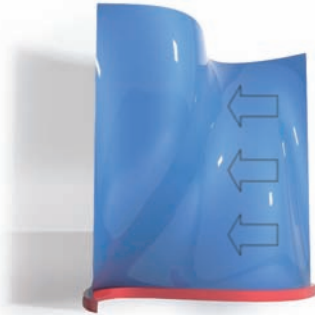


Figure 5.44: Results with the structure fixed on all four edges



### 5.4.2 Case 2 : One Edge

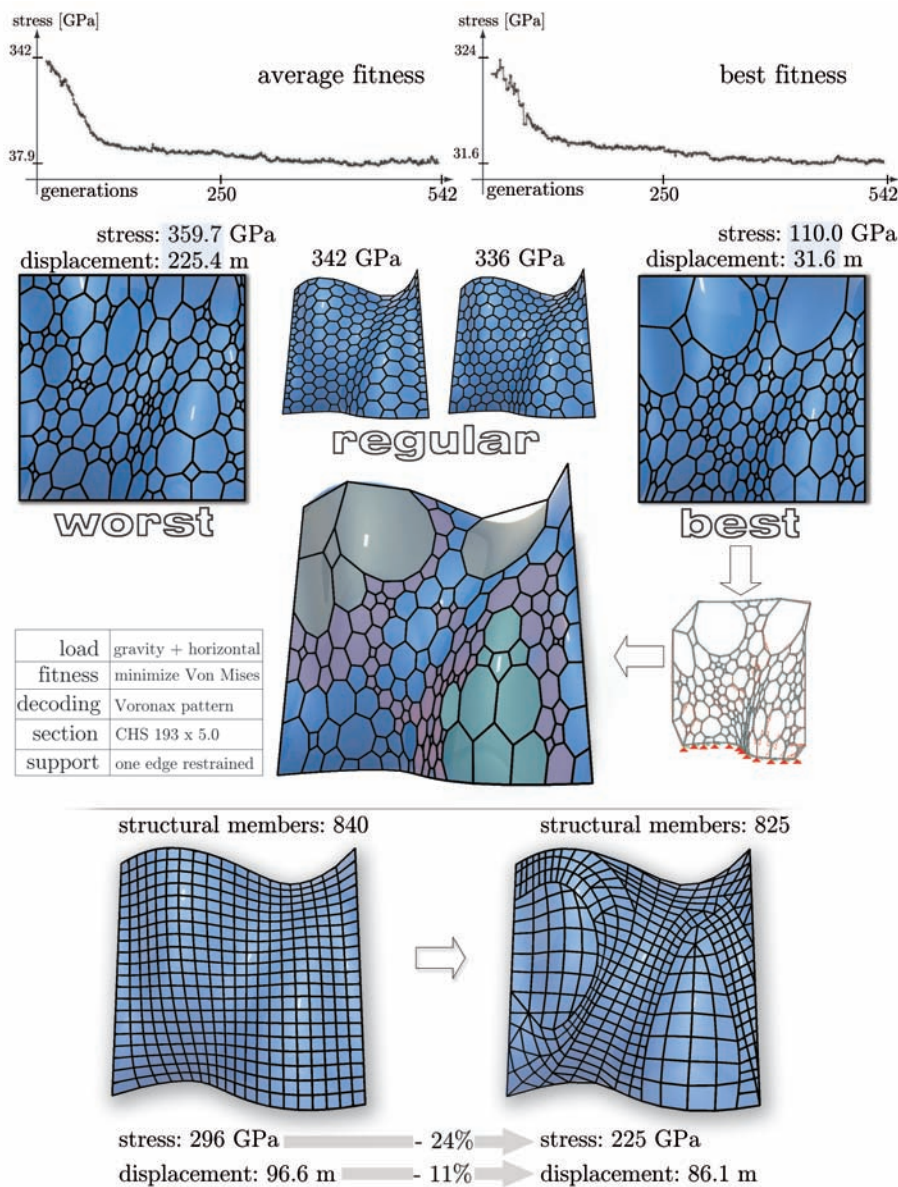


**Figure 5.45:** Case 2: Bottom edge restrained with horizontal load

In this second example, the same free form wall will be observed, but this time supported only at the lower edge. In Figure 5.45 is a visualization of that support setting. Since the stresses created are larger in this case, the surface was scaled down to  $21 \times 21m$  and the same circular hollow section (CHS 193x5.0) is used. The scaling was done only to get more realistic stresses in structural members, but the size of the nominal values doesn't affect the optimization process. Even if the optimization is done with all material limits highly exceeded, it shows the same convergence and offers the same optimal solutions. All of the other settings (load case, Voronoi seed, etc.) are the same as in the previous example.

**Results** In Figure 5.46 we can see that, as expected, the density shifts a little bit downward, i.e., toward the support. This is logical, since the stresses in this cantilever disposition are naturally the largest in the area where the joints are restrained. Still, it keeps the concave diagonal and stiffness around the convex parts, with a larger member density, in order to stabilize the entire structure. That is why the convex parts in the lower right and upper left corner have bigger cells again. They simply rely on the denser parts. There are no penalty functions here and that is why there are a couple of extremely big cells at the top (painted yellow), but with few restrictions that can easily be avoided. The worst solution surprisingly does have a similar disposition as the best one, but due to the big density at the top and the right, the stress generated is much bigger. This shows clearly that very small differences in geometry can produce big differences in stress amount, and in this case our best solution has more than 3 times smaller total amount of Von Mises stress that the worst generated one, i.e.,  $110GPa$  opposed to  $359GPa$ . Regular hexagonal structures show as bad performance as the worst generated solution, with  $342GPa$  and  $336GPa$ .

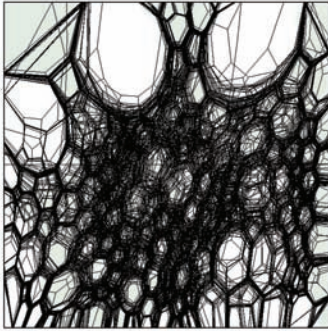
Figure 5.46 also shows a uniform quadrangular structure on the left. Since the disposition of the grid density is similar to the previous experiment (with all four edges restrained) the same optimized quadrangular grid is used for comparison in this case as well. In this case it would probably be profitable to make the polygons on the top even larger. Those kinds of *tunings* can be made according to the specific requirements of a project. Here, it is only important to see that in this case we can use the Voronax information to optimize a grid with different pattern combination as well. The amount of generated stress in the solution on the right is 24% smaller and the total amount of displacement is reduced by 11%.



**Figure 5.46:** Results with the structure fixed only at the bottom

**Conclusion** If we have a cantilever situation, i.e., the structure is only supported at the one end, there has to be an extremely clever compensation between density of the members on both ends of the structure. Basically we always need more density in the vicinity of the restricted joints, but we do not want the deformations on the other end to be too large. That is why we can use GAs to get a balanced distribution and use penalty functions to avoid oversized cells and members. Additionally, as in the previous example, we always have to consider the curvature of the free form surface to get an optimal design. A look through the generation 530 is shown in Figure 5.47, where it can be seen how the GAs increased density of the cells in the middle to stabilize the entire structure. It is a result that comes from a distorted shape and something that we





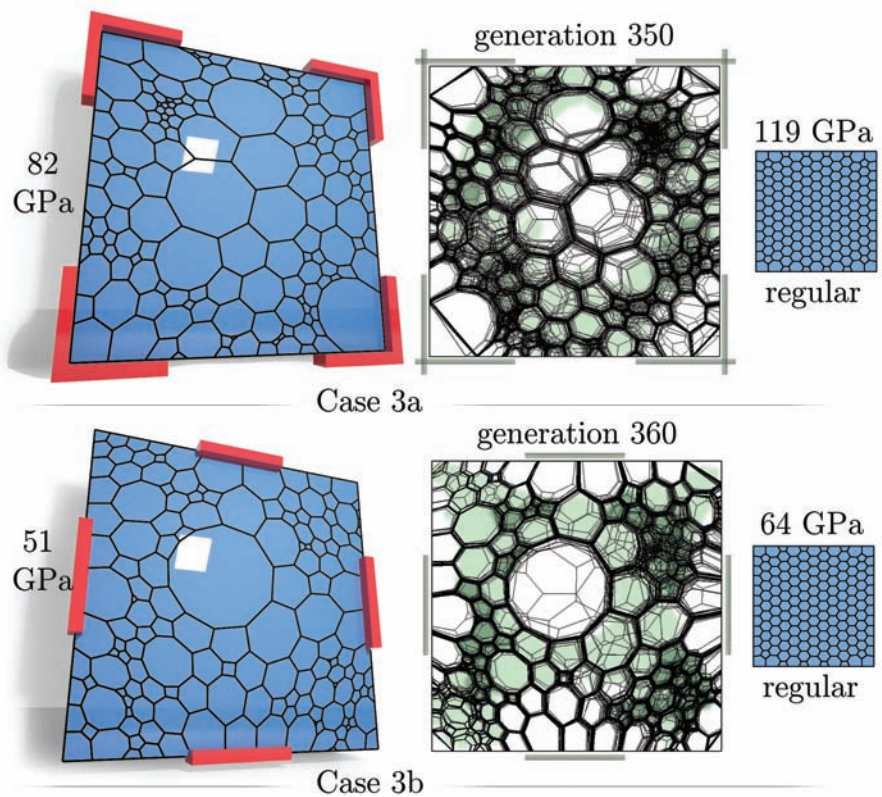
**Figure 5.47:** Case 2: Generation 530

could hardly intuitively take as an optimal design, although it is (in this case) the minimal stress solution.

### 5.4.3 Case 3 : Partial Support

Different combinations of supports were tried out on the vertical wall, but they all showed the similar tendency, strengthening the structure around the convex parts and adapting the diagonal members to the curvature. That comes from the fact that the strong curvature of the free form surface is in this case much superior as a factor in comparison to the support combination. Therefore, tests with different supports will have similar optimal geometrical solutions. That is why another surface will be used to show what happens if the support is partitioned. In fact, to make the effect clear, we'll use a  $21 \times 21m$  rectangle flat surface. The cross-section of the structural members used in these experiments is also CHS 193 x 5.0.

**Results** There are two cases, 3a and 3b, one with nodes restrained in the corners and the other with restrained nodes in the middle of each edge, as shown in Figure 5.48. Next to the solutions,



**Figure 5.48:** Results from cases 3a and 3b

the 50 individuals of one generation are lined up, to show the convergence toward optimal solution. In the case 3a, the look through

generation 350 is shown, and in case 3b we can see the convergence of all individuals in generation 360. An interesting pattern of behavior can be seen in these two offered solutions, where the areas that cannot rely on the support are denser and therefore stiffened up.

**Conclusion** Intuition can sometimes mislead us in thinking that we can stabilize the structure by increasing the density near the restraints. Especially since that is true in some cases, which can be seen in some of the previous sections. However, the grid shell solutions with least stress, in this case, are clearly the ones where there is a reasonable amount of members in the restrained area and a larger density in the parts where the largest deformations would occur. Interestingly, the center part in both cases has bigger cells, stabilized with the *O*-shaped formation of denser cells in case 3a and *X*-shape formation in case 3b. Again, the oversized cells can be easily controlled with penalty functions, but our goal is to read the intention and design our own structure afterward, according to the guide lines that GAs provided. Another interesting fact is that if we remember the worst solution from the first example in the previous section, depicted in Figure 5.33, it looks similar to our best solution in the 3a case here, i.e., it has the same *O*-shaped formation of denser cells. That shows how sensitive the structure can be according to the shape and factors like load or restraint. By changing restraints from 4 edges to 4 corners we get completely different optimal geometrical grid disposition and that is something that can hardly be manually and intuitively determined, especially when designing a structure over heavily curved free form surfaces.

#### 5.4.4 Case 4 : All Edges - Movable

In order to demonstrate what happens when the joints on the edges are free to rotate or move, and the grid shell has to take over additional stress, we can see the results of an optimization process using the stretched, bridge-like surface. A simple shape is used again, and the advantage is that the results can be clearly demonstrated. The only negative aspect is that, due to the small curvature, the gains will not be substantial (as in the Section 5.3). But that does not get in the way of demonstrating the efficiency of the method. The load applied here is gravitational, i.e., self-weight of the structure (steel members +  $1\text{KN}/\text{m}^2$  glass load). Cross-sections of the structural members are rectangular (140,70,10,10) and vertically oriented.

**Results** In Figure 5.49 there are two columns. On the left are the results of the experiments made with the joints on the two ends of our bridge surface restrained from all movements. The results of the optimizations done with the joints restrained only in

$y$  and  $z$  direction, i.e., the joints are free to move in longitudinal( $x$ ), direction and to rotate in all 3 directions, are to be found in the right hand column.

On the left, we start with the worst generated solution and the optimal solution from generation 608. In the optimal one, the structure is stiffened up near the supported edges as well as in the middle. The deformation pictures can be very helpful in this case. From the deformation of the solution with fixed edges (in the top view) we can see how the structure *escapes* sideways. The zone in the middle is denser, to prevent that movement. Beneath the deformations, we can see the graph, showing a nice convergence of average fitness after 612 generations. In this experiment, it is interesting to see that the regular hexagonal structure shows worse performance ( $108GPa$ ) than the worst generated solution ( $77.3GPa$ ). Below the depicted hexagonal structure, there is a *design attempt*. Namely, following the *intention*, extracted from the best offered solution (depicted in green in the figure), there is a grid structure *manually* designed. It shows that the fitness value is close to the best one offered. That means that the intention is read properly and that we should move in that direction to find an acceptable solution.

On the right is an experiment done with the joints that were free to rotate in all directions and move in the  $x$  direction. All of the other settings are the same as in the experiment on the left. The best solution looks different here. The deformation shows that there is no more buckling in the middle, and that the important thing is to enlarge the density near the supported edges, in order to take over the large horizontal displacements. The graph shows a nice convergence of the average fitness and beneath the graph we can see the intention of the GAs algorithm. To test if we read the intention right, we can suggest a design, like the one depicted on the bottom of the right hand column. Interestingly, the proposed solution is even better ( $380GPa$ ) than the best generated one ( $407GPa$ ). That means that the intention was read properly and that GAs helped us to design a much more acceptable solution (according to the Minimize Von Mises stresses fitness function) than the regular one. Again, the regular structure shows a very bad performance ( $575$ ), i.e., again worse than the worst generated solution ( $545GPa$ ).

**Conclusion** The gains in these experiments are not that big, due to the small curvature of the surface. Nevertheless, it can be seen how small changes in every aspect of the construction (joint restraint in this case) can result in different optimal grid structures. In the engineering practice, one would have to test different cases *manually*, and would therefore be limited to very few experiments. GAs on the other hand can test thousands of solutions for each of the different settings.

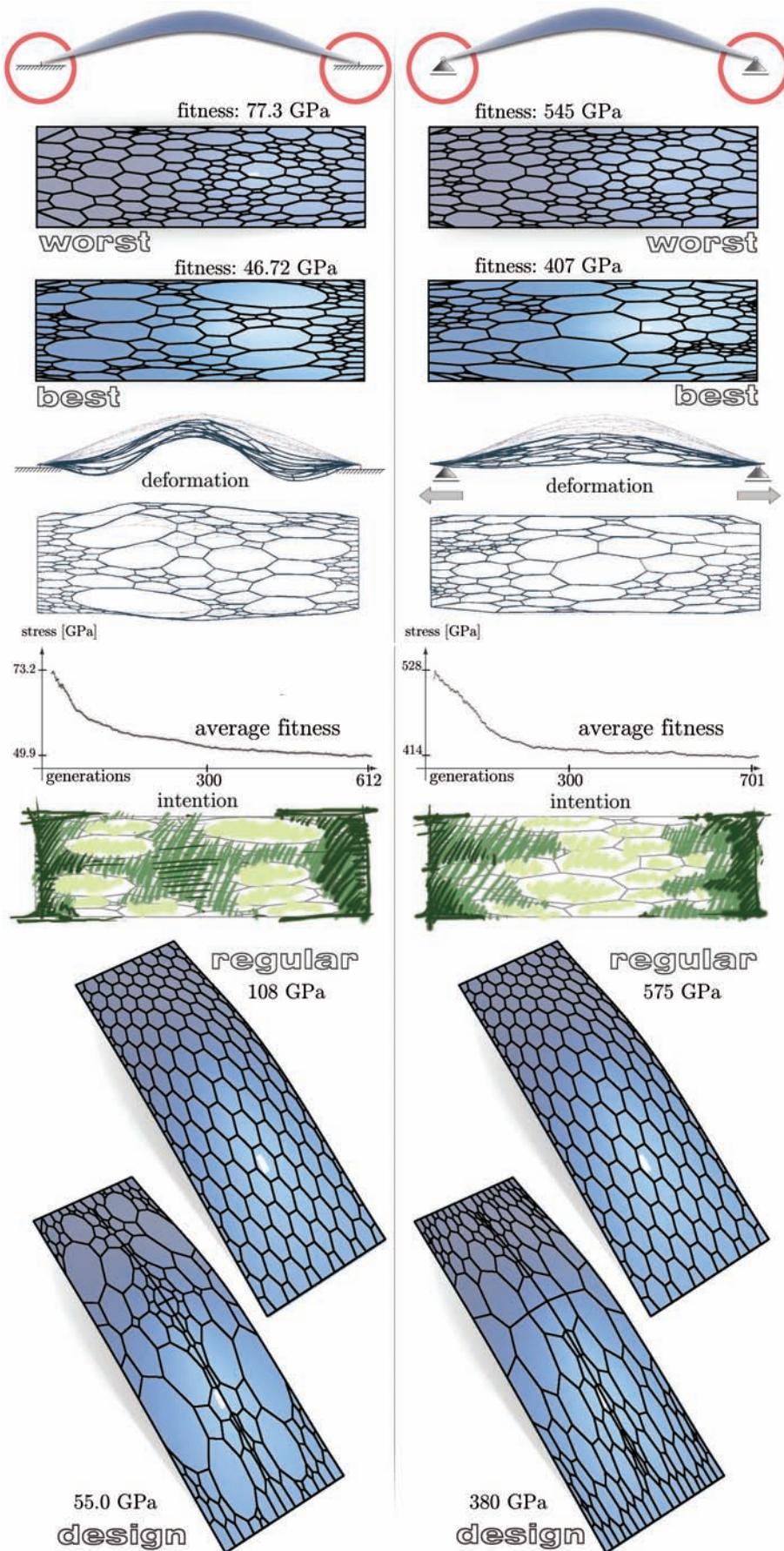


Figure 5.49: Analysis of different joint restraints

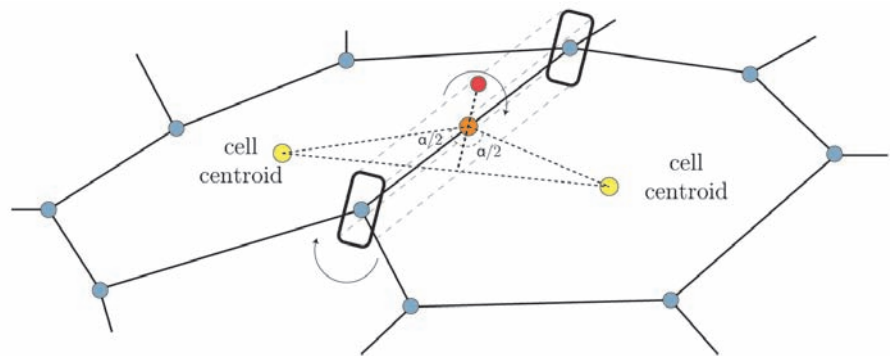


## 5.5 Different Sections

The possible variation of the structural member cross-section type is going to be addressed in this section. Since the main purpose of the presented research is to demonstrate and explain the method of free form grid shell structural optimization, the choice of the cross-section doesn't have a large influence. This means that the effectiveness of the proposed optimization system doesn't depend on the shape of the cross-section of the individual grids, i.e., it works for any section type and size selected at the beginning of the process.

One additional aspect has to be considered, and that is the orientation of structural members. When they are not tubes (circular cross-section), then the stress generated depends on their orientation, i.e., rotation around their longest axis. In most of the experiments presented so far, grid shells are generated with all members with vertically oriented rectangular cross-section (member's local  $z$  axis is parallel to the global  $z$  axis), although the orienting (rotating) method of members according to their position was implemented as an option in the algorithm. Two reasons why that was not used will be shortly explained, but first the algorithm of the method will be described.

The aforementioned method, of dividing the angle between the neighbouring cells in half, is used to implement the orientation in the software. The process is depicted in Figure 5.50. Since the



**Figure 5.50:** Orientation of structural members according to the non-planar neighbouring cells

cells are not planar, the approximation of the angle of member rotation is used with the help of the cell centroid. In Appendix C the method of cell recognition is described, and the calculation of centroid for each cell. The centroids are then connected with each other and with the middle of the member longitudinal axis to create a triangle. Then the triangle angle at that middle point is divided in half, thus providing an orientation point - red point in the figure. The coordinates of that point in space are used as

input data for the FEM software which then rotates and orients the structural member according to it.

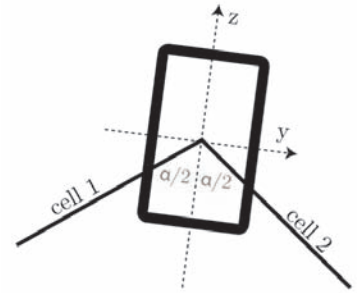
First reason for not implementing the orientation is the *correctness*, or the *precision*, of that attempt. Namely, every structural member divides two faces (cells), two triangles, quadrangles or n-gons in a Voronax structure. If those two faces are planar, then there is only one correct orientation of the member, i.e., in a way that its local  $z$  axis (Figure 5.51) divides the angle between cells in half. If the cells are not planar, then it is debatable what the right rotation should be. This is something that has to be solved in the future, if the development of grid shells with non-planar faces continues. When a number of acceptable solutions is acquired, it will be easy to import them into the existing algorithms and expand the proposed optimization method to generate individual grid solutions with properly oriented structural members.

The second reason is one which has already been emphasized many times. It is the fact that the proposed method works regardless of the orientation of the structural members. Whatever the initials settings are, GAs will converge toward an optimal solution defined by those settings. It is not the intention to try every possible combination of those settings, but to prove that the method works for any combination. Then, in the future, every part of the algorithm can be expanded and changed without the effect on the efficiency of the optimization.

Therefore, since the proper orientation of members costs a lot of computing time to do something that is debatable, and since it doesn't affect the proof of the efficiency of the proposed method (which is the purpose of this research), it is avoided.

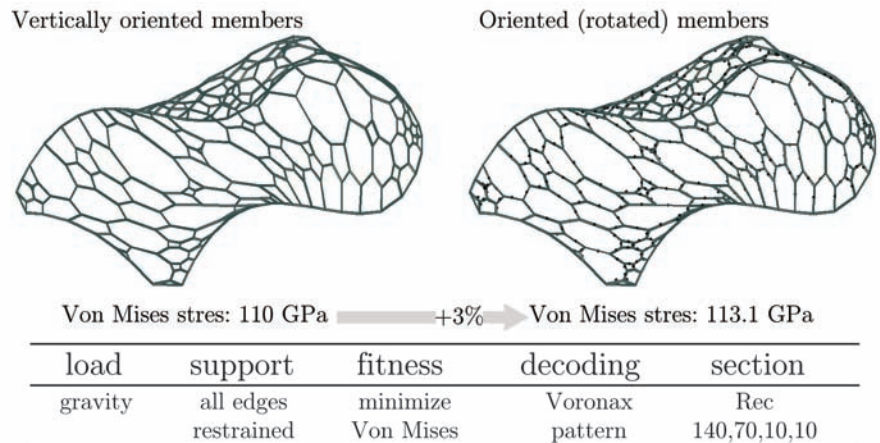
Nevertheless, several grid shells that were generated by the Genetic Algorithms were tested with vertically oriented members and with rotated members according to the method just described. The total amount of Von Mises stress is then calculated for both options and compared. With most of the grid shells the difference in the Von Mises stress generated was in the range of 0%-5%. This doesn't mean that we can say that all results from the optimizations done with oriented (rotated) members hold for vertically oriented and vice versa. This just implies that the orientation of structural members in the experiments performed shows a much smaller significance, i.e., a smaller effect on the generated stress, than the grid density. Therefore, for most of the optimizations performed we can say that the results hold for grids with both *oriented* and *non-oriented* members. As an example, in Figure 5.52 is an optimal structure from the Section 5.1, the stress of which is calculated with vertically oriented and rotated members. The difference between solutions is around 3%, which is the average difference.

It was already explained why we are concentrating only on geometrical and topological optimization here (and not on the material



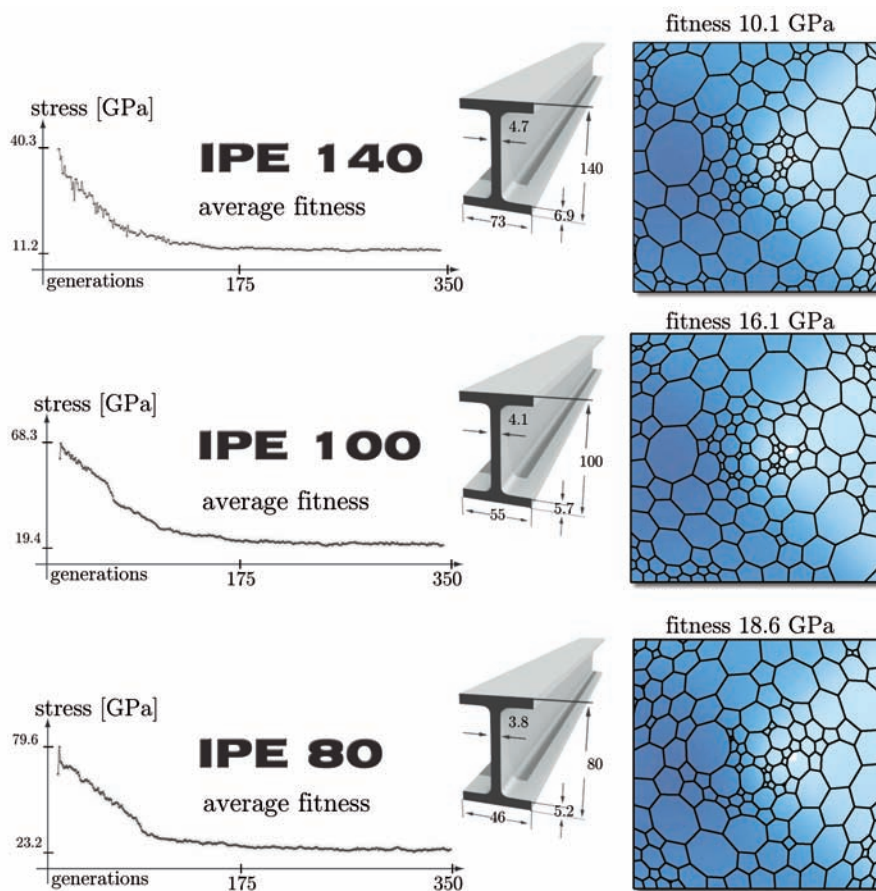
**Figure 5.51:** Proper orientation of the structural member according to the planar neighbouring cells





**Figure 5.52:** Comparison of the same grid shell with vertically oriented (left) and rotated members (right). The small black squares in the middle of the structural members on the right are software’s (GSA) markings of the cross-sectional rotation

and cross-sectional optimization). In the performed experiments, the cross-sections used were chosen according to the span and the number of structural members, so that the average stress in one structural member is reasonable. In most of the experiments an attempt was made to keep the average Von Mises stress in one member below  $100MPa$ . But, for example, we could try to do several optimizations with different cross-sections too see if the results would change, and if the nominal value of the generated stress changes something. This was done for several surfaces, and the results showed that there are basically no differences. When trying to minimize the stress in the structure, the end result was always the convergence toward the same geometrical pattern. In Figure 5.53 are the results of three optimizations, done with three different IPE steel profiles (IPE 140, IPE 100 and IPE 80). In all three optimization processes we have a nice convergence after 350 generations and on the right side we see the optimal offered solution for each attempt. Naturally, as the cross-sections grow smaller, the generated Von Mises stress becomes larger, but the geometrical disposition remains the same. Regardless of the cross-section size, the minimal stress solution in all three cases has greater cell density in the middle and in the corners.



**Figure 5.53:** Three GAs optimization processes with different steel cross-sections

## 5.6 Summary

In the last four sections, the application of GAs optimization process was demonstrated, using some simple examples. It was attempted to prove that the process converges to an optimal solution, according to the parameters and conditions that we set. Definition of those parameters comes with the experience, and engineers learn from practice what type of the worst case scenarios should they analyze in order to achieve an acceptable structural solution. But once the *questions* are formed, instead of trying out a few different combinations manually, we can let GAs test tens of thousands of solutions for us and give us an optimal *answer* to that question.

The examined variables (fitness, pattern, load, support) were chosen since they nicely demonstrate the variation of optimal solutions according to different input settings. If the specific project requires different variables to be investigated, the presented plugin application can always consider other parameters or easily define new ones. Genetic Algorithms work with genome information and fitness evaluation, and it was shown how every grid structure

can easily be transformed into a chromosome, and how there are no limitations in developing our own system of evaluation (fitness function).

Everything presented so far has one clear goal. That goal is to demonstrate that if we have some predefined free form NURBS surface, and we are supposed to design a grid shell over it, there is always a lot of space for the statical optimization of that grid structure. A very important fact, that there is practically no more cost difference in producing identical or unique structural members, made it possible to think about the best geometrical disposition of the grid. In the examples presented, an attempt was made to show that the gains from the geometrical alterations can be big and that there is always a substantially large design space that contains the solutions that are statically favorable when compared with the standard uniform grids.

The focus of the described research is not on the results but on the method. Experiments and their results are used to prove the efficiency of the proposed method. The reason for this is the generality of the described optimization system. Namely, the number of possible combinations of input parameters (shape, grid pattern, material, cross-section, load combination, support combination, fitness function, penalty function, GA parameters, etc.) is too big in order to try them all out. Therefore it was reasonable to concentrate on the algorithm itself, and to make a number of experiments sufficient to prove that the optimization converges toward the optimal solution according to input settings, i.e., offers a grid shell solution statically more efficient than the standard uniformly distributed grid shells used nowadays.

Since the GAs optimization process is inspired by Nature and its selection method, it makes sense to compare the end results, our *artificial* structures, with the structures that Nature produces. In the next chapter we will see how the described methods move us one step closer to the Nature's highly optimized structural systems.

# 6

## Nature

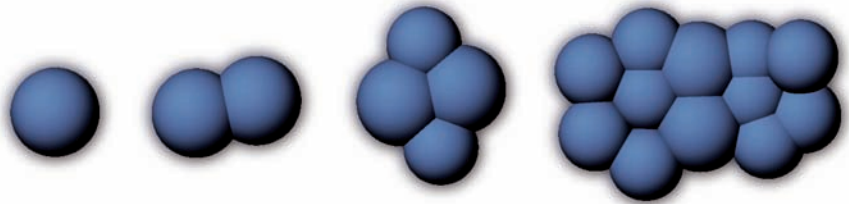
In previous chapters it was shown how Nature's basic rule of selection can be combined with computers in order to get an answer to different questions that start with: "*What is the optimal ...*". It will be useful to turn shortly to the forms generated in Nature in order to draw some parallels between natural and artificial structures. Direct comparison would not be an easy task, since the fitness functions in structural design of buildings and living creatures can be quite different. The most obvious difference is that in Nature, structures are usually dynamic, optimized for different motions, and in architecture we deal mostly with non-movable systems. In deployable structures, GAs can be used as effectively, but the fitness functions would become more complex, and that is something that will surely be a part of future research. That will be another step forward to getting closer to the Nature's *engineering skills*.

### **Natura non facit saltus [10]**

A very important characteristic of our natural environment is suitably summed up in: *Nature does not make a leap*. Nature doesn't *know* anything, it tries. Each and every one of its creations is influenced by a large number of factors from its environment, and therefore basically randomly (accidentally) altered. Natural selection preserves the better design and relies on inheritance. Now the big question is, whether we have surpassed Nature and do not use any random alteration, or whether our comprehensive knowledge and intelligence is just an illusion and we are basically using the same principle of inheritance? We rely on what we know and then try something new with small alterations, thus probably mistaking all the accumulated skills from the beginning of mankind for an illusion of conscious and comprehensive thinking.

The explanation for Voronoi-like structures came out of this continuous process of alterations. The basic principle of cell division at the microlevel, depicted in Figure 6.1, shows how every cell in

Nature has its own centroid according to which it tries to define its form. The geometry is then influenced by its size and strength (material properties) and by the neighbouring particles and their centroid and volume. It can be seen how the Voronoi diagram basically represents a graphical explanation of structure formation in Nature.



**Figure 6.1:** Simplified graphical representation of Mitosis, cell division process, depicting a natural formation of Voronoi diagram



**Figure 6.2:** Beehive - hexagonal structure

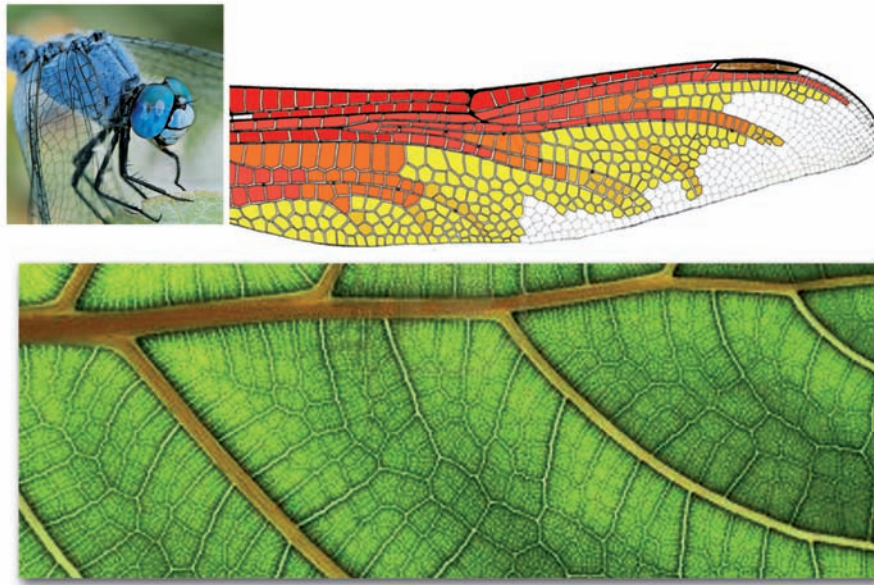
Figure 6.2 shows a beehive, the perfect example of Nature's optimization process converging to an optimal structure. It is a structure that can hold the greatest possible amount of honey, with the least possible consumption of precious wax. We can recognize the hexagonal pattern, made by bees that stand of the same relative distance from each other and try to sweep equal spheres. If each bee represents a Voronoi seed, this can be seen as a regular Voronoi structure with hexagonal disposition. For irregular Voronoi structures we can refer to the leaf structures or insect wings (Figure 6.3), as beautiful examples of structurally optimized designs, with definition of primary, secondary and tertiary elements.

If we look at the foam structures (Figure 6.4), we can basically explain the inception of the idea that lies behind the Voronax structure. Foam is the attempt of the thin films to trap the gas in the best possible way, i.e. minimizing work and material. Since sphere is a shape that can contain the largest possible volume with least surface, they are formed next to each other, *fighting* for space (the spheres in 3D are analog to the circles in 2D). This *fighting* is actually a relaxation process where each cell tries to minimize its potential energy. Searching for the least material (and maximal volume) solution, they iteratively rearrange to form a system of convex cells, following the similar principles that we used to create Voronax structures from Voronoi diagrams.

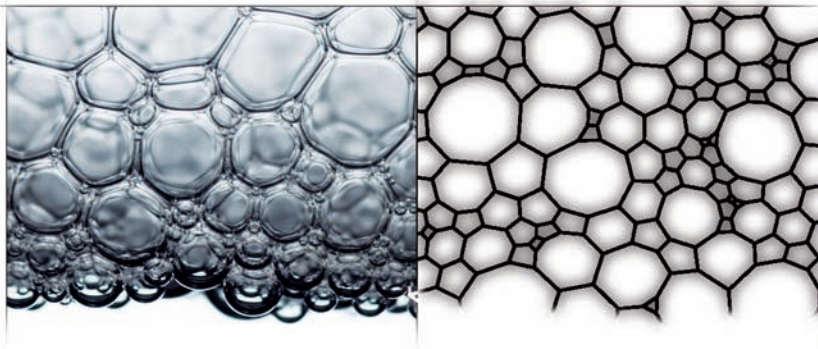
### **Natura nihil frustra facit [10]**

This sentence, written in Latin, means *Nature does nothing in vain*. The concept was discussed in the first chapters when we talked about Nature's minimum material and minimum potential energy, that we also use in structural optimization. But even here, the most important of all questions has to be asked: Why?





**Figure 6.3:** Different examples of the Voronoi pattern in Nature: Dragon fly wing (up) and leaf structure (down)



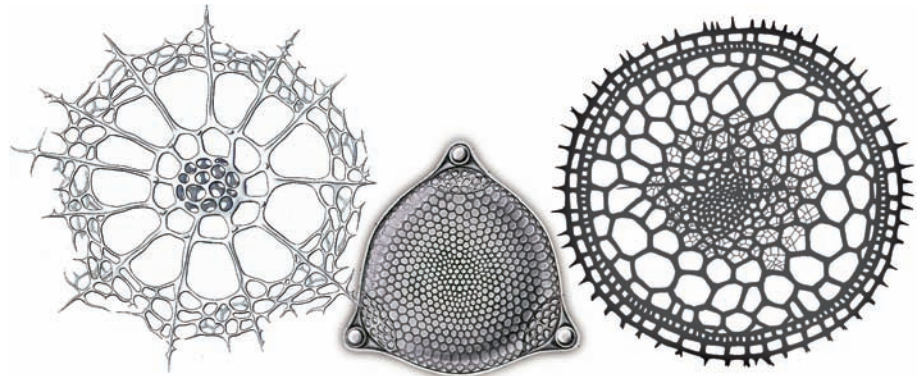
**Figure 6.4:** Foam(left) and Voronax (right)

Charles Darwin can help us in resolving that when he says, “...natural selection is continually trying to economize in every part of the organization. If under changed conditions of life a structure before useful becomes less useful, any diminution, however slight, in its development, will be seized on by natural selection, for it will profit the individual not to have its nutriment wasted in building up a useless structure.” [10]. The optimized design survives, the waste of material and energy leads to extinction. There were lots of bad Nature’s designs in Earth’s history that simply do not exist any more. Darwin continues, “...we should bear in mind that animals displaying early transitional grades of the structure will seldom continue to exist to the present day, for they will have been supplanted by the very process of perfection through natural selection.” [10]. This beautiful simplicity justifies our goal, and explains our urge



to optimize, i.e., to design.

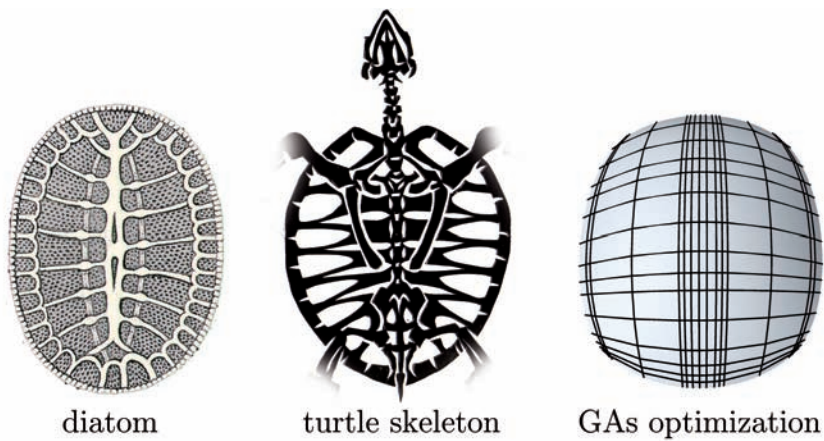
Radiolarians are microorganisms with mineral skeletons that can help us draw some parallel between Nature and our optimized grid shell structures. We can try to compare some diatoms and radiolarians (Figure 6.5), perfected throughout years, with some of our artificially optimized structures. The diatoms and radio-



**Figure 6.5:** Two diatoms (left, middle) and a radiolarian (right) with the larger cell density in the center and on the edges, like in several of our experiments

larian depicted are illustrations of Ernst Haeckel, made in the 19<sup>th</sup> century [18]. These comparisons are interesting especially because they show how every structure in Nature has some grid pattern, on a macro or micro level, and that they are all actually a form of Voronoi or Voronax structures. We can see in this example how the depicted radiolarians followed the same stiffening of the center part and the edge-cells that we had in several of our examples. That is however something that cannot be taken for granted, since radiolarians have to satisfy a much larger number of conditions and objectives than just minimization of stress.

It is clear that every animal or human skeleton is an example of Nature's structural optimization. If we remember one of the previous examples, done with the rectangular pattern, as well as Voronax structures from Section 5.2, it is clearly distinguishable that the GAs try to define primary (*spine*) and secondary elements. An easy parallel can be then drawn to the animal skeletons but also to the leaf structure or insect wings. It is apparent that the uniform allocation of structural members does not offer optimal solutions and that grouping them together shows far better results. This is another very important clue that has been used by engineers throughout history which can be confirmed with our GAs optimization processes. In Figure 6.6 we can see an interesting comparison of one diatom, a turtle skeleton and an optimal solution offered by our GAs algorithm with the rectangular pattern.



**Figure 6.6:** Diatom, turtle shell and GAs artificial solution



# 7

## Conclusions

In this final conclusion, an attempt will be made to sum up everything presented so far. The advantages of the proposed optimization method will once again be emphasized. Advantages and flaws of the Voronax structures will be brought to attention. Finally, some of the plans for future research will be discussed, along with the expansion possibilities of the developed methods. The list of innovations and methods developed in this research will first of all be recapitulated.

### Innovations

Design of the free form grid shell, from a geometrical point of view, is not an easy task and usually involves many steps of *manual* work. In this research it was shown how an  $xy - uv$  transformation can be used to define an algorithm that generates a grid structure over any NURBS surface. It was demonstrated how, with the use of Voronoi diagram, we can input a few parameters and automatically generate a triangular, quadrangular, hexagonal, or natural Voronoi structure.

After showing how the Force Density Method can be used in the grid shell design, in this research this method was expanded to work for any kind of grid structure (any kind of graph). It was explained how to construct an algorithm that will relax a grid structure (2D or 3D) in space, or by keeping it on some predefined free form surface. This method was used to develop a new type of grid structure. Voronoi diagrams, generated over a NURBS surface, were relaxed, restrained to the surface, and the resulting structure was named Voronax. It was explained what structural advantages this structure has, compared to the Voronoi, and how they can be used for the grid shell optimization.

The results obtained from the optimization processes showed how the design software and static analysis software can be successfully combined. It was proven that iterative algorithms, which generate and examine tens of thousands of structures in a matter

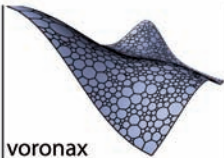
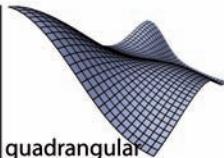
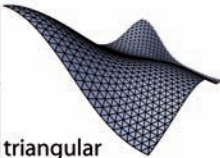
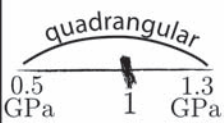
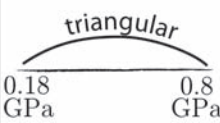
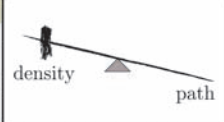
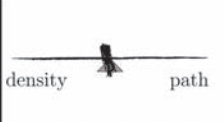

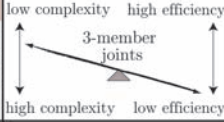
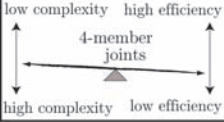
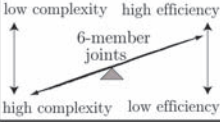
of hours, can be made. One complete procedure for a grid structure in our GAs optimization algorithm has many steps: 1. Select the parents, 2. Produce a child, 3. Mutate and fix unacceptable chromosome 4. Decode the chromosome (get Voronoi points on the surface), 5. Calculate Voronoi diagram from Voronoi points, 6. Relax Voronoi diagram over the NURBS surface to gain Voronax structure, 7. Prepare Voronax structure for FEM analysis (define nodes, elements, cross sections, support, loads), 8. Analyze structure (FEM calculation), 9. Evaluate structure (calculate fitness according to the chosen fitness function and data from FEM analysis), 10. Calculate error (according to the penalty functions) 11. Combine fitness value and error within the final fitness scaling. Eleven steps, yet all this is done in 1-2 seconds per grid shell, in an average experiment for a grid structure with around 500 structural members, with a Core2 Duo Intel processor, working on 2.4 GHz and 4GB of RAM memory. Most of that time is used for the Voronoi-Voronax relaxation process, the speed (precision) of which can be easily controlled. If we think about the exponential growth of the processor speed, we can imagine that we will perform optimization processes, that we did in 10 hours now, in real-time one day soon.

It was shown how the Genetic Algorithms, as a known and accepted optimization method, can be used to optimize free form grid shells. It was explained how to do that in detail, and the steady convergence of every optimization process demonstrated its effectiveness. It was discussed how every aspect can be expanded and what parameters can be controlled for fine tuning of every optimization. We saw how a structural system can always be represented in form of a chromosome, and that structure can always be evaluated in an infinite number of ways. Those two ingredients are all we need to achieve convergence toward some optimal solution.

In order to compare grid shell structures, new methods of evaluation were developed. It was shown how, combining design and FEM software, we can automatically add all the Von Mises stresses in the structural members, or calculate total displacement in all joints, so that we can compare the entire grid shells.

Maybe the most important achievement is the comparative analysis of different types of grid shells. It was shown how the density of the grid, the orientation of structural members and the creation of structural paths, can influence the statical efficiency of the structure. Not only for a single type of grid shell, but also what influence those characteristics have on different types of grid structures generated over the same surface. Therefore, for someone who wants to design a grid shell over some free form surface, it is clearer what the trade-offs are, how they can approach the concept, and how they could design a statically optimal structure using different patterns, or combining them. This comparison is general, and cannot be de-

scribed with exact numbers, since the statical efficiency depends on many factors, like the shape of the surface, load and support combination, material and cross-section of the members, stiffness of the joints, etc. However, this research helped to gain a better understanding of different grid shell types, and to start classifying them, like depicted in Figure 7.1. The values in the table reflect the conclusions that can be drawn from the experiments done within the presented research, i.e., with the surfaces and parameters presented so far. Therefore the gains can probably be even larger with different settings and different surfaces.

	 voronax	 quadrangular	 triangular
<b>GAIN</b> differences between the regular and the optimized solutions in <b>Von Mises stress</b>	20-400%	10-60%	1-20%
<b>GAIN</b> differences between the regular and the optimized solutions in <b>displacement</b>	30-1000%	10-200%	3-50%
<b>COMPARISON</b> comparison of the stress, generated with different types of grid shells, with the same number of structural members	voronax 1 GPa	 quadrangular 0.5 GPa    1 GPa    1.3 GPa	 triangular 0.18 GPa    0.8 GPa
<b>DENSITY vs PATH</b> influence of the cell density and structural paths on the statical efficiency	 density    path	 density    path	 density    path
<b>JOINTS</b> rigidness - complexity of the joints is proportional to the statical efficiency	 3-member joints low complexity high efficiency high complexity low efficiency	 4-member joints low complexity high efficiency high complexity low efficiency	 6-member joints low complexity high efficiency high complexity low efficiency
	combining all of the patterns can lead to the optimal solution, considering statical and optical aspects, as well as production costs		

**Figure 7.1:** Characteristics of different grid shell types, generated over different free form surfaces. The table reflects conclusions drawn from the optimizations performed as a part of the presented research.

We can see that the more structural members can move (like in the Voronax structure) the larger the gains from the optimization can be, in generated stress or total displacement. If we generate different types of grids with the same number of structural members, then the triangular grid shell will always have a smaller amount of generated Von Mises stress than the Voronax structure, and the performance of the quadrangular structure will depend on the con-



ditions. The importance of the structural paths is naturally the largest where there are three basic directions in the grid (triangular grid), less important with two directions (quadrangular grid), and in the Voronax structures the density of the cells is decisive, since no structural paths are formed. The complexity of the joints (and the cost of their production) rises with the number of members connected in them, and also the stiffness and stability of the structure. Eventually, with the use of the optimization methods presented in this research, a mixture of all patterns in one grid could be the best solution for a given free form shape.

### Summary

Grid shell structures that were generated in our experiments should be seen as a proof that, for different set of input parameters, it is always possible to optimize geometry and topology of a structure and gain statically more efficient systems. The efficiency of the optimization does not depend on the art of the input parameters. We have seen that the gains can be achieved with any kind of pattern, whether it was triangular, quadrangular, hexagonal or Voronax. In the same manner, for every type of support or load combination there is always a space, here referred to as *design space*, in which we can find our solution. Genetic Algorithms can offer a solution that is *at the border* of that design space. In that way it gives us a direction in which we can move in order to achieve an optimal design (according to specific criteria). We can then choose how close are we going to approach it, respecting different restrictions that are common in architecture. With some standard spans, loads and member cross-sections, we saw that we can generate structures that have even 4 or 5 times less total amount of stress, and up to 10 times smaller amount of joint displacements, only with the geometrical rearrangement of structural members.

We have seen that evaluation comes in a form of fitness function. It is usually a mathematical function that combines different parameters into one value used to compare different individual solutions. There is an infinite number of possibilities to define them, and that is the freedom that GAs offer and their main advantage. In our research, some of the basic goals were investigated, like minimization of stress and displacement, or enhancement of stability. However, it was made clear, that in conjunction with the FEM software, we can easily define our own, original and creative, specific functions for single-objective or multi-objective optimization. It was additionally shown that any restriction, that comes from manufacturing possibilities, cost differences, design reasons, etc., can be imposed on our solution with the use of penalty functions. In that way, it can offer a solution within any set of boundaries that we define.

---

Another important factor in engineering practice is intuition. That is however nothing more than a large accumulated experience that we use to make conclusions and find optimal solutions. Software applications, like the presented GAs optimization method, will make us gather the experience, learn how to design structures over free form shapes and, eventually, that will lead to the development of something that we can call an engineer's intuition for free form structural design. It is interesting to see how even with the simplest experiments it is very hard to guess what the optimal solution is at the beginning, but once the GAs offer their fittest individual it is *obvious* that it is the right solution. In some cases it is not that obvious, but it still leads to the gathering of knowledge and experience that will help us deal with free form geometry in architecture much better than we did so far.

### Voronax - The Bio-Grid structure

We can say now with certainty that grid shells will have a very broad application with the unification of structure and facade and with the expansion of free form architecture. With Voronax structures we can easily adjust density and therefore have more freedom in optimizing them statically. With uniform, regular pattern structures there are big restrictions, since they always form guide lines (structural paths) that have to be smooth in order to look nice. In Voronax structures they don't exist and the rearrangement of cells to fit statical conditions doesn't *hurt* the structure optically, and it always gives it an authentic Natural look.

On the other hand, the non-existence of the structural paths gives Voronax often worse statical performance in comparison to the regular ones. Additionally, Voronax structure optimized for one load combination doesn't necessarily behave well with other loads. Therefore, it is fair to say that in the larger part of the experiments, the total amount of Von Mises stress of an optimized Voronax structure was much bigger than in a uniform triangulated grid shell. That was expected because of the large stiffness that rigid triangles bring to the structure. Joints with six members are also one of the reasons why the triangular structure performs better. However, when the production costs are considered, it is less complicated to manufacture joints with 4 members, or even 3 members connected. That difference in cost has to be considered when choosing a suitable pattern for our grid shell. When compared to quadrangular grids, there were cases when Voronax had less total amount of stress, and cases where quadrangular grid was better, depending on the shape and the input parameters. That is why it is hard to imagine Voronax *replacing* regular structures immediately. Their use would have to be justified with the design intentions. With the steel structural members and production technologies to-

day, if we want the cheapest and the most effective structure, it looks like the Voronax loses the competition with a triangular grid shell for now. That is why the focus of this research is not on introducing Voronax structures as a replacement for all others grid solutions. The focus is on using them to show how a grid structure over some predefined free form surface can be optimized, regardless of the grid pattern. The aim is to show that if we change the grid density, if we rearrange structural members, we can increase stability or save material. And with the help of Voronax and Genetic Algorithms we can see how that rearrangement should look and use it for any pattern.

A few attempts were shown where the optimization with the triangular (Delaunay triangulation and Delaunax) or quadrangular pattern was performed, directly, without the help of Voronax. But the intentions were not easily *readable*. The regular patterns are simply too rigid and restrictive and cannot generate different densities as easily as Voronax can. The solutions offered are also not *ready to use*, because they are usually optically not acceptable. Regular patterns are optically very sensitive, forming different guide lines and paths, and therefore have to be designed carefully and usually *manually*. Doing an optimization with the Voronax pattern, and then transferring that knowledge to the grid we want to build, proved to be a much more efficient method. Voronax pattern always shows a clear picture of the optimal density disposition, and then we can even mix several patterns in the design process. We can leave the triangular structure where it is needed to stiffen up the grid and switch to quadrangles, pentagons, hexagons, etc. where the Voronax optimization resulted in smaller cell density. We can also quickly evaluate our grids in a matter of seconds, thus speeding up the testing process.

One can wonder then, why Nature didn't develop triangular structures, since they are *better* than the irregular Voronoi pattern. Nature uses fiber materials which are more efficient than isotropic materials. It also usually optimized its structures by starting with a solid model, and than *deleting* material until it developed into a grid. The connections formed in that way are practically cast, and therefore they are much stiffer than the joints we can achieve with steel connections. The conclusion can be made that, with that level of material efficiency, triangular structures, with their dense structure and 6-member joints, would be a waste of material. A much lighter, Voronoi or Voronax, type of structure could then take the pressure and the tension efficiently. Therefore there is a big chance that, with the development of new materials and the development of Nature inspired structures, the time of Voronax structures will come, and on its way there are two main obstacles: Face planarity and Stability.

---

## About Face Planarity

In free form grid shell design there has been a lot of research done in order to find geometrical solutions with planar faces (planar grid polygons). It was one of the reasons why grid shells were mostly triangulated (since triangle is always planar), and why the shape was limited to translational surfaces when we wanted to create a quadrangular grid. There is also a respectively easy way to obtain a hexagonal and pentagonal structure by *cutting* the surface with planes. All that is done so that we can make flat glass panels and cover our grid shell roof structure with them. The production of single-curved and double-curved glass tiles was avoided due to great costs.

The largest problem with face planarity is that when we have that condition, statical optimization is very restrained. If we optimize triangular structures, as shown, the gains are small in comparison to the other ones. With quadrangular structures, we are limited to a small number of planar face solutions, and statical optimization becomes even more limited. With planar hexagonal meshes, there is practically no room for the change of grid density, since the planarity condition is too geometrically restrictive.

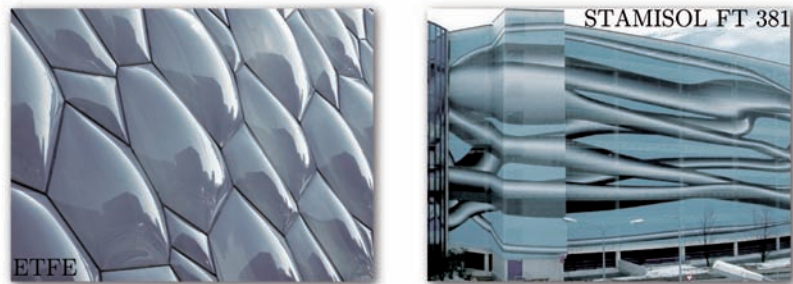
With Voronax structures over free form surfaces, almost all cells would have to be covered with double curved elements. It would make no sense to optimize the structure according to the curvature of the cell surface for two reasons. First, the process would lead to the optimal solution with greater density of the cells in the areas with large curvature, and smaller density in the areas with small curvature (or flat areas). That however leads to an optical optimization, and can hardly be combined with the statical one. Even if we did that, the second reason against it is that we would still end up with solely double curved panels. From the production point of view, that wouldn't make any difference, since the panels would all have respectively small (and similar) curvature and their production costs would be the same.

Unfortunately, the conclusion has to be made that, with the materials and production capabilities today, flat panels and statical optimization do not go hand in hand very well. But there are new solution in sight.

**Solution** First of all, it is important to remind ourselves that the structure and facade are becoming unified into one single-layered *skin*. The grid shell structures are therefore becoming an universal structural system that offers an *envelope* for the entire building. As such, there are lots of other facade materials that can cover the structure. If we take metal, for example, the costs of double curved panel production are not that big in comparison to glass, making Voronax structures acceptable. Especially if we use Genetic

Algorithms to optimize the structure and reduce costs by reducing the material needed.

Second, the method for production of double curved glass will also become cheaper and acceptable as the technology is developed. But more importantly, we are witnessing the development of new materials, like textile and plastic ones. One example is the successful use of ETFE (Ethylene tetrafluoroethylene) that has no problem with double curvature or cell size, as it can be seen in Figure 7.2 (left) on the Water Cube project by Herzog & de Meuron (although ETFE is still no replacement for glass due to different physical and optical characteristics). In the same figure, on the right, we can see a  $20 \times 200m$  facade of the Vienna Airport car park, covered with Stamilol FT 381 printed textile. Those examples represent only a few possible solutions from many to come. That is why, instead of spending our energy to fit our structural solutions into the boundaries defined by manufacturing conditions, we can use it to create new manufacturing methods and develop new materials.



**Figure 7.2:** New materials suitable for free form grid structures

### About Stability

Since triangle is the stiffest form, triangular grid shells are the most stable ones. Quadrangular grid structures are therefore very often stabilized with diagonal bracings, to prevent skewing of the rectangle. The solution for the problem of stability lies in the clever joint design.

**Solution** In a free form grid shell structure, joints (nodes) are the most complicated parts to solve, geometrically and statically. The joints used today in steel free form grid structures are also extremely complicated and expensive to produce. A significant research has to be done to develop new geometrical solutions and to test new materials. The joint is the most important part of a grid shell and clever solution could increase stability, making structures like Voronax acceptable. Voronoi and Voronax grids generally have joints where only 3 members meet, which is easier to produce than

---

the 6-member joint of the triangular grid, but the stability condition is a great challenge that has to be met. There are some examples of successful Voronoi-like structures, as shown in the example of Water Cube (Figure 7.3), but their statical efficiency is still debatable. With the arrival of new materials, like the fiber reinforced ones, we will be able to make stiffer connections that would prevent torsion and skewing in single-layered grid shells. With the parametric programming, considering the fact that the joints in free form structure are all unique anyway, we can develop tools that will parametrically design the geometry of each joint to meet the forces in the best possible way. All of this is a very interesting matter and will be subject of future research.

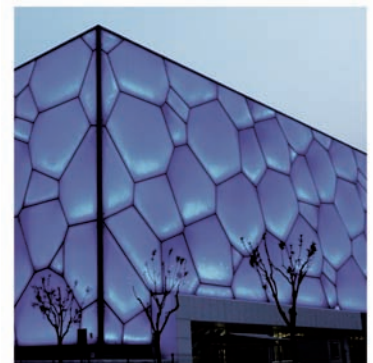
### The Future

Up until the recent past, FEM analysis of any structure was a complicated task that needed lots of preparation and calculation time. Today, with the exponential growth of the processor speed, not only can we calculate complicated structures in a matter of seconds, but we can create an iterative process that can do that millions of times. In those iterative algorithms the future of structural design is being born. As mentioned before, our part will only be to ask the question right, and the answer will be given by a machine. It is irrelevant what kind of optimization algorithms will be used, but for now the stochastic ones, like Genetic Algorithms, promise a lot. One of the main reasons is their generality, i.e., applicability to practically any optimization problem. It was proven that, if they are set properly, they can be extremely efficient.

Most of the possible applications of the merger of graphics and static analysis have not yet been investigated, and surely offer solutions beyond our imagination. It opens up huge possibilities, and represents the future of structural design. Many procedures performed by engineers *manually*, can now be automatized. Computers can be used to generate thousands or millions of possible combinations and solutions, something that we as human beings could never process. The only thing that we have to do is ask the question properly (which is not an easy task!) and then we can use different optimization methods to get the answers.

As mentioned before, a large effort will be directed toward smart, parametric solution of the joint geometry. Parallel to that, new materials have to be tested and applied in the grid shell structural design.

As far as the optimization of single-layered grid shell structures with the use of Genetic Algorithms goes, this is just the beginning. Every single aspect of the application written is made in a way that it can be expanded easily. For example, the *coding* and *decoding* part can be altered to comprehend any kind of 3D



**Figure 7.3:** Water Cube, Beijing, 2008, Herzog & de Meuron



structure. We can imagine an optimization of the structure for the entire free-formed skyscraper, or even an optimization of the fiber and molecular structure in materials (with the further development on nanotechnology). This can be done with simple alterations of the presented method. Deployable structures are becoming more popular, and with the proper set of dynamic fitness functions we can generate beautiful and efficient bio-structures, that will be able to transform and adjust to the environment conditions, as well as plants or animals can.

The tools that we have today are slowly starting to get ahead of us. Breakthroughs in science are challenging our imagination every day. Buckminster Fuller wrote about new inventions and how they have to wait 50 years until they get applied in the building industry [14]. I think that time-span is getting shorter and shorter now. New generations of architects and engineers, that have to know programming languages as well as they know mathematics or art history, are being educated right now. The future of structural engineering is in their hands.

# Appendix A

## Application

### A.1 User Dialog

In order to illustrate somewhat abstractly written explanations, we can see how the dialog in our application looks like. Namely, we'll take the most complicated dialog for Voronax structures, and we will see how it looks when the optimization process is performed and the solutions can be drawn and examined. Other patterns, like triangular, quadrangular or hexagonal, have their own, similar but simpler, dialogs. The Voronax dialog is presented in Figure A.1.



Figure A.1: Voronax Dialog

Without going into the detailed explanation of each aspect, because that has already been done, a global overview is offered. In the upper left corner we can see the information about the number of nodes on the surface and on the  $U$  and  $V$  surface edge, that our generated solutions will have. Underneath, a *minimal allele distance* on the surface and on the edge defines the minimal distance that two Voronoi points can have. If, in the generation process, points with smaller distances are generated, the code

changes the problematic point until it fits the constraints. In the lower left corner we can see the standard GAs settings. Population size determines the number of individuals in one generation. Then we can define the number of generations we want to generate and crossover and mutation probability. On their right, there is a space to define the crossover type, but in almost all experiments the *uniform crossover* is used, since it showed the best results. In the upper middle part we can see the definition of the pattern. We can choose Voronoi, Voronax or Delaunay, since they are all based on the same principle of Voronoi diagram generation. Other patterns, as mentioned, have separate, simplified dialogs. Next to the pattern definition we can see the choice for initial population. If we check *random*, the zeroth generation will be randomly created. If we check *txt*, the generation will be imported from a prepared file, with the chromosomes of the individuals for one entire generation. This is used to continue optimization processes. Whenever the process ends, we can extract the last generation and start again from there, thus continuing the optimization process as if it had never been interrupted.

On the right upper side we see the evaluation settings. Every setting has a number, a code, that represents a specific set of parameters. Object function type refers to the fitness function defined, and underneath it are the section type, the support type and load case types. In Figure A.2 there are four tables that present some of the codes for different fitness functions, cross-sections, support and load types, used in our experiments.

fitness function		cross-section type		support type		load type	
0	Sum Von Mises	0	IPE 140	0	all 4 sides	0	gravity - G
1	Average Von Mises	1	IPE 100	1	only U sides	1	G+nodes, Z,-1 KN/m2
2	Average Length	2	IPE 80	2	4 corners	2	G+nodes, Y,-1 KN/m2
3	Sum of Moments	3	REC 200,70,10,10	3	4 mid-points	3	G+nodes, X,-1 KN/m2
4	Sum of Displacements	4	REC 140,70,10,10	4	4 sides, pinned	4	G+surface, Z,-1 KN/m2
5	Load Buckling Factor	5	REC 110,70,10,10	5	one U side	5	G+half surface, Z
6	Frequency	6	REC 70,70,10,10	6	U sides, z free	6	wind , X, -3 KN/m2
	...		...		...		...

**Figure A.2:** Examples of the coding of some of the fitness, cross-section, support and load types used in the experiments

Finally, in the bottom right corner, penalty functions are defined. We can chose 1, 2, or 3 functions and for each one we can choose the type and its limit value. By clicking *Breed*, we are asked for the names of 3 files where the data should be saved and the algorithm starts.

## A.2 Export Files

In Section 4.1.1 it was explained how the output data of the optimization process consist out of 3 separated files. Without repeating the description of the files, the following Figures A.3, A.4, and A.5 show small cutouts. The general file has all of the information about the optimization process at the beginning, followed by a list of all generations and all individuals in those generations with their fitness value, rank and scaled fitness value, as shown in Figure A.3. The chromosome file (Figure A.4) consists of arrays of positive real numbers that are later used in the decoding functions to be transformed into structures. The graph file (Figure A.5) is pretty simple, it has the number of generations at the beginning with the number of individuals in one generation and the size of the chromosome to know how to draw the graphs. After those first three numbers, there are four values for each successive generation, i.e., maximal fitness value, minimal fitness value, average fitness value and sum of all fitnesses in one generation.

```

GA Parameters
-----
Population size      = 50
Chromosome length   = 304
Maximum # of generation = 400
Crossover probability = 0.600000
Mutation probability = 0.001000
Division place      = 152
-----

Recl. Initial Generation Statistics
-----
Init Population maximum fitness = 78444.447386
Init Population average fitness = 53461.980573
Init Population minimum fitness = 45345.186533
Init Population sum of fitness = 2772199.028645
Init Population sum of x = 14.660354
-----

Generation: 1
# string rank x fitness # parents xsite string rank x fitness
0 0.624107 64233.342848 1 0 (9,29) 0 0.516285 65342.768459
1 0.373652 69876.408850 1 (9,29) 0 0.633800 61853.010296
2 0.682168 62801.957887 2 (10,25) 2 0.367089 69128.846486
3 0.708797 62191.344308 3 (10,25) 2 0.746057 59511.799445
4 0.713885 62074.681915 4 (23,11) 4 0.888131 55906.396569
5 0.930949 57097.287244 5 (23,11) 4 0.741092 59637.810609
6 0.000000 78444.447386 6 (24,40) 6 0.455069 66896.193941
7 0.678584 62923.823347 7 (24,40) 6 0.752385 59351.231167
8 0.357170 70534.339256 8 (7,10) 8 0.000000 78444.447386
9 0.902317 57753.842947 9 (7,10) 8 0.691000 60908.997343
10 0.897844 57854.103726 1 10 (7,9) 10 0.607583 63025.864313
-----
Generation: 2 Statistics: max=78444.447386 min=53067.517352 avg=61442.780822 sumFitness=3072139.041100 sumx=33.498273 nmutation=2 ncross=6

```

Figure A.3: GAs general description

```

; 0.218574; 0.530808; 0.618274; 0.501846; 0.258553; 0.730338; 0.696158; 0.334544; 0.274300; 0.335643; 0.79732
75; 0.301309; 0.375561; 0.400830; 0.416486; 0.088717; 0.433515; 0.257454; 0.882138; 0.204474; 0.690817; 0.244
6308; 0.025605; 0.688742; 0.630085; 0.341594; 0.117649; 0.343638; 0.860347; 0.608539; 0.316050; 0.295694; 0.6
; 0.026002; 0.886502; 0.822718; 0.257668; 0.332835; 0.134587; 0.645161; 0.873226; 0.887936; 0.312876; 0.43311
82; 0.450423; 0.455944; 0.869564; 0.815760; 0.534196; 0.590136; 0.040010; 0.582812; 0.540208; 0.621235; 0.887
6513; 0.196722; 0.534745; 0.400433; 0.446821; 0.029023; 0.662740; 0.487320; 0.736412; 0.620869; 0.171514; 0.4
; 0.015809; 0.153233; 0.554399; 0.746178; 0.978912; 0.375622; 0.843532; 0.136845; 0.651479; 0.966338; 0.81835
63; 0.826746; 0.545335; 0.324107; 0.837733; 0.285470; 0.860714; 0.226844; 0.811579; 0.632313; 0.071810; 0.668
8521; 0.196448; 0.919218; 0.502113; 0.157292; 0.506546; 0.434736; 0.464064; 0.209143; 0.149876; 0.273812; 0.1
; 0.973113; 0.966063; 0.084414; 0.183294; 0.787805; 0.981262; 0.748894; 0.796106; 0.921812; 0.289041; 0.81521
22; 0.109775; 0.016388; 0.888516; 0.396374; 0.300363; 0.404706; 0.845332; 0.154759; 0.156468; 0.389843; 0.387
8858; 0.793023; 0.381268; 0.967833; 0.417249; 0.541368; 0.789453; 0.312387; 0.056642; 0.040284; 0.472884; 0.4
; 0.371838; 0.821039; 0.496384; 0.125645; 0.599445; 0.610218; 0.230293; 0.066683; 0.726615; 0.796350; 0.85137
11; 0.433393; 0.089114; 0.326456; 0.254280; 0.290872; 0.065493; 0.812464; 0.675436; 0.516678; 0.977355; 0.317
6196; 0.885586; 0.177679; 0.863643; 0.319468; 0.308908; 0.075930; 0.650349; 0.555986; 0.686361; 0.051759; 0.2
; 0.608631; 0.764336; 0.815699; 0.389080; 0.138310; 0.026765; 0.819483; 0.967009; 0.747948; 0.833735; 0.49308
52; 0.311472; 0.690542; 0.372845; 0.890469; 0.075014; 0.515580; 0.091494; 0.529954; 0.639180; 0.767754; 0.323
4102; 0.296335; 0.176946; 0.806024; 0.767693; 0.851192; 0.649678; 0.743461; 0.585772; 0.414045; 0.185217; 0.6
; 0.035646; 0.339000; 0.387249; 0.699271; 0.898221; 0.880337; 0.745415; 0.570849; 0.767449; 0.853786; 0.98193
40; 0.644490; 0.361583; 0.945006; 0.630726; 0.845363; 0.147343; 0.619953; 0.917447; 0.826075; 0.356517; 0.517
1661; 0.124943; 0.945860; 0.508866; 0.808863; 0.243873; 0.526994; 0.633351; 0.786554; 0.392071; 0.434400; 0.0
; 0.457533; 0.438246; 0.742424; 0.932279; 0.031678; 0.298318; 0.588397; 0.594226; 0.010681; 0.773309; 0.56498
10; 0.789727; 0.390332; 0.946043; 0.924436; 0.398206; 0.214362; 0.160192; 0.318949; 0.816401; 0.015870; 0.797
1949; 0.245277; 0.914365; 0.745628; 0.278970; 0.470107; 0.405286; 0.927091; 0.320841; 0.939299; 0.152989; 0.4

```

Figure A.4: Solution chromosomes

```

171
50
304
78444.447386
55513.910910
55443.980573
2772199.028645
#
78444.447386
55513.910910
64867.648224
3243382.411195
#
78444.447386
53067.517352
61442.780822
3072139.041100
#
78444.447386
50695.705004
61733.879010
3086693.950519
#
69356.494398
45710.148778
59135.538125
2956776.906268
#
71536.044769
45710.148778
58301.518500
2915075.824981
#
72203.117842
46045.483402
58965.098546
2948254.927302
#
72203.117842
48959.931069
60144.028791
3007201.439551
#
67221.697488
48604.100359
59112.537798
2955626.689906
#

```

Figure A.5: Graph information

### A.3 Draw Results

After the optimization process is done, and all three files are filled with data containing the information needed to reproduce any of the solutions, we can see how we can extract and draw our solutions. In Figure A.6 there is a dialog used for drawing the individual grid shells.

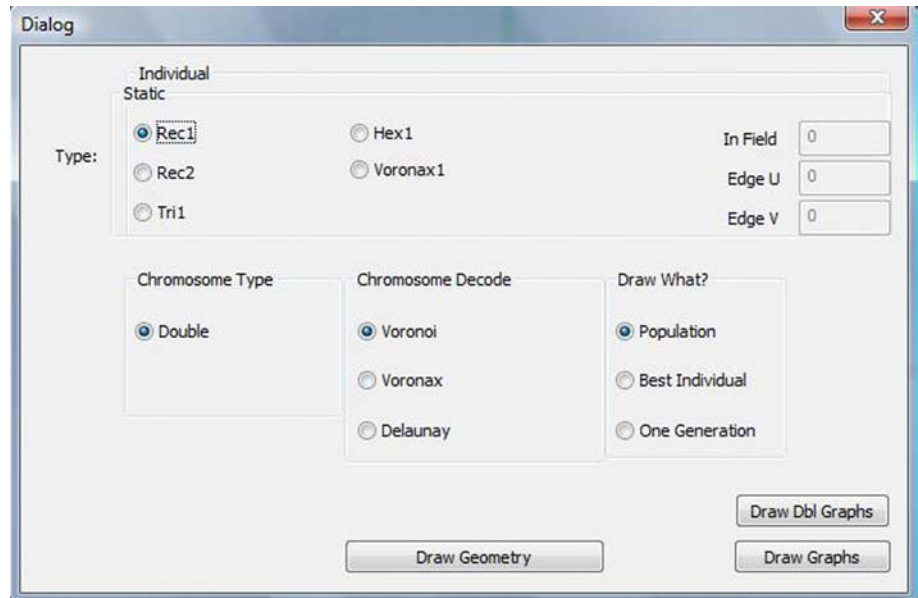


Figure A.6: Draw Dialog

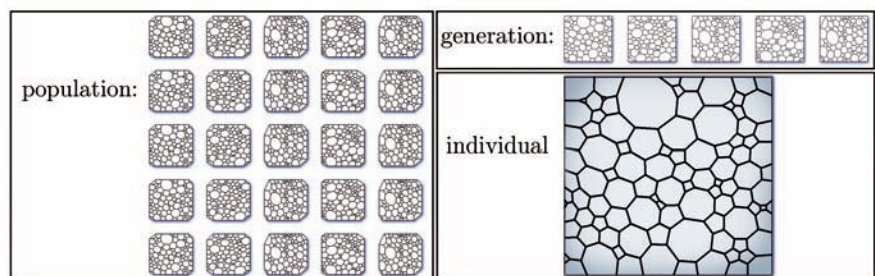


Figure A.7: Population, Generation, Individual

In the upper part it can be seen that we can choose the pattern. This is important, because if we want the chromosome to be decoded properly we have to choose the right pattern. In that way we are actually choosing the decoding function. Top right shows that in the case of Voronax we have to define the number of points on the surface and on the edges as it was defined in the optimization process, so that the chromosome is properly decoded. In the middle of the dialog, there is a second aspect of decoding. It is important to remember that every set of points can be generated as Voronoi, Voronax or Delaunay. The pattern determines just the

disposition of points (Voronoi seed), gained after the decoding of the chromosome, but we need additional information that tells us if we want to generate Voronoi from those points, Voronax, or if we want to connect the seed to obtain Delaunay triangulation.

On the right of the dialog we can chose if we want to draw the entire population, one generation or one individual, as depicted in Figure A.7.





# Appendix B

## Code Structure

The complete software consists of thousands of code lines. But in order to illustrate how it works, the basic individual's data structure and the global algorithm scheme will be explained here.

### B.1 Software and Methods Used

The complete algorithm is written in form of a plug-in for Rhinoceros 3D. Rhinoceros (Rhino) is a commercial software that uses NURBS geometrical representation and it is one of the best programs for the design of free form shapes, thus used mainly nowadays for the design of free form architecture. The program has its own programming language called Rhinoscript, based on VBA (Visual Basic for Applications) but for our purposes it was too slow and not flexible enough. The second option that Rhino offers is to write a plug-in in VB.NET or Visual C++. Since the Rhinoceros is actually written in Visual C++, the idea was to write a plug-in using the C++ language thus achieving the best performance by calling all the Rhinoceros functions directly.

When that was settled, the FEM static analysis had to be included. The decision was made to use the Oasys GSA Analysis commercial program. It offers an easy way of calling its basic function with the use of OLE Automation. That is a mechanism that provides an infrastructure whereby applications can access and manipulate shared *automation objects* that are exported by other applications, in this case GSA Analysis. Basically, it was possible to call the program directly from our C++ code, analyze structure and read the results, repeating that step in our GAs iterative optimization process.

## B.2 Data Structures

### B.2.1 Individual

Every individual is represented as a complex structure of informations. All the functions that act upon individuals change those values throughout the process and use them for all GAs operations, including the generation of output files. Figure B.1 shows the characteristics of one individual, i.e., the data types used to hold the needed information. On the right of the figure there is a short,

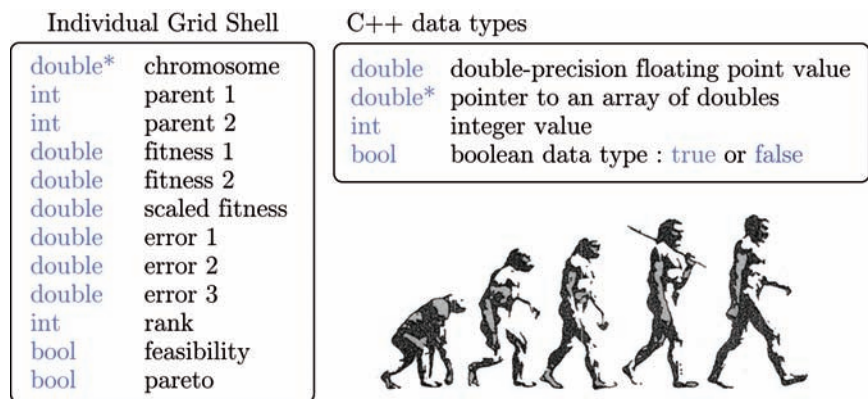


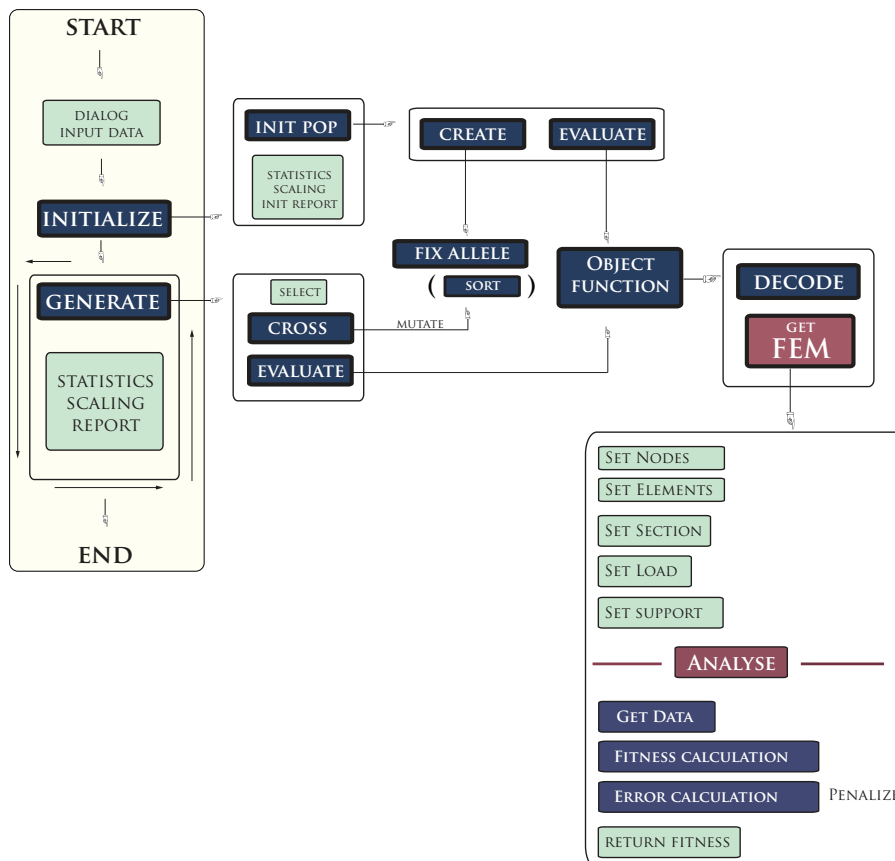
Figure B.1: Individual’s data structure

basic explanation of C++ data types applied (detailed explanation of the types can be found in [46]). As we can see, first there is an array of numbers representing the chromosome as described in section 4.2. Information about individual’s parents follows. The integer values are actually their identification numbers which come from their position in the generation. The fitness values follow. In this case there are two, representing the individual that can be used for two-objective optimization. This can vary from 1 to  $n$ , depending on how many fitness functions we want to include into the GAs process. The same is valid for errors, since there can be any number of them. Each individual holds the information about how much it exceeds the limit prescribed by every penalty function defined. If the solution is penalized or not, depends on the information in *bool feasibility* that describes each individual as feasible or infeasible (true or false). Between the fitness values and the error values we see a *double* data type holding the scaled fitness value. In the case of multi-objective optimization there is also only one value (as in the single-objective optimization), since, as described in Section 4.5, we scale the fitness according to its domination state, considering all fitness functions at once. Information about that state is kept in the *bool pareto*, helping us know how to scale the individual, i.e, as dominating or dominated solution. The *integer* value *rank* is calculated after all of the solutions in one generation are compared and ranked. This can be used in different selection procedures.

However, in the research, the rank didn't find any particular use, since the selection methods based on it are elitist, and therefore avoided, as described in Section 4.1.2.

## B.2.2 The Algorithm

In Figure B.2 there is a simplified scheme of the entire GAs optimization process. The part on the left (painted yellow) represents the main function. After the input information is supplied over the dialog, the initial generation is generated and evaluated (initialize). The process then enters a loop where, as described in Section 4.1.1, selection, crossover and mutation are performed (generate). Fitness scaling follows together with the calculation of *statistics* (best fitness, worst fitness, average fitness, sum of fitness) and writing of data into the text files (report).



**Figure B.2:** Algorithm structure

Now let's look at the parts right of the main function. *Fix allele* refers to fixing of all the errors that can happen in a chromosome, i.e., in the values of its genes (alleles). One of the common errors, for example, is the generation of double values that would result in double Voronoi points, thus leading to errors in the algorithm that calculates the Voronoi diagram. Therefore, there are functions that

check the chromosomes and fix them if they have to. If the position of values is important, the chromosome can be sorted at this point and values can be arranged from the smallest to the largest or vice versa. However, in our research, as explained in Section 4.2.1, the position of genes in the chromosome made no difference.

The evaluation implies a call to an *object function* that leads to decoding (described in Section 4.2.2) and finally a set of functions described in the figure as *Get FEM*. In the right lower corner of the Figure B.2 the names of those functions are listed. In preparation of the model for FEM analysis we have to use the generated grid shell structure that comes from the decoding functions and turn it into a FEM model. We set and numerate the nodes automatically out of the grid's points and structural elements out of its generated lines. Then we set the load and support according to the chosen type and let the program do the FEM analysis. Afterward, the data needed for the calculation of the fitness value are obtained (stress, deformation, buckling factor, etc.) and the fitness value (or values) is calculated. The error calculation follows, and the functions returns both of the values (fitness and error). They continue their way to the fitness scaling in the main function, and the process repeats then by starting selection again.

That would be a brief explanation of the process. It shows how all the steps, described thoroughly so far, fit into one big picture.

# Appendix C

## Cell Recognition

For the definition of the surface load, like glass weight, snow, etc. we need the information about the cells in our structure. In a *polygon mesh* data structure that information is automatically available, but since we are dealing with structures consisting of polygons with more than 4 edges, an additional algorithm had to be made to collect that data. The algorithm, constructed for this research, can be roughly divided in two parts: collecting node information and *circling*.

### Getting Node Information

The first step is to collect the information about connections in the structure. Each node, as well as every member, have numbers assigned to them. The idea is to observe each node and collect the information about the members connected to it, as shown in Figure C.1. With the algorithm that compares all the points with all the member ends, we can gather the required data.

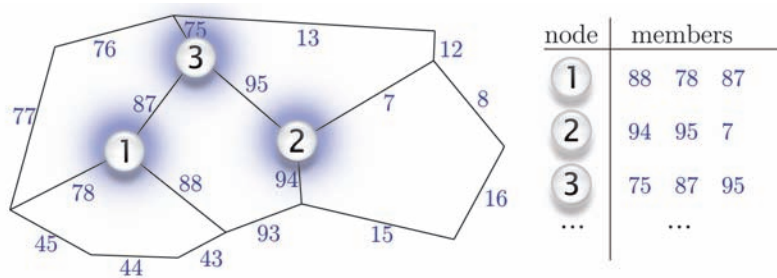


Figure C.1: Gathering connection information

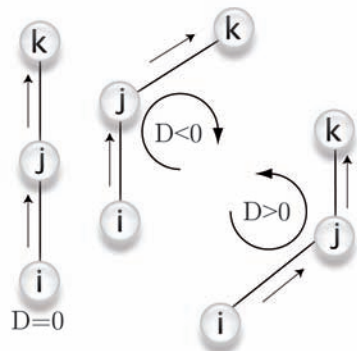
### Circle Around

After we have all the connections, we have to check if there are nodes with less than three connections. Those cases would imply that there is some error, and the algorithm would stop, since the results wouldn't be correct. When the structure is checked, next



step can be performed, i.e., an algorithm that starts from each point and *circles around* to define each cell. This can be done with the help of 2D determinant. Namely, three points in a plane  $i = (i_x, i_y), j = (j_x, j_y), k = (k_x, k_y)$  are defined with their  $x, y$  coordinates and using that information, we can calculate their determinant:

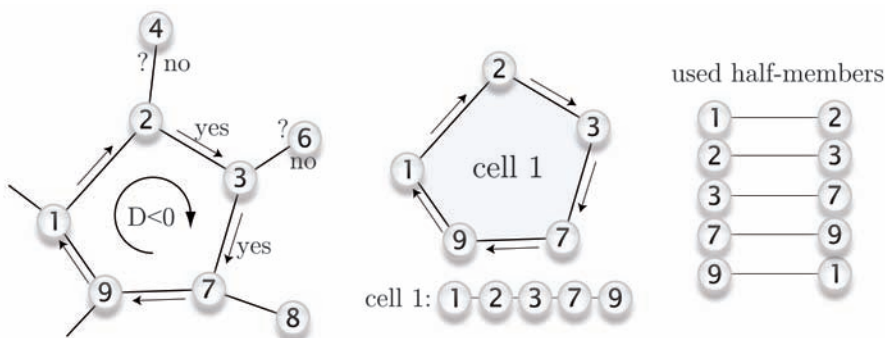
$$D = \begin{vmatrix} 1 & i_x & i_y \\ 1 & j_x & j_y \\ 1 & k_x & k_y \end{vmatrix} \tag{C.1}$$



**Figure C.2:** Polygon orientation

If the determinant is 0, then all three points lie on one line. If the determinant is negative, then the point array shows clockwise orientation (negatively oriented), and if the determinant is positive, points show counterclockwise orientation (positively oriented), as depicted in Figure C.2. If we know that all nodes in our structure can be defined over their  $uv$  parameters as  $i = (i_u, i_v), j = (j_u, j_v), k = (k_u, k_v)$ , we can use this 2D algorithm to determine the orientation of every three point group in the grid.

If we have a situation as depicted in Figure C.3, we can start from node 1, take one of its connections (2 for example) and search for a point that gives a clockwise direction. We check the node 4, conclude that it is counterclockwise, and go to the node 3 which satisfies our condition. Then we start from node 2 and repeat the process thus getting new array 2,3,7. This is continued until the third point in a group (7,9,1) is recognized as a starting point (1) and the cell is *complete*.

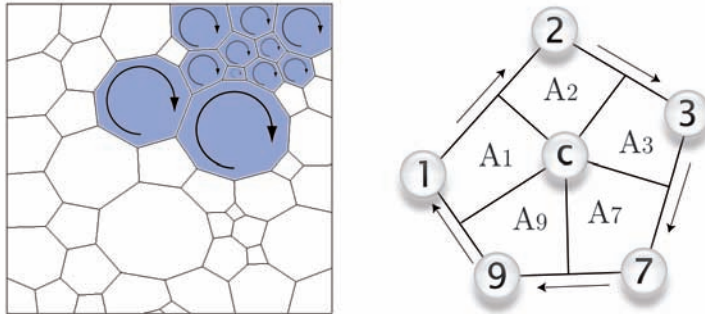


**Figure C.3:** Cell information

Each member in the grid, with endpoints  $A$  and  $B$ , can be represented by two half-members, or joint of two vectors  $AB$  and  $BA$ . Each of those half-members in the structure belongs to one cell only, and by marking them in a specific manner, we can ensure that there are no doubled cells. If the half-member is marked as used, that means that the cell that it belongs to is already recognized and that it is not necessary to continue with *circle around* algorithm. Of course, if  $AB$  is used, that doesn't mean that  $BA$  can't be used. This is all possible only with convex polygons, but luckily one of

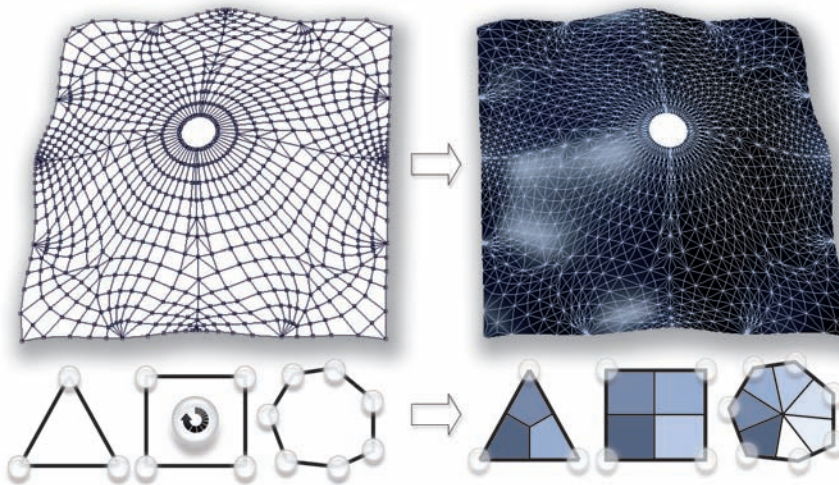
the main characteristics of Voronoi and Voronax diagrams is that they are always convex.

Each cell holds the information about its nodes and it can use them to estimate the surface and assign to each node its part. Therefore, each node  $i$  in the cell has a certain surface area  $A_i$ , expressed in  $m^2$ , which is then multiplied with the load applied on that surface.



**Figure C.4:** Surface partition

In the cases where there are more than three connections in one point (which is rare for Voronax structures) we only have to check which one has the bigger value for its determinant (or smaller angle) and take that point as the next one in our *circle around* algorithm, since that point belongs to the *inner* polygon. Everything else remains the same. Figure C.5, on the left, shows one of the optimized grids used in the experiments. The net created by recognizing the cells, and drawing a mesh over them, is shown on the right. This demonstrates that the algorithm can basically detect cells of a grid structure with any combination of pattern, generated over a free form NURBS surface.



**Figure C.5:** Face recognition works for any type of grid shell



# Bibliography

- [1] Chinchuluun A., Pardalos PM., and Migdalas A. *Pareto Optimality, Game Theory and Equilibria*. Springer Berlin, 2008.
- [2] Domingez A., Stiharu I., and Sedaghati R. Practical design optimizations of truss structures using the genetic algorithms. *Research in engineering design*, 17(2):73–84, 2006.
- [3] Hopgood A. and Mierzejewska A. Transform ranking: a new method of fitness scaling in genetic algorithms. In *Research and Development in Intelligent Systems XXV*, pages 349–354, 2008.
- [4] Th. Bulenda and Knippers J. Stability of Grid Shells. *Computers & Structures*, 79:1161–1174, 2001.
- [5] John Chilton. *Heinz Isler (Engineer’s Contribution to Architecture)*. Thomas Telford Ltd, 2000.
- [6] Brian Cotterell and Johan Kamminga. *Mechanics of Pre-industrial Technology: An Introduction to the Mechanics of Ancient and Traditional Material Culture*. Cambridge University Press, 1992.
- [7] Langmead D. and Garnaut C. *Encyclopedia of Architectural and Engineering Feats*. ABC-CLIO, 2001.
- [8] Powell D. and Skolnick M. Using Genetic Algorithms in Engineering Design Optimization with Non-Linear Constraints. In *Proceedings of the 5th International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, 1993.
- [9] Charles Darwin. *The Correspondence of Charles Darwin: Volume 14, 1866*. Cambridge University Press, 1955.
- [10] Charles Darwin. *On The Origin Of Species*. Dover Publications, 2006.
- [11] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry*. Springer Verlag, 1997.
- [12] Heino Engel. *Tragsysteme/Structure Systems*. Hatje Cantz Verlag, 2006.

- [13] Kleiner F. and Mamiya C. *Gardner's Art Through the Ages: The Western Perspective*. Thomson Learning, 2005.
- [14] Buckminster Fuller. *Critical Path*. St. Martin's Griffin, 1982.
- [15] Galileo Galilei. *Dialogue Concerning the Two Chief World Systems*. Modern Library, 2001.
- [16] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.
- [17] Pottmann H., Asperl A., Hofer M, and Kilian A. *Architectural Geometry*. Junius Verlag, 2010.
- [18] Ernst Haeckel. *Kunstformen der Natur*. Marixverlag, 2004.
- [19] Raphael T. Haftka and Zafer Gürdal. *Elements of Structural Optimization*. Kluwer Academic Publishers, 1992.
- [20] Cowan H.J. A History of Masonry and Concrete Domes in Building Construction. *Building and Environment*, 12:1–24, 1977.
- [21] John H. Holland. *Adaptation in Natural and Artificial Systems*. Ann Arbor, 1975.
- [22] Knippers J. and Helbig T. Vom Entwurf bis zur Ausführung frei geformter Netzschalen, eine Prozesskette. *Stahlbau*, 77(1):10–15, 2008.
- [23] Smith J., Hodgins J., Oppenheim I., and Witkin A. Creating Models of Truss Structures with Optimization. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2002*, 21(3), 2002.
- [24] Sack J.R. and Urrutia J. *Handbook of Computational Geometry*. North Holland, 1999.
- [25] Jerry Judelson. *Sustainable Retail Development*. Springer, 2009.
- [26] Deb K. and Gulati S. Design of truss-structures for minimum weight using genetic algorithms. *Finite Elements in Analysis and Design*, 37(5):447–465, 2001.
- [27] Linkwitz K. About formfinding of double-curved structures. *Engineering Structures*, (21):709–718, 1999.
- [28] Rasheed K. *A Genetic Algorithm for Continuous Design Optimization*. PhD thesis, The State University of New Jersey, 1998.

- [29] H. Kardestuncer. *Unification of Finite Element Methods*. Elsevier Science Ltd, 1984.
- [30] Jan Knippers. Ingenieurporträt: Johann Wilhelm Schwedler: Vom Experiment zur Berechnung. *Deutsche Bauzeitung*, 12(4):105–10, 2000.
- [31] Branko Kolarevic. *Architecture in Digital Age - Design and Manufacturing*. Taylor and Francis, 2005.
- [32] Richard Kostelanetz. *A Dictionary of the Avant-Gardes*. Routledge, 2001.
- [33] Ray Kurzweil. *The Singularity Is Near: When Humans Transcend Biology*. Penguin (Non-Classics), 2006.
- [34] Gil L. and Andreu A. Shape and cross-section optimisation of a truss structure. *Computers & Structures*, 79(7):681–689, 2001.
- [35] Gründig L., Moncrieff E., Singer P., and Ströbel. A history of the principal developments and applications of the Force density method in Germany 1970-1999. In *IASS-IACM*, Chania-Crete, 2000.
- [36] Gründig L. and U. Hangleiter. Computation of prestressed cable-nets with the force densities method. In *IASS-Symposium Cable Structures*, Bratislava, 1975.
- [37] Gründig L. and H.-J. Schek. Die Berechnung von vorgespannten Seilnetzen und Hängenetzen unter Berücksichtigung ihrer topologischen und physikalischen Eigenschaften und der Ausgleichrechnung. *DGK Reihe C*, (216):145–158, 1976.
- [38] K. Linkwitz and H.-J. Schek. Einige Bemerkungen zur Berechnung von vorgespannten Seilnetzkonstruktionen. *Ingenieur-Archiv*, (40):145–158, 1971.
- [39] Claus Mattheck. *Design in Nature: Learning from trees*. Springer Verlag, 2004.
- [40] Efrn Mezura-Montes. *Constraint-Handling in Evolutionary Optimization*. Springer Berlin, 2009.
- [41] John Nash. *Non-Cooperative Games*. PhD thesis, Faculty of Princeton, USA, 1950.
- [42] Oasys. *GSA Analysis Help*.
- [43] Nanakorn P. and Meesomklin K. An adaptive penalty function in Genetic Algorithms for structural design optimization. *Computers & Structures*, 79(29-30):2527–2539, 2001.



- [44] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer Verlag, 1995.
- [45] Ingo Rechenberg. *Evolutionsstrategie '94*. Frommann Holzboog, 1994.
- [46] Lippman S. and Lajoie L. *C++ Primer*. Addison-Wesley Professional, 2005.
- [47] Farzad Sadjadi. Comparison of fitness scaling functions in genetic algorithms with applications to optical processing. *Optical Information systems II*, 5557, October 2004.
- [48] Chris Scarre. *The Seventy Wonders of the Ancient World: The Great Monuments and How They Were Built*. Thames and Hudson, 1999.
- [49] Alexander Schiftner. Planar quad meshes from relative principal curvature lines. Master's thesis, Institute of Discrete Mathematics and Geometry, Vienna University of Technology, 2007.
- [50] Mathias Paul Scholz. *Advanced NXT: The Da Vinci Inventions Book*. Apress, 2010.
- [51] Ulrich Schwarz. *Ingenieurbaukunst - made in Germany. 2010/2011*. Junius Verlag, 2010.
- [52] Tim Smith. *Eden*. Transworld Publishers, 2008.
- [53] Hoffmeister F. Bäck T. Genetic Algorithms and Evolution Strategies: Similarities and Differences. *Lecture Notes in Computer Science*, 496/1991:681–689, 1991.
- [54] Yamashita T. and Kato S. Elastic buckling characteristics of two-way grid shells of single layer and its application in design to evaluate the non-linear behaviour and ultimate strength. *Journal of Constructional Steel Research*, 57(12):1289–1308, 2001.
- [55] D'Arcy Wentworth Thompson. *On Growth and Form*. Cambridge University Press, 1992.
- [56] Peter von Bülow. *An Intelligent Genetic Design Tool(IGDT) Applied to the Exploration of Architectural Trussed Structural Systems*. PhD thesis, Institut für Leichtbau Entwerfen und Konstruieren, 2007.
- [57] Kida Y., Kawamura H., Tani A., and Takizawa A. Multi-Objective Optimization of Spatial Truss Structures by Genetic Algorithm. *Forma*, 15(2):133–139, 2000.

# LEBENS LAUF

## ■ Persönliche Daten

Name: Miloš Dimčić  
Geburtstag: 30.07.1982. in Pančevo, Serbien  
Adresse: Hohentwielstr. 58A D-70199 Stuttgart  
Tel: 0176 64 13 71 82  
E-Mail: m.dimcic@itke.uni-stuttgart.de  
milos.dimcic@gmail.com

## ■ Schulbildung

1988 – 1996 Grundschole, Jabuka, Serbien  
1986 – 1995 Gymnasium Uroš Predić, Pančevo, Serbien

## ■ Ausbildung

2000 – 2006 Architekturstudium an der Universität Belgrad  
2007 – 2011 Doktorand am Institut für Tragkonstruktionen und  
Konstruktives Entwerfen der Universität Stuttgart

## ■ Berufspraxis

2007 – 2009 Kooperation mit „Knippers Helbig Beratende Ingenieure“,  
Stuttgart, an den folgenden Projekten:  
-Institute of Peace, Washington, USA  
-EXPO Axis, Shanghai, China  
-Bao'an Flughafen, Shenzhen, China

## ■ Sonstige Qualifikationen

Sprachkenntnisse: Englisch fließend  
Deutsch fließend

Seit 1999 MENSA Mitglied  
(weltweiter Verein für hochbegabte Menschen)

Seit 1990 passionierter Klavierspieler





