

Generation of Structures in Chemistry and Mathematics

Nicolas Van Cleemput

Promotor: prof. dr. Gunnar Brinkmann

Proefschrift ingediend tot het behalen van de graad van
Doctor in de Wetenschappen: Informatica

Vakgroep Toegepaste Wiskunde en Informatica
Faculteit Wetenschappen
2012



Contents

Acknowledgements	vii
-------------------------	------------

Short survey	xi
---------------------	-----------

1 Definitions	1
1.1 Graph theory	2
1.1.1 Graphs	2
1.1.2 Subgraphs	4
1.1.3 Bipartite graphs	5
1.1.4 Edge colouring	6
1.1.5 Plane graphs	6
1.2 Chemical graph theory	7
1.3 Tilings and Delaney-Dress symbols	9
1.3.1 Periodic tilings	9
1.3.2 The flag space of a tiling	11
1.3.3 The flag graph of a tiling	13
1.3.4 Delaney-Dress symbols	13
1.3.5 A geometric interpretation of flags	18
1.3.6 Examples	19
1.3.7 Basic algorithms on Delaney-Dress symbols	23
1.4 Generation algorithms	31

1.4.1	Construction operations	32
1.4.2	Isomorphism rejection	33
1.4.3	Isomorphism rejection by lists	34
1.4.4	Canonical representatives	35
1.4.5	Read/Faradžev-type orderly algorithms	36
1.4.6	McKay's canonical construction path method	36

I Generation of mathematical structures **39**

2 Pregraphs **41**

2.1	Definitions	42
2.2	Cubic pregraph primitives	45
2.3	Reduction operations for pregraph primitives	46
2.4	Generation of cubic pregraph primitives	51
2.5	Generation of cubic pregraphs	59
2.6	3-edge-colourable pregraphs	60
2.7	Bipartite pregraphs	63
2.8	Admitting a 2-factor composed of quotients of C_4	65
2.9	Testing	69
2.10	Results	70

3 Delaney-Dress symbols **85**

3.1	C_4^q -marked pregraphs	86
3.2	Delaney-Dress graphs	110
3.3	Delaney-Dress symbols	112
3.3.1	Enumerating all Delaney-Dress symbols	114
3.3.2	Restricting the tilings	117
3.4	Results	127
3.5	Future work	131

II Generation of chemical graphs **137**

4 Azulenoids **139**

4.1	Definitions	140
4.2	The enumeration of azulenoids	141
4.2.1	Translation to Delaney-Dress symbols	141
4.2.2	Restrictions on the Delaney-Dress symbol	143
4.2.3	Enumerating the octagon tilings	150
4.2.4	Inserting the azulene	151
4.3	Testing	155
4.4	Marked and unmarked tilings	156
4.5	Inserting the azulene in the chamber system	156
4.6	Visualising the results	164
4.6.1	Periodic graph	165
4.6.2	Embedding the tiling for a Delaney-Dress symbol	169
4.6.3	Periodic tilings, cylinders and tori	175
4.7	Results	177

5 Pseudo-convex patches **179**

5.1	Definitions	180
5.2	Constructing pseudo-convex patches	181
5.3	Extending the spiral	184
5.3.1	The algorithm	185
5.3.2	Overview of the cases	186
5.3.3	The non-degenerate case	187
5.3.4	A first class of degenerate cases	189
5.3.5	A second class of degenerate cases	194
5.4	Avoiding isomorphic copies	208
5.5	Splitting the patches into shells	209
5.6	Isolated pentagons	210
5.7	Implementation	210

6 Nanocones	211
6.1 Introduction	212
6.2 Cone Patches	213
6.3 Families of Cone Patches	215
6.3.1 Classification of nanocones	220
6.4 Algorithm	266
6.5 Testing	267
6.6 Results	267
6.7 Number of hexagons in cone patches	287
A Software	297
A.1 azul	298
A.2 cone	298
A.3 ddgraphs	299
A.3.1 Examples	300
A.4 PGVisualizer	302
A.5 pregraphs	303
A.5.1 Examples	303
A.6 PregraphViewer	305
B File formats	307
B.1 (Pre)graphs	308
B.1.1 Multi_code	308
B.1.2 Pregraph_code	308
B.1.3 Pregraphcolor_code	309
B.1.4 Embedded pregraphs	310
B.2 Planar graphs	310
B.2.1 Planar_code	310
B.3 Delaney-Dress symbols	311
B.4 Periodic graphs	312

B.5	Block lists	314
B.5.1	Block definitions	315
B.5.2	Example	317

Bibliography	319
---------------------	------------

Dutch summary/Nederlandse samenvatting	325
---	------------

List of Figures	331
------------------------	------------

List of Tables	339
-----------------------	------------

Index	345
--------------	------------

Acknowledgements

Although many people have contributed in some way to this work, only my name is written below its title. Therefore, I would like to take this opportunity to thank those people who, in some way, also contributed to this thesis.

First of all, I really have to thank my advisor, Gunnar Brinkmann. I do not think anybody could overestimate the amount of work that he has put into this thesis. He was always available to discuss any problem that popped up, and the pages of text that he commented on were countless. When I started my PhD, all directions were still open, and Gunnar was an excellent guide to help me find my own route through these different topics.

I would also like to thank the members of my jury. I understand that this thesis turned out to be a comprehensive document. Nevertheless, they managed to get through it and give suggestions for improvement by correcting small mistakes and asking to clarify some parts.

Next I would like to thank my parents. They are not only responsible for my existence, but also made it possible for me to start and finish my studies in mathematics. I have always been grateful, but this seems like an appropriate moment to put it down in black and white.

My girlfriend, Joyce, also deserves many thanks. Not only for supporting me throughout the years I was working on my PhD, but also for putting up with the neglect during the busier periods. She contributed much joy to my life, and in doing so, she gave me the motivation to keep going.

Through the years, I have had several co-authors and met several other researchers at conferences. This led to many interesting discussions and introduced me to many new topics. This was an enriching experience for me, and for this I owe them all my gratitude.

I would also like to thank the people from our research group 'Combinatorial Algorithms and Algorithmic Graph Theory', and I hope they find somebody else to take care of the bill once I am gone.

Currently, I have already been working for more than six year at the department 'Applied Mathematics and Computer Science'. I would like to thank all my colleagues (past and present) for being friends and offering me a hospitable working

environment.

Finally, I would like to thank all my friends. I am bound to forget someone, so I will not make any claims that the following enumeration is exhaustive.

I thank all the people at fencing club ‘DC Rheynaerde vzw’, ‘Oost-Vlaamse Schermclubs vzw’ and ‘Vlaamse Schermbond vzw’ for giving me the opportunity to spend my time fencing or having meetings about fencing.

I thank Tom and Valerie for many evenings spent drinking cava and playing board games. I hope that many more may follow.

I thank Bart and Griet for being good friends, although we both lack the time to see each other often. Now that this thesis is finished, I might find some more time to visit.

I thank Freija for a true friendship of many years and for always being available to talk about serious, or less serious matters. I hope our friendship, which started at the university, will last for many more years.

And just to make sure I do not forget anybody: I would like to thank everybody who, in some way, influenced my life. You all contributed to the person I am today. Thank you!

Short survey

By structure generation we mean designing, implementing (and running) an algorithm to construct objects from a certain class. Structure generation has many applications in theoretical chemistry and mathematics. In theoretical chemistry it forms an important tool to predict structures and test hypotheses. In mathematics it is used for the classification of structures and for the search for counterexamples to conjectures. We make a distinction between random generation algorithms and exhaustive generation algorithms. The first type of algorithm has as goal the generation of a random, equally distributed subset of all structures within a specific class. The second type has as goal the complete generation of a specific class. In this thesis we will only discuss exhaustive generation algorithms. More specifically, we will discuss only isomorphism-free, exhaustive generation algorithms. This means that we not only generate all structures from a specific class, but we also guarantee that only one structure per isomorphism class is generated.

In the **first chapter** we introduce concepts which are used throughout the thesis. The most important two concepts are of a graph and a tiling.

A graph $G(V, E)$ is a structure that consists of two sets. The elements of the first set V are called the vertices of the graph. The elements of the second set E are subsets of size 2 of V and are called the edges of the graph. The classic way to represent a graph is to draw the vertices as points and the edges as lines that connect these points. A graph is cubic if each vertex is incident to exactly 3 edges.

A tiling is a subdivision of the plane into tiles. We only consider tilings where each tile is finite and where every finite environment contains a finite number of tiles. The places where more than two tiles meet, are called the vertices of the tiling, and the places where exactly two tiles meet, are called the edges of the tiling.

If the symmetry group of a tiling contains two independent translations, then we call the tiling a periodic tiling.

An equivariant tiling is a pair (T, G) where T is a tiling and G is the symmetry group of that tiling.

In this chapter we also introduce Delaney-Dress symbols[25]. This structure consists of a cubic multigraph possibly with semi-edges (i.e., edges with only one vertex) for which the edges are coloured with three colours, and two functions that map

the vertices to natural numbers. These Delaney-Dress symbols encode equivariant, periodic tilings, so a Delaney-Dress symbol is a combinatoric, finite representation of an equivariant, periodic tiling. The graph in a Delaney-Dress symbol is called a Delaney-Dress graph.

The remainder of this thesis is subdivided into two parts. The subject of the first part is structure generation for mathematics.

In the **second chapter** we introduce a generation algorithm for generalised cubic graphs. Cubic graphs are a well-studied class of graphs and there exist several efficient programs for the generation of simple cubic graphs. We generalise the concept graph by also allowing multi-edges, semi-edges and loops as well as any combination of these. We group all these classes under the name pregraphs.

We break down the problem by first reducing it to the generation of cubic pregraph primitives. These are multigraphs with degrees 1 and 3. We describe a generation algorithm, based on McKay's 'canonical construction path'-method[43], which generates the cubic pregraph primitives starting from cubic simple graphs. Afterwards we apply the homomorphism principle[32] to generate the cubic pregraphs from the cubic pregraph primitives.

We conclude this chapter by describing some modifications to the generation algorithm which allow the generated structures to be limited to only those that are 3-edge-colourable or only those that are bipartite. Finally we also develop an algorithm which can determine in linear time whether a given cubic pregraph has a 2-factor such that each component is a quotient of C_4 . This class is interesting because these graphs are the underlying, uncoloured graphs for the Delaney-Dress graphs.

In the **third chapter** we return to the filter at the end of the previous chapter. We now use the analysis of the structure of cubic pregraphs with a 2-factor such that each component is a quotient of C_4 , to split these graphs into blocks. Then we describe a generation algorithm that uses these blocks to immediately generate graphs of this class. The gain in speed when compared to the filtering technique used in the previous chapter is of such a level that it now also becomes possible to generate Delaney-Dress graphs. Finally we take the last step and describe how we can generate Delaney-Dress symbols and thus equivariant tilings from these Delaney-Dress

graphs.

After the third chapter we start the second part of this thesis. The subject of the second part is structure generation for chemistry.

In the **fourth chapter** we describe a generation algorithm for a specific class of ‘graphite-like’ networks. Azulene ($C_{10}H_8$) is an isomer of naphthalene. It consists of a five-ring and a seven-ring that share two carbon atoms and a bond. This research was started following a question by Edward Kirby. He was interested in knowing which carbon networks were possible such that there exists a partition of the atoms into azulenes. These structures are called azulenoids. We model this using tilings and focus on those networks where there is only one orbit of azulenes under the symmetry group of the tiling. In this chapter we describe in detail the algorithm that was developed to generate all Delaney-Dress symbols that encode tilings that correspond to such azulenoids. Finally we also describe how we visualised the Delaney-Dress symbols in such a way that it was meaningful for chemists.

A 1,5-patch is a finite, bridgeless plane graph with three kinds of faces: one “outer face” with unrestricted size, 1 to 5 pentagons and an unrestricted number of hexagons. Furthermore all the vertices have degree 3 except some of the vertices of the outer face which have degree 2. If we denote the number of vertices of the outer face with degree two, respectively degree three, by b_2 , respectively b_3 , then the Euler Formula gives us that $b_2 - b_3 = 6 - p$ with p the number of pentagons in the patch. If the outer face contains no consecutive vertices of degree three, then we call the patch pseudo-convex. In the **fifth chapter** we describe an algorithm to generate all pseudo-convex patches with a given boundary sequence (i.e., a sequence of degrees for the vertices of the outer face). This algorithm is based on the encoding of pseudo-convex patches in [40]. Instead of directly constructing the pseudo-convex patches, this algorithm uses operations that act on the boundary sequence and an outer spiral (a sequence of faces as the pseudo-convex patch is unwounded spiral-wise starting from a special edge in the outer face). These have the benefit that they are both relatively short sequences of numbers, while the pseudo-convex patch can be a very large graph. This means that we have reduced the overhead during generation considerably.

In the **final chapter** we turn our attention to nanocones. A nanocone is a carbon network which is conceptually situated between graphite and the one-side infinite nanotubes: besides hexagons this structure contains between 1 and 5 pentagons, so that neither the flat shape of graphite nor the constant diameter tube of the nanotubes can be formed.

We view a nanocone as a disordering of a tiling; more specific a disordering of the hexagonal lattice such that all disorderings are faces which are pentagons. We use a result by Balke[39] which says that a disordering of an equivariant tiling is completely determined by the original equivariant tiling, a winding number and a conjugacy class of an automorphism in the symmetry group of the tiling. Using this result we show that there are 8 (infinite) classes of nanocones. Afterwards we subdivide each of the eight (infinite) classes in an infinite number of finite classes which also take the localization of the pentagons into account. Finally we describe how we can use the algorithm for the generation of pseudo-convex patches to generate a representation of each nanocone in such a class.

The research in this thesis is not reported in chronological order. The generation of azulenooids was the first thing that was done. After that we made the classification of the nanocones. The algorithm for the generation of pseudo-convex patches was designed while writing a program to generate the nanocones based on the classification. Then we worked on the generation algorithm for the cubic pregraphs. The generation algorithm for Delaney-Dress graphs is the most recent work in this thesis.

1

Definitions

A beginning is the time for taking the most delicate care that the balances are correct.

Frank Herbert, Dune

In this chapter we will first pack our bag with the necessary tools and get the general lay of the land of generation algorithms and (chemical) graph theory.

1.1 Graph theory

We will try to keep as closely as possible to the standard definitions in mathematics in general and in graph theory in particular. For the reader's convenience we start by giving a brief summary of the definitions and the terminology as we will use it. For a more in-depth overview see e.g., [51].

1.1.1 Graphs

In graph theory there exist several definitions of a graph, next to several extensions of the concept of a graph. Here we will only give one definition of a simple graph and give a definition of graphs with loops. Other definitions and generalizations will be introduced when they are needed.

(simple)
graph
vertices
edges
adjacent
incident
degree

Definition 1.1.1 A **(simple) graph** $G(V, E)$ is an ordered pair (V, E) of sets with V a countable set of arbitrary objects and E a set of 2-element subsets of V . The elements of V are called the **vertices** of G and the elements of E are called the **edges** of G .

Two vertices $x, y \in V$ are called **adjacent** if there exists an edge $e \in E$ such that $x \in e$ and $y \in e$. We will denote the edge $e = \{x, y\}$ by xy or yx .

A vertex $x \in V$ and an edge $e \in E$ are called **incident** if $x \in e$.

The number of edges incident to a given vertex x is called the **degree** of the vertex x . ◇

In this thesis we will mostly be using these graphs as models for certain objects such as e.g., molecules. We are interested in the structure of the molecules for which these graphs serve as models. As such it is not important for us which elements are in V , and we will often just represent them with the numbers 1 to $|V|$. Since the assignment of these numbers to the vertices is often arbitrary we can not really speak of equality, but we will use the concept of isomorphism.

isomor-
phism

Definition 1.1.2 Graphs $G = (V, E)$ and $G' = (V', E')$ are **isomorphic**, written

as $G \cong G'$, if there exists a bijection $\phi : V \rightarrow V'$ such that for all vertices $x, y \in V$ we have that x and y are adjacent in G if and only if $\phi(x)$ and $\phi(y)$ are adjacent in G' . \diamond

A special case of an isomorphism is the case where the bijection goes from V to V for the same graph $G(V, E)$. When this isomorphism is different from the identity it means that the graph G has a symmetry of some kind.

Definition 1.1.3 An isomorphism that maps a graph to itself is called an **automorphism**. The group of all automorphisms of a graph G is called the **automorphism group** of the graph G and is denoted as $Aut(G)$. \diamond

automorphism
 $Aut(G)$

In this thesis we will almost always mean ‘up to isomorphism’ when we say that a graph is unique or when we say that two graphs are equal.

There exist several extensions of the concept of a graph. We will mention a few here and some other extensions will be defined when needed.

Definition 1.1.4 A **graph with loops** $G(V, E)$ is an ordered pair (V, E) of sets with V a set of arbitrary objects and E a set of 2-element subsets of $V \cup \{L\}$ with $L \notin V$. The elements of V are called the vertices of G and the elements of E are called the edges of G . The edges that contain the element L are called **loops**. \diamond

graph with loops
loops

The definitions of adjacency and of incidence remain the same for graphs with loops. To calculate the degree the loops are counted as two edges.

Definition 1.1.5 A **multiset** \mathcal{M} is an ordered pair $\mathcal{M} = (M, m)$ with M a set and m a function $m : M \rightarrow \mathbb{N} \setminus \{0\}; e \mapsto m(e)$. For all $e \in M$ the value $m(e)$ is called the **multiplicity** of e and the multiset \mathcal{M} is said to contain $m(e)$ copies of e . \diamond

multiset
multiplicity
 \in

We have the following definition for the operator \in :

$$e \in M \Leftrightarrow e \in \mathcal{M}$$

\diamond

Definition 1.1.6 A **multigraph** $M(V, \mathcal{E})$ is an ordered pair (V, \mathcal{E}) of sets with V a set of arbitrary objects and $\mathcal{E} = (E, m)$ the multiset of edges. \diamond

multigraph
single edge
multi-edge
underlying graph

The elements of \mathcal{E} with multiplicity 1 are the **single edges** of M and the elements of \mathcal{E} with a larger multiplicity are the **multi-edges** of M .

The graph $G(V, E)$ is the **underlying graph** of the multigraph $M(V, \mathcal{E})$. \diamond

Two vertices are adjacent if they are adjacent in the underlying graph. An edge and a vertex are incident if they are incident in the underlying graph. To calculate the degree, each edge is counted with its multiplicity.

1.1.2 Subgraphs

subgraph **Definition 1.1.7** A graph $H(V', E')$ is called a **subgraph** of the graph $G(V, E)$ if we have that $V' \subseteq V$ and $E' \subseteq E$.

induced subgraph If a graph $H(V', E')$ is a subgraph of the graph $G(V, E)$ and E' contains all the edges $xy \in E$ with $x, y \in V'$, then $H(V', E')$ is called an **induced subgraph** or also the subgraph induced by the set V' in the graph $G(V, E)$. \diamond

Some special types of subgraphs are specifically important in graph theory.

path **Definition 1.1.8** A subgraph $P(V', E')$ of the graph $G(V, E)$ is a **path** if it has the form

$$V' = \{v_0, v_1, \dots, v_k\} \quad E' = \{v_0v_1, v_1v_2, \dots, v_{k-1}v_k\}$$

with $k \geq 0$ and v_0, \dots, v_k $k + 1$ distinct vertices. If $k = 0$, then the set E' is empty.

length The number of edges in a path is the **length** of that path. \diamond

Using paths we can define the concept of a connected graph.

connected **Definition 1.1.9** A graph $G(V, E)$ is **connected** if for each pair of vertices $x, y \in V$, there exists a path in G that starts in x and ends in y . \diamond

In this thesis we will work almost exclusively with connected graphs. Whenever we deal with disconnected graphs we will explicitly state this.

cycle **Definition 1.1.10** A subgraph $C(V', E')$ of the graph $G(V, E)$ is a **cycle** if it has the form

$$V' = \{v_0, v_1, \dots, v_k\} \quad E' = \{v_0v_1, v_1v_2, \dots, v_{k-1}v_k, v_kv_0\}$$

with $k \geq 2$ and v_0, \dots, v_k $k + 1$ distinct vertices.

length The number of edges in a cycle (or equivalently, the number of vertices) is the **length** of that cycle. \diamond

1.1.3 Bipartite graphs

Definition 1.1.11 A countable set \mathcal{P} is called a **partition** of the set S if the following three properties are satisfied **partition**

1. \mathcal{P} is a set of non-empty disjoint subsets of S ,
2. $\bigcup_{P \in \mathcal{P}} P = S$, and
3. for all $P_1, P_2 \in \mathcal{P} : P_1 \cap P_2 = \emptyset$.

In plain words a partition is a countable set of non-empty disjoint subsets of a set whose union is equal to that set.

The **size of a finite partition** \mathcal{P} is the number of elements in \mathcal{P} . The elements of \mathcal{P} are referred to as the parts of the partition. **size**

The **discrete partition** is the partition $\mathcal{P}_S = \{\{s\} | s \in S\}$.

◇ **discrete partition**

It follows immediately from this definition that the discrete partition exists only for countable sets.

Definition 1.1.12 For each integer $k \geq 2$ a graph $G = (V, E)$ is called **k -partite** if there exists a partition of V with size k such that there are no edges between vertices in the same part of the partition. **k -partite**

A 2-partite graph is called a **bipartite** graph.

◇ **bipartite**

The different parts of such a partition are often seen as different colour classes and a k -partite graph is also called **k -colourable**, because the vertices can be coloured with k colours in such a way that the endpoints of an edge never receive the same colour. The most interesting property of a graph with respect to vertex colourings is the minimum number of colours needed to colour the vertices of a graph. **k -colourable**

Definition 1.1.13 The **chromatic number** $\chi(G)$ of a graph $G = (V, E)$ is the smallest number k for which the graph G is k -partite. **chromatic number**

◇

A result that has often proven useful and powerful for characterising bipartite graphs is the following lemma.

Lemma 1.1.14 A graph is bipartite if and only if it contains no odd cycle.

See e.g., [51] for a proof of this lemma.

1.1.4 Edge colouring

Another way to colour a graph is to assign colours to the edges in such a way that all edges that contain a common vertex receive different colours.

**edge
colouring**

Definition 1.1.15 Given a non-empty set C , an **edge colouring** of a graph $G = (V, E)$ is a map $c : E \mapsto C$ with the property that two edges $e_1, e_2 \in E$ are mapped to different colours if they have a non-empty intersection.

**chromatic
index**

The **chromatic index** $\chi'(G)$ of a graph $G = (V, E)$ is the smallest number k for which there exists an edge colouring $c : E \mapsto C$ of $G = (V, E)$ with $|C| = k$. \diamond

For any graph this chromatic index is either Δ or $\Delta + 1$, with Δ the maximum degree of the graph. It is clear that an edge-colouring has at least $\Delta(G)$ colours, since all the edges that meet at a vertex with degree $\Delta(G)$ must receive different colours. That $\Delta + 1$ colours are sufficient for any graph is a famous theorem by Vizing [10].

Theorem 1.1.16 (Vizing, 1964) For each graph $G : \Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$.

See e.g., [51] for a proof of this theorem.

Another well known and useful result is the Parity Lemma [5]. We will here give the generalization given in [42].

Lemma 1.1.17 (Parity Lemma) Let $G = (V, E)$ be a graph with an edge-colouring $c : E \mapsto C$ and let a_k ($k \in C$) be the number of vertices v in V such that no edge incident to v is coloured k , i.e.

$$a_k = |\{v \in V \mid \forall e \in E : v \in e \Rightarrow c(e) \neq k\}|,$$

then for all $k \in C : a_k \equiv |V| \pmod{2}$.

Proof: Define for each $k \in C$ the set $E_k = \{e \in E \mid c(e) = k\}$. Then for each $k \in C$ we have $a_k = |V| - 2|E_k|$, which proves the lemma. \blacksquare

1.1.5 Plane graphs

Graphs that can be drawn in the plane without intersecting edges, form a class of graphs which has received specific attention, owing to the many applications of these graphs. Graphs with this property are called **planar** and such a drawing is called a plane graph.

planar

Definition 1.1.18 A **plane graph** is an ordered pair (V, E) of finite sets with the **plane graph** properties:

- $V \subseteq \mathbb{R}^2$;
- E is a set of subsets of \mathbb{R}^2 which are homeomorphic to the closed unit interval $[0, 1]$ such that the images of 0 and 1 are different and are both elements of V ;
- different edges have different sets of endpoints;
- the interior of an edge contains no vertex and no point of any other edge.

◇

Such a plane graph defines a simple graph G in a natural way, and we will usually use the name $G(V, E)$ to denote the plane graph (V, E) .

Definition 1.1.19 Given a plane graph (V, E) the **faces** of that graph are the **face** regions of the set $\mathbb{R}^2 \setminus (V \cup \bigcup_{e \in E} e)$. ◇

An important result for plane graphs is Euler's famous Formula [1]: it shows the connection between the vertices, the edges and the faces of a plane graph.

Theorem 1.1.20 (Euler's Formula) Let G be a connected finite plane graph with v vertices, e edges, and f faces, then

$$v - e + f = 2.$$

See [51] for a proof of this theorem.

1.2 Chemical graph theory

Graphs are purely mathematical objects, but they very often serve as models for real-life problems. An application where the use of graphs as models is very natural is in chemistry. In chemical graph theory a molecule is often represented by a graph such that the vertices correspond to the atoms in the molecule (or possibly to a significant subset of the atoms) and the edges correspond to the bonds (or possibly to multiple bonds). A research area where this representation has proven to be very fruitful is the field of fullerenes and related carbon molecules.

fullerene

A **fullerene** is a molecule consisting completely of carbon atoms in such a way that each atom shares a bond with exactly three other carbon atoms and that every carbon ring is either a pentagon or a hexagon. The most famous member of this family of molecules — and also the origin of the name for this family — is Buckminsterfullerene C_{60} . This molecule was discovered by Kroto et al. [21] in 1985. It was their discovery that this molecule has the shape of a truncated icosahedron that started a whole new field of research studying these ball-like molecules.

If we use the translation above and construct a graph F by letting the vertices of F correspond to the atoms and letting the edges of F connect vertices for which the atoms share a bond, then F will be a planar cubic graph. This motivates the following definition.

combinatorial fullerene

Definition 1.2.1 A **combinatorial fullerene** is a finite cubic plane graph with only pentagonal faces and hexagonal faces. \diamond

A (combinatorial) fullerene contains exactly twelve pentagonal faces. This follows from Euler's Formula. The proof is an easy exercise in using Euler's Formula for molecules of this kind.

Lemma 1.2.2 A (combinatorial) fullerene contains exactly twelve pentagonal faces.

Proof: Assume we have a fullerene with v vertices, e edges and f faces of which p are pentagonal faces and h are hexagonal faces. This means we have

$$f = p + h.$$

Since each vertex is cubic and thus is incident to three faces, we also have

$$v = \frac{5p + 6h}{3}.$$

Each edge is incident to exactly two faces, so we also have

$$e = \frac{5p + 6h}{2}.$$

If we substitute these three equalities into Euler's Formula we get

$$\frac{5p + 6h}{3} - \frac{5p + 6h}{2} + p + h = 2,$$

which is equivalent to

$$p = 12.$$



This mathematical representation of a chemical molecule allows us to use graph-theoretical results to look for properties of these molecules, or — the subject of this thesis — to apply combinatorial generation algorithms to find all possible structures that satisfy a certain property. One of the problems in the latter case is to translate the chemical restrictions into mathematical restrictions in order to bound the number of generated structures. There are e.g., 1812 combinatorial fullerenes with 60 vertices, but chemists are often only interested in one of those isomers, namely the one that corresponds to Buckminsterfullerene, because this is the most stable among these isomers. An important criterion for the stability of fullerenes is the so-called **Isolated Pentagon Rule** (IPR). This rule states that a fullerene is more stable if no two of its pentagonal faces share an edge.

**Isolated
Pentagon
Rule**

In this thesis we will look at two families of molecules that are related to fullerenes: the family of azulenooids (Chapter 4) and the family of nanocones (Chapter 6). The definitions of these families will be given at the time that this is needed. We will also look at a certain type of subgraphs in fullerenes, namely pseudo-convex patches (Chapter 5).

1.3 Tilings and Delaney-Dress symbols

1.3.1 Periodic tilings

A tiling is a subdivision of the plane into tiles. Tilings as we use them here, can be defined as follows.

Definition 1.3.1 *Let S be a partition of the Euclidean plane \mathbb{E}^2 and let $T = \{T_0, T_1, T_2\}$ be a partition of S , then T is a tiling of the Euclidean plane \mathbb{E}^2 if the following conditions are met:*

1. *the elements of T_0 are singletons;*

2. *the elements of T_1 are homeomorphic to the open interval $(0, 1)$;*
3. *the closure of an element of T_1 is homeomorphic to the closed interval $[0, 1]$ and the images of 0 and 1 are also elements of elements of T_0 ;*
4. *each element of T_0 has a non-empty intersection with the closure of at least three elements in T_1 ;*
5. *the elements of T_2 are homeomorphic to the open disk $\{z \in \mathbb{C} \mid |z| < 1\}$;*
6. *the closure of an element of T_2 is homeomorphic to the closed disk $\{z \in \mathbb{C} \mid |z| \leq 1\}$, the image of $\{z \in \mathbb{C} \mid |z| = 1\}$ is the union of finitely many elements of T_0 and T_1 (and, hence, the same number of elements of both [36]);*
7. *the closure of an element of T_2 contains at least three elements of T_1 ;*
8. $\forall x \in \mathbb{E}^2 : x$ *has a neighbourhood that intersects only a finite number of elements from T_0, T_1 and T_2 ;*

◇

vertices

The elements of the set T_0 are called the **vertices** of the tiling and we will mostly identify these singletons with their elements. The elements of the set T_1 are called the **edges** of the tiling. The number of edges that contain a vertex v in their closure is called the **degree** of the vertex v . The elements of T_2 are referred to as the tiles or the **faces** of the tiling. The number of vertices that the closure of a face f contains is called the **degree** of the face f . By definition the closure of an edge contains exactly two vertices. The vertices and edges in the closure of a face form a simple closed curve since they are homeomorphic to $\{z \in \mathbb{C} \mid |z| = 1\}$. It follows from the Jordan curve theorem [2] that for each edge e there are exactly two faces that contain e in their closure.

periodic

A tiling is called **periodic** if its symmetry group contains two independent translations. When a tiling is periodic, we only need a finite set of tiles to reproduce the complete tiling by repeatedly shifting and copying that finite set in as many directions as necessary [48].

skeleton graph

Definition 1.3.2 *The **skeleton graph** of a tiling $T = \{T_0, T_1, T_2\}$ of the Euclidean*

plane \mathbb{E}^2 is the (infinite) graph $G(T_0, E)$ with $xy \in E$ if x and y are different and there exists an edge $e \in T_1$ so that $\{x, y\} \subset \bar{e}$.

A cycle in the skeleton graph that corresponds to the edges and vertices around a face in the tiling is called a **facial cycle**. ◇ facial cycle

1.3.2 The flag space of a tiling

Given a tiling T as defined in Definition 1.3.1, we can now define the flag space of a tiling.

Definition 1.3.3 The **flag space** $\mathcal{F}(T)$ of T is the set of triples (v, e, f) consisting of one vertex v , one edge e and one face f such that $(v =) \bar{v} \subseteq \bar{e} \subseteq \bar{f}$ together with a Coxeter group Σ operating on this set: **flag space**

$$\Sigma = \langle \sigma_0, \sigma_1, \sigma_2 \mid \sigma_i^2 = \mathbf{1} \rangle,$$

that satisfies the following conditions:

1. $(v, e, f)\sigma_0 = (v', e, f)$ with $v \neq v'$;
2. $(v, e, f)\sigma_1 = (v, e', f)$ with $e \neq e'$;
3. $(v, e, f)\sigma_2 = (v, e, f')$ with $f \neq f'$.

◇

Since there are exactly two vertices that belong to the closure of an edge and since each edge is in the closure of exactly two faces, it follows that σ_0 and σ_2 are well defined involutions. That σ_1 is indeed a well defined involution follows from the fact that the vertices and edges around a face form a simple closed curve and in this curve a vertex can only be in the closure of exactly two edges.

Using the same properties, it is easily seen that for each vertex v of the tiling there are $2d$ flags containing v as their “0-dimensional component”, where d is the degree of the vertex; for each edge e of the tiling there are 4 flags containing e as their “1-dimensional component”; and for any n -gon f in the tiling, there are $2n$ flags containing f as their “2-dimensional component”.

By choosing a start flag and alternately applying the actions σ_1 and σ_2 , the flags containing a given vertex v of degree k form a circular sequence of $2k$ flags

$$(v, e_1, f_1), (v, e_2, f_1), (v, e_2, f_2), \dots, (v, e_k, f_k), (v, e_1, f_k).$$

This implies that the sequence above corresponds to two interwoven $\sigma_1\sigma_2$ -orbits of size k with opposite orientation, namely

$$(v, e_1, f_1), (v, e_2, f_2), \dots, (v, e_k, f_k);$$

and

$$(v, e_2, f_1), (v, e_3, f_2), \dots, (v, e_1, f_k).$$

By choosing a start flag and alternatingly applying the actions σ_0 and σ_2 , the flags containing a given edge e form a circular sequence of 4 flags

$$(v_1, e, f_1), (v_2, e, f_1), (v_2, e, f_2), (v_1, e, f_2).$$

This implies that the sequence above corresponds to two interwoven $\sigma_0\sigma_2$ -orbits of size 2 with opposite orientation, namely

$$(v_1, e, f_1), (v_2, e, f_2);$$

and

$$(v_2, e, f_1), (v_1, e, f_2).$$

By choosing a start flag and alternatingly applying the actions σ_0 and σ_1 , the flags containing a given face f of degree k form a circular sequence of $2k$ flags

$$(v_1, e_1, f), (v_2, e_1, f), (v_2, e_2, f), \dots, (v_k, e_k, f), (v_1, e_k, f).$$

This implies that the sequence above corresponds to two interwoven $\sigma_0\sigma_1$ -orbits of size k with opposite orientation, namely

$$(v_1, e_1, f), (v_2, e_2, f), \dots, (v_k, e_k, f);$$

and

$$(v_2, e_1, f), (v_3, e_2, f), \dots, (v_1, e_k, f).$$

Since the plane is connected, for any two flags $F, F' \in \mathcal{F}(T)$ there exists an element σ in Σ such that $F\sigma = F'$ which also implies that $F'\sigma^{-1} = F$.

1.3.3 The flag graph of a tiling

We may now combine all this information in a graph.

Definition 1.3.4 The **flag graph** $\Gamma(T) = (\mathcal{F}(T); \mathcal{E}_0 \cup \mathcal{E}_1 \cup \mathcal{E}_2)$ of T is the graph **flag graph** with vertex set $\mathcal{F}(T)$ and edges $\mathcal{E} = \mathcal{E}_0 \cup \mathcal{E}_1 \cup \mathcal{E}_2$, together with a partitioning of the edges into three sets $\mathcal{E}_0, \mathcal{E}_1$ and \mathcal{E}_2 . Two flags $F, F' \in \mathcal{F}(T)$ are connected by an edge in \mathcal{E}_i if and only if $F = F' \sigma_i$ ($i = 0, 1$ or 2). An edge of \mathcal{E}_i is called a σ_i edge. \diamond

Extending the term, we will also refer to the flag graph as a graph, although it is technically a pair consisting of a graph and a partition of the edges.

Definition 1.3.5 The σ_i, σ_j -**components** ($0 \leq i < j \leq 2$) of $\Gamma(T)$ are the con- σ_i, σ_j -
nected components of the graph $\Gamma_{ij}(T) := (\mathcal{F}(T); \mathcal{E}_i \cup \mathcal{E}_j)$. \diamond **components**

Since σ_i ($0 \leq i \leq 2$) is an involution, each vertex of the flag graph is incident to exactly one edge in \mathcal{E}_i . The flag graph is a connected graph, because for any two flags $F, F' \in \mathcal{F}(T)$ there exists $\sigma \in \Sigma$ that maps F to F' . For all $0 \leq i < j \leq 2$: the σ_i, σ_j -components correspond in a canonical one-to-one fashion to the subsets in T_k ($k \in \{0, 1, 2\} \setminus \{i, j\}$): a given σ_i, σ_j -component is always a cycle of $2m$ flags connected alternatingly by σ_i and σ_j edges, where m is equal to 2 for a σ_0, σ_2 -component and equal to the degree of the corresponding vertex or face for other components.

The flag graph of a tiling of the plane is an infinite graph.

1.3.4 Delaney-Dress symbols

The flag graph is already more convenient to work with than the flagspace, but it is still an infinite structure. When we have a periodic tiling, we can reconstruct the complete tiling using only a finite piece of it. We can now translate this property to the flag graph.

Definition 1.3.6 An **equivariant tiling** is a pair (T, G) such that $T = \{T_0, T_1, T_2\}$ **equivariant**
is a tiling and G is a subgroup of isometries of the plane that stabilize the sets T_0, T_1 **tiling**
and T_2 as sets. \diamond

Definition 1.3.7 Two tilings T and T' are **topologically equivalent** if there exists **topologi-**
a homeomorphism that maps tiles of T onto tiles of T' . \diamond **cally**
equivalent

In the definition above, the symmetry group of the tiling is not mentioned. We are interested in tilings together with their symmetry group and therefore we introduce a stronger form of equivalence.

**equivari-
antly
equivalent**

Definition 1.3.8 *Two tilings T and T' are **equivariantly equivalent** if there exists a homeomorphism that maps tiles of T onto tiles of T' and that maps (by conjugation) the symmetry group of T to the symmetry group of T' . \diamond*

Clearly, G acts also on the flag space $\mathcal{F}(T)$ and commutes with Σ , so it induces automorphisms of the flag graph $\Gamma(T)$. The group G acts fixed-point free on $\mathcal{F}(T)$, as $g(v, e, f) = (v, e, f)$ for some flag $(v, e, f) \in \mathcal{F}(T)$ implies that also $(v, e, f)\sigma_i$ ($i = 0, 1, 2$) must remain fixed under g , and then we can repeat this step for any of these flags and we find that the complete flag space must remain fixed; so g must be the identity.

$\mathcal{D}(T, G)$

We can now consider the orbit space $\mathcal{D}(T, G) := G \backslash \mathcal{F}(T)$ consisting of all G orbits of flags in $\mathcal{F}(T)$. As G commutes with Σ , $\mathcal{D}(T, G)$ forms the vertex set of a multigraph with loops

$$\Gamma(T, G) := (\mathcal{D}(T, G); G \backslash \mathcal{E}_0 \cup G \backslash \mathcal{E}_1 \cup G \backslash \mathcal{E}_2)$$

with three types of edges which we get by identifying any two σ_0 -, σ_1 -, or σ_2 -edges $\{F_1, F'_1\}$ and $\{F_2, F'_2\}$ from $\Gamma(T)$ if and only if there exists some $g \in G$ with $g(F_1) = F_2$ and, hence, $g(F'_1) = F'_2$. This also means that the edges of this graph correspond to the action induced on the orbit space $\mathcal{D}(T, G)$ by Σ in the same way that the edges of the flag graph correspond to the action of Σ . The induced actions are no longer fixed-point free and therefore this graph may contain loops.

**Delaney-
Dress
graph**

Definition 1.3.9 *The graph $\Gamma(T, G)$ is the **Delaney-Dress graph** of the equivariant tiling (T, G) . \diamond*

Any vertex of the Delaney-Dress graph is incident to exactly one edge of each $G \backslash \mathcal{E}_i$ ($i \in \{0, 1, 2\}$).

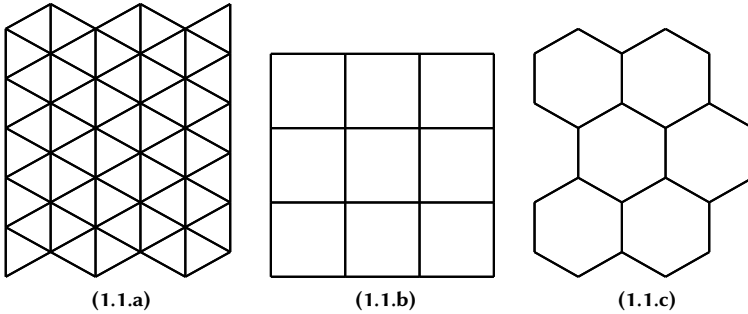


Figure 1.1: Three tilings with the same Delaney-Dress graph.

Example 1.3.10 Since for all three tilings in Figure 1.1 there is only one orbit of faces, one orbit of edges and one orbit of vertices, these three tilings all have the same Delaney-Dress graph with one vertex. Hence, the Delaney-Dress graph alone is not sufficient to distinguish between different tilings.

The only extra information that we need, and that is lost in the Delaney-Dress graph, is the size of the original $\sigma_i\sigma_j$ -orbits in the flag space. Therefore we define the following functions

$$\forall 0 \leq i < j \leq 2 : r_{ij} : \mathcal{F}(T) \rightarrow \mathbb{N}; F \mapsto \min\{l \in \mathbb{N}, l > 0 \mid F(\sigma_i\sigma_j)^l = F\}.$$

It follows from this definition that all the r_{ij} are constant on $\sigma_i\sigma_j$ -orbits and that $r_{02} = 2$ for all the flags in the flag space. For each flag F in the flag space we can define the following functions

$$\forall 0 \leq i < j \leq 2 : m_{ij} : \mathcal{D}(T, G) \rightarrow \mathbb{N}; G \cdot F \mapsto m_{ij}(G \cdot F) := r_{ij}(F).$$

Since the functions r_{ij} are constant on $\sigma_i\sigma_j$ -orbits, these functions m_{ij} are well-defined.

With these last functions we have all the tools at hand for the next definition.

Definition 1.3.11 The **Delaney-Dress symbol** of an equivariant tiling (T, G) is the triple $(\mathcal{D}(T, G), m_{01}, m_{12})$. **Delaney-Dress symbol**

These Delaney-Dress symbols are now sufficient to distinguish between equivariantly different tilings as is shown by the following theorem.

Theorem 1.3.12 (Dress [18]) *Two tilings are equivariantly equivalent if and only if their respective Delaney-Dress symbols are isomorphic.*

It can easily be verified for the tilings in Figure 1.1. Figure 1.1.a has $m_{01} = 3$ and $m_{12} = 6$, Figure 1.1.b has $m_{01} = m_{12} = 4$ and Figure 1.1.c has $m_{01} = 6$ and $m_{12} = 3$.

Hitherto we have described how to construct the Delaney-Dress symbol of a tiling of the plane. But more difficult and important will be this question: if we have a Delaney-Dress symbol, does there exist a tiling of the plane that corresponds to this symbol? First we need to state when we call a structure a Delaney-Dress symbol.

Definition 1.3.13 *A triplet $(\mathcal{D}; m_{01}, m_{12})$ is a Delaney-Dress symbol if and only if*

1. \mathcal{D} is a finite set;
2. Σ acts transitively on \mathcal{D} ;
3. m_{01} is constant on $\langle \sigma_0, \sigma_1 \rangle$ orbits and for each $d \in \mathcal{D} : d(\sigma_0 \sigma_1)^{m_{01}(d)} = d$;
4. m_{12} is constant on $\langle \sigma_1, \sigma_2 \rangle$ orbits and for each $d \in \mathcal{D} : d(\sigma_1 \sigma_2)^{m_{12}(d)} = d$;
5. $\forall d \in \mathcal{D} : d(\sigma_0 \sigma_2)^2 = d$;

◇

Definition 1.3.14 *The function $r_{ij} : \mathcal{D} \rightarrow \mathbb{N}$ with $0 \leq i < j \leq 2$ is defined such that $r_{ij}(d)$ is the smallest non-zero integer such that for each $d \in \mathcal{D} : d(\sigma_i \sigma_j)^{r_{ij}(d)} = d$. The function $v_{ij} : \mathcal{D} \rightarrow \mathbb{N}$ with $0 \leq i < j \leq 2$ is defined as follows: $v_{ij} \equiv \frac{m_{ij}}{r_{ij}}$. ◇*

curvature

Definition 1.3.15 *The **curvature** $K(\mathcal{D})$ of a Delaney-Dress symbol \mathcal{D} is the following invariant*

$$K(\mathcal{D}) = \sum_{d \in \mathcal{D}} \left(\frac{1}{m_{01}(d)} + \frac{1}{m_{12}(d)} - \frac{1}{2} \right).$$

◇

Theorem 1.3.16 *A Delaney-Dress symbol $(\mathcal{D}; m_{01}, m_{12})$ is the Delaney-Dress symbol of a periodic tiling*

- *of the hyperbolic plane if and only if $K(\mathcal{D}) < 0$;*
- *of the euclidean plane if and only if $K(\mathcal{D}) = 0$;*
- *of the sphere if and only if $K(\mathcal{D}) > 0$ and for all $0 \leq i < j \leq 2$ and for each $d \in \mathcal{D}$ we have that $\frac{4}{v_{ij}(d)K(\mathcal{D})} \in \mathbb{N}$.*

As is explained in [48], there is, unfortunately, no complete and easily accessible proof of Theorem 1.3.16 available from the literature. A complete, but rather involved proof is contained in [27]. A proof for the Euclidean case appears in [26].

Delaney-Dress symbols turned out to be useful and powerful tools for the enumeration of tilings (see [24],[36],[50]..., for example).

We introduced the Delaney-Dress graph of a tiling and illustrated that it is not possible to distinguish between all equivariant tilings using only their Delaney-Dress graphs. A Delaney-Dress symbol of an equivariant tiling contains basically the same structure as the Delaney-Dress graph of that tiling, but also has some additional information in the form of the functions m_{01} and m_{12} . The structural information in a Delaney-Dress symbol is not given as a graph, but as an action of the group Σ on the set \mathcal{D} . Sometimes, however, it can be useful to have a graph representation. That is why we will define the Delaney-Dress graph of a Delaney-Dress symbol.

Definition 1.3.17 *Given a Delaney-Dress symbol $(\mathcal{D}, m_{01}, m_{12})$, the Delaney-Dress graph $\Gamma(\mathcal{D})$ of $(\mathcal{D}, m_{01}, m_{12})$ is the graph with as vertices the elements of \mathcal{D} together with a partition of the edges into three sets E_0, E_1 and E_2 . Two vertices $d_1, d_2 \in \mathcal{D}$ are connected by an edge $e \in E_i$ if and only if $d_1 \sigma_i = d_2$. \diamond*

As was the case with the flag graph of a tiling, we will, by abuse of language, speak of the Delaney-Dress graph of a Delaney-Dress symbol as a graph. We will switch between the graph representation and the group representation of the structure of a Delaney-Dress symbol depending on which representation is most suited. We will therefore e.g., often refer to the elements of \mathcal{D} as the vertices of the Delaney-Dress symbol.

Note also that although we just use the name Delaney-Dress graph, this graph is actually a multigraph with loops.

 $\Gamma(\mathcal{D})_{ij}$
 $\sigma_i\sigma_j$ -
component

Definition 1.3.18 *The graph $\Gamma(\mathcal{D})_{ij}$ ($0 \leq i < j \leq 2$) is defined as $(\mathcal{D}; E_i \cup E_j)$. The set C_{ij} is the set of components of $\Gamma(\mathcal{D})_{ij}$. A component in C_{ij} is a $\sigma_i\sigma_j$ -**component** of $\Gamma(\mathcal{D})$. \diamond*

Since the functions m_{ij} are constant on σ_i, σ_j -components, we can reuse the same notation and define for each $0 \leq i < j \leq 2$ the following function:

$$m_{ij} : C_{ij} \rightarrow \mathbb{N}; C \mapsto m_{ij}(C) = m_{ij}(c) \text{ with } c \in C.$$

1.3.5 A geometric interpretation of flags

We defined a Delaney-Dress symbol of a tiling based on the flag space of that tiling. This led directly to combinatorial objects that are useful when generating Delaney-Dress symbols. There is however a more geometric interpretation of these flags, which we will give here because they provide a more intuitive way of thinking about Delaney-Dress symbols in relation to the tilings they encode.

barycentric
subdivision

Definition 1.3.19 *A **barycentric subdivision** B_T of a periodic tiling T is a tiling that is derived from T in the following manner:*

1. *Choose a point in the interior of each edge and each tile of T . Together with the vertices of T , these points will form the vertices of B_T .*
2. *For each tile, connect the point chosen in its interior with its vertices and all the points chosen on its edges by pairwise disjoint arcs.*

*The vertices of the barycentric subdivision are divided in three groups corresponding to the vertices, the edges and the faces of the original tiling T . Each tile has degree three and contains exactly one vertex from each group, and these three vertices correspond to incident structures in T , i.e., a tile in the barycentric subdivision can only contain a vertex corresponding to a vertex v of T , a vertex corresponding to an edge e of T and a vertex corresponding to a face f of T if v is contained in the closure of e , and if e is contained in the closure of f . The barycentric subdivision can always be constructed to have the same symmetry as the tiling [48]. We will only consider barycentric subdivisions that are constructed in that way. A tile of the barycentric subdivision is called a **chamber**. \diamond*

chamber

Like the flags in paragraph 1.3.2 these chambers contain one vertex corresponding to a vertex of T , one vertex corresponding to an edge of T and one vertex corresponding to a face of T . It is in fact so that we can see these chambers as a geometric representation of the flags. The equivalent of the flag space is then called the **chamber system**.

**chamber
system**

1.3.6 Examples

Before we continue with some basic properties of Delaney-Dress symbols we will first give some simple examples of Delaney-Dress symbols.

In Example 1.3.10 we showed that there are three tilings of the euclidean plane that correspond to the Delaney-Dress graph consisting of a single vertex. We will now take a closer look at some of those tilings, but this time in terms of the chamber system.

Example 1.3.20 *In Figure 1.2 the barycentric subdivision of the square tiling is shown. It is quite straightforward to verify that there is only one orbit of chambers under the symmetry group of the tiling. For each ordered pair of faces there always exists a translation mapping the faces onto each other. Together with the fact that the center of a face is the center of a rotation of order 4 and the dashed lines are mirror symmetries, this proves that each chamber can be mapped to each other chamber by a symmetry of the tiling.*

The following is an example of the influence of the symmetry group on the Delaney-Dress symbol. This will also be discussed in more detail in paragraph 1.3.7.1.

Example 1.3.21 *The rectangular tiling in Figure 1.3 is closely related to the tiling in the previous example. The only difference is that it has been stretched along the X axis, causing the dashed lines to no longer be mirror axes. This means that there are two orbits of chambers in this tiling: one orbit for which the chambers contain a vertex which corresponds to the short edge and one orbit for which the chambers contain a vertex corresponding to the long edge of the tiling. This illustrates how the Delaney-Dress graph codes the equivariant tiling, i.e., the tiling together with its symmetry group.*

Finally we will also show a somewhat larger example of a Delaney-Dress symbol.

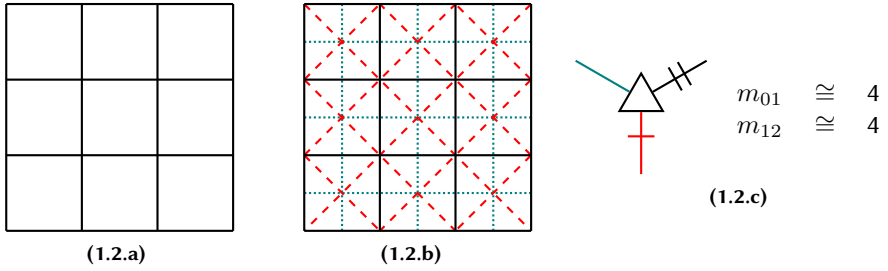


Figure 1.2: The square tiling (left), the barycentric subdivision of the square tiling (center) and the Delaney-Dress symbol of the square tiling (right).

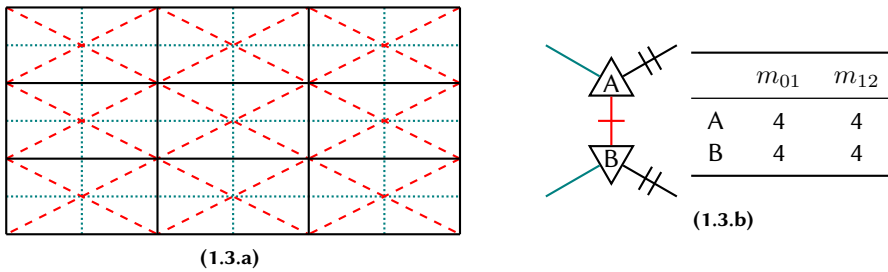


Figure 1.3: The barycentric subdivision of the rectangular tiling (left) and the Delaney-Dress symbol of the rectangular tiling (right).

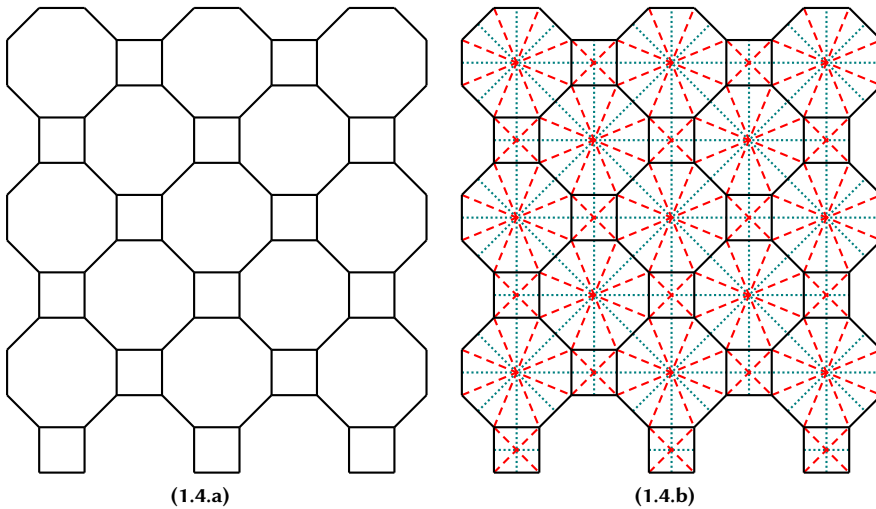


Figure 1.4: The bathroom tiling (left) and its barycentric subdivision (right).

Example 1.3.22 The tiling in Figure 1.4.a is called the bathroom tiling. It consists of regular octagons and squares. Given any two faces of the same size, there always exists a translation mapping the faces onto each other. Each centre of a face is a rotation centre of order 4, each line that connects the centre of two neighbouring faces is a mirror axis, and each line that extends the border between two neighbouring octagons is a mirror axis. In Figure 1.5 a small triangle containing three chambers is highlighted. Using the symmetries described above it is possible to cover the complete tiling with this triangle, so the Delaney-Dress graph will at most have order 3. That no two of these three chambers belong to the same orbit can be easily verified. One chamber lies in a square while the other two lie in an octagon, so they cannot be in the same orbit. The two chambers in the octagon are in different orbits because one chamber shares an edge with the square and the other does not. This gives us the Delaney-Dress symbol that is shown in Figure 1.6.

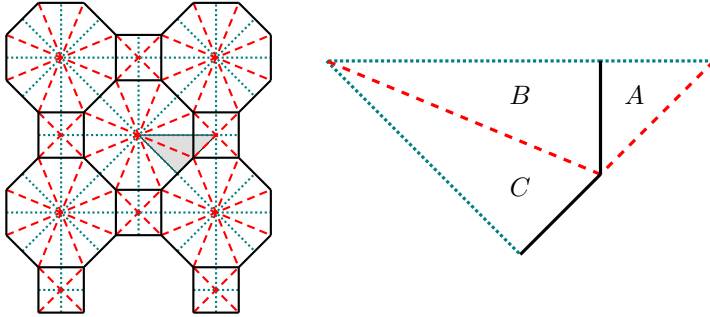


Figure 1.5: Representatives of the three orbits of chambers for the bathroom tiling.

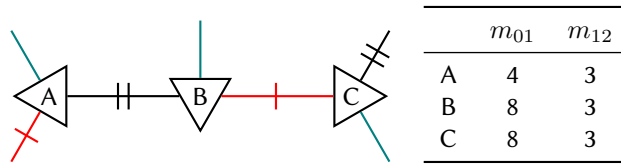


Figure 1.6: The Delaney-Dress symbol of the bathroom tiling from Figure 1.4.a

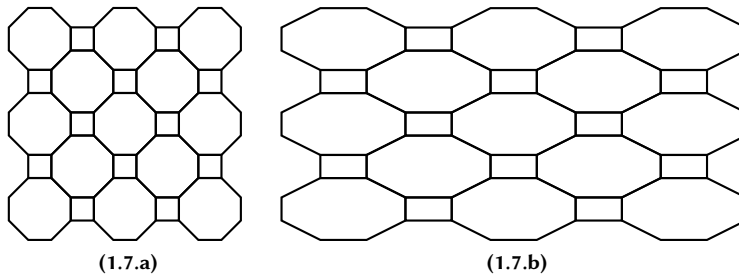


Figure 1.7: Two tilings that are topologically equivalent, but not equivariantly equivalent.

1.3.7 Basic algorithms on Delaney-Dress symbols

1.3.7.1 Minimal Delaney-Dress symbols

A Delaney-Dress symbol codes a tiling together with its symmetry group — that is an equivariant tiling. As we stated earlier this means that two tilings are equivariantly equivalent if their Delaney-Dress symbols are isomorphic [18].

The two tilings in Figure 1.7 are clearly not equivariantly equivalent because they have different symmetry groups: in Figure 1.7.a, a rotation of order 4 around a center of a face is a symmetry of the tiling, but in in Figure 1.7.b these symmetries are not present. Figure 1.7.b is however just Figure 1.7.a stretched along the horizontal axis. This means that the two tilings are topologically equivalent because there exists a homeomorphism that maps them onto each other. The tiling in Figure 1.7.a is topologically the same tiling as in Figure 1.7.b, but with more symmetry. The question now arises whether we can see this relation in the corresponding Delaney-Dress symbols. First, we will need some definitions.

Definition 1.3.23 An equivariant tiling (T', Γ') is a **symmetry breaking** of (T, Γ) if and only if there exists a homeomorphism ψ of the Euclidean plane $\mathbb{E}^2 : T = \psi(T')$ and $\psi\Gamma'\psi^{-1} \subsetneq \Gamma$. **symmetry breaking**
 \diamond

It follows from the definition that this is a transitive relation.

A special manner of constructing symmetry breakings is by introducing marked faces.

**marked
faces**

Definition 1.3.24 Let (T, Γ) be an equivariant tiling, let \mathcal{M} be a set of the tiles of (T, Γ) and let $\Gamma_{\mathcal{M}}$ be the subgroup of Γ that stabilizes \mathcal{M} as a set, then the equivariant tiling $(T, \Gamma_{\mathcal{M}})$ is said to be constructed from (T, Γ) by **marking the faces** in \mathcal{M} . \diamond

In case $\Gamma_{\mathcal{M}}$ is a true subgroup of Γ this is a symmetry breaking, otherwise the two tilings are equivariant.

morphism

Definition 1.3.25 A function $f : (\mathcal{D}; m_{01}, m_{12}) \rightarrow (\mathcal{D}'; m'_{01}, m'_{12})$ between two Delaney-Dress symbols $(\mathcal{D}; m_{01}, m_{12})$ and $(\mathcal{D}'; m'_{01}, m'_{12})$ is a **morphism** if and only if for each flag $c \in \mathcal{D}$:

- $\forall i \in \{0, 1, 2\} : f(\sigma_i(c)) = \sigma_i(f(c))$
- $m'_{01}(f(c)) = m_{01}(c)$
- $m'_{12}(f(c)) = m_{12}(c)$

\diamond

It has been shown in [25] that if there exists a morphism from the Delaney-Dress symbol of an equivariant tiling (T', Γ') into the symbol of an equivariant tiling (T, Γ) and the symbol of the second tiling contains less elements, then (T', Γ') is a symmetry breaking of (T, Γ) .

minimal

Definition 1.3.26 A Delaney-Dress symbol is called **minimal** if there does not exist a morphism from this symbol to a smaller symbol. \diamond

Note 1.3.27 The name *minimal* is chosen because this symbol has minimal size. The corresponding equivariant tiling has maximal symmetry, which is why some texts use the name *maximal* symbol.

In [48] it is proven that each tiling (T, Γ) has a unique minimal Delaney-Dress symbol. Finding the minimal Delaney-Dress symbol of a given tiling (T, Γ) , i.e., finding the Delaney-Dress symbol of the tiling with maximal symmetry that is isomorphic to (T, Γ) , comes down to finding the smallest Delaney-Dress symbol for which there

exists a morphism for the original symbol into that symbol. This also comes down to finding equivalence classes of flags in the original Delaney-Dress symbol.

An algorithm to construct the minimal Delaney-Dress symbol of a tiling is given in [48]. This algorithm constructs equivalence classes by recursively trying to form equivalence classes of flags. Afterwards we get the minimal Delaney-Dress symbol by taking the equivalence classes as the vertices in the new Delaney-Dress graph. For a tiling with a Delaney-Dress graph with n flags, the algorithm has a total running time of $O(n^2 \cdot a(n))$ with a the inverse Ackermann function.

For more details on the implementation of this algorithm we refer to [48].

1.3.7.2 Fundamental domain and cover

When we want to reconstruct the tiling that corresponds to a Delaney-Dress symbol, we will need the fundamental patch of the Delaney-Dress symbol. In later chapters we will also be interested in the following problem: Given a Delaney-Dress symbol $(\mathcal{D}; m_{01}, m_{12})$ that corresponds to the equivariant tiling (T, Γ) , find a Delaney-Dress symbol $(\mathcal{D}'; m'_{01}, m'_{12})$ that corresponds to a tiling (T', Γ') , such that (T, Γ) is a symmetry-breaking of (T', Γ') and Γ' only contains translations. This Delaney-Dress symbol $(\mathcal{D}'; m'_{01}, m'_{12})$ is then called a translation-only cover of the original Delaney-Dress symbol.

Definition 1.3.28 *If $f : (\mathcal{D}; m_{01}, m_{12}) \rightarrow (\mathcal{D}'; m'_{01}, m'_{12})$ is a morphism, then the set of flags that is mapped to a $\sigma_i \sigma_j$ -component C in $(\mathcal{D}'; m'_{01}, m'_{12})$ is called the **cover** of C .*

*Furthermore we also call the symbol $(\mathcal{D}; m_{01}, m_{12})$ a **cover** of the symbol $(\mathcal{D}'; m'_{01}, m'_{12})$.* \diamond

Definition 1.3.29 *A Delaney-Dress graph is **orientable** if it is bipartite. A Delaney-Dress symbol is **orientable** if its Delaney-Dress graph is orientable.* \diamond

Definition 1.3.30 *Given an equivariant tiling (T, Γ) with Delaney-Dress symbol $(\mathcal{D}; m_{01}, m_{12})$ a $\sigma_i \sigma_j$ -component C in \mathcal{D} ($0 \leq i < j \leq 2$) is said to be **branched** if $r_{ij}(c) < m_{ij}(c)$ with $c \in C$.* **branched component**

*The **branching size** of a branched $\sigma_i \sigma_j$ -component C is equal to $v_{ij}(c)$ with $c \in C$.* **branching size** \diamond

extracted **Definition 1.3.31** Given an equivariant tiling (T, Γ) with Delaney-Dress symbol $(\mathcal{D}; m_{01}, m_{12})$ a vertex, edge or face is said to be **extracted** if the corresponding $\sigma_i \sigma_j$ -component C in \mathcal{D} ($0 \leq i < j \leq 2$) is unbranched. \diamond

fundamental patch **Definition 1.3.32** A **fundamental patch** of an orientable Delaney-Dress symbol $(\mathcal{D}, m_{01}, m_{12})$ is a connected, maximal subgraph (\mathcal{D}, E') of $\Gamma(\mathcal{D})$ such that for each branched component C , there exists an edge e such that $e \notin E'$. \diamond

By maximal in the definition above, we mean that no additional edge of the Delaney-Dress graph can be added to the subgraph without breaking the property that for each branched component C , there exists an edge e such that $e \notin E'$.

A fundamental patch of a Delaney-Dress symbol can be obtained by first taking a spanning tree of the Delaney-Dress graph and then closing $\sigma_i \sigma_j$ -components which are unbranched. Note that closing is the correct term here, because we are dealing with an orientable Delaney-Dress symbol and thus each $\sigma_i \sigma_j$ -component is a cycle.

When a Delaney-Dress symbol is an orientable symbol without any branched components, the symbol's equivariant tiling contains only translational symmetries. If the symbol however contains branched components then we will have to construct a cover. Below we give a short description of how this translation-only cover of a Delaney-Dress symbol can be obtained[65].

To create the cover we start by checking to see if the symbol is orientable. When the symbol of an equivariant tiling is orientable, the symmetry group of the equivariant tiling does not contain any reflections or glide reflections. To get the orientable cover of a non-orientable symbol, we first construct a spanning tree of the symbol. This spanning tree is bipartite — it does not contain any cycles, so certainly no odd cycles — and we can colour it with two colours. Then we make two copies of the original symbol, one copy has the same colours as the spanning tree, the other has opposite colours. When an edge connects two chambers c_1 and c_2 with the same colour, we remove the edge between c_1 and c_2 in both copies and connect c_1 of the first copy to c_2 of the second copy, and c_1 of the second copy with c_2 of the second copy. We define functions m_{01}^o and m_{12}^o such that the values for a chamber in any of the two copies is the same as the values for the functions m_{01} and m_{12} for the original chamber. This results in a symbol that is twice as big and orientable: since the

original symbol was not orientable there is at least one pair of edges that connect the spanning tree of the first symbol to the spanning tree of the second symbol, so this new symbol is certainly connected and by construction this new symbol is bipartite and thus orientable. The tiling corresponding to this larger Delaney-Dress symbol is indeed a symmetry-breaking of the tiling corresponding to the original symbol, because the function that maps the chambers in the copies to the same chamber in the original symbol is, by construction of the larger symbol, a morphism.

Lemma 1.3.33 *The equivariant tiling corresponding to the oriented cover $(\mathcal{D}^\circ, m_{01}^\circ, m_{12}^\circ)$ of a not-orientable Delaney-Dress symbol $(\mathcal{D}, m_{01}, m_{12})$ is a symmetry-breaking of the equivariant tiling corresponding to $(\mathcal{D}, m_{01}, m_{12})$.*

Proof: The oriented cover consists of two copies of the original symbol. For each chamber $v \in \mathcal{D}$, there are two corresponding vertices in \mathcal{D}° . We will denote these chambers by v_1 and v_2 . We have to prove that the function f which maps for each chamber $v \in \mathcal{D}$ both $v_1 \in \mathcal{D}^\circ$ and $v_2 \in \mathcal{D}^\circ$ to v , is a morphism.

By construction of the oriented cover for each chamber $c \in \mathcal{D}^\circ$ the function f satisfies the property that $m_{01}(f(c)) = m_{01}^\circ(c)$ and $m_{12}(f(c)) = m_{12}^\circ(c)$.

What remains to be checked is that for each $i \in \{0, 1, 2\}$ and for each chamber $c \in \mathcal{D}^\circ$ we have that $f(\sigma_i(c)) = \sigma_i(f(c))$.

Suppose we have two chambers $c, d \in \mathcal{D}$ such that $\sigma_i(c) = d$ for some $i \in \{0, 1, 2\}$. The chamber c , respectively d , corresponds to two chambers $c_1, c_2 \in \mathcal{D}^\circ$, respectively $d_1, d_2 \in \mathcal{D}^\circ$. During the construction of the oriented cover there are two possibilities: either c and d received the same colour, or they received different colours.

Consider first the case that they received the same colour. In this case we have the following situation in the oriented cover

$$\begin{aligned}\sigma_i(c_1) &= d_2, \\ \sigma_i(c_2) &= d_1.\end{aligned}$$

This means we have

$$f(\sigma_i(c_1)) = f(d_2) = d = \sigma_i(c) = \sigma_i(f(c_1)),$$

and similar reasoning can be constructed for c_2 .

The case that c and d receive different colours is also quite similar. In the oriented cover we have the following situation

$$\sigma_i(c_1) = d_1,$$

$$\sigma_i(c_2) = d_2.$$

This means e.g., that we have for c_1

$$f(\sigma_i(c_1)) = f(d_1) = d = \sigma_i(c) = \sigma_i(f(c_1)).$$

■

If the fundamental patch of the oriented symbol is isomorphic to the oriented symbol, we are done. Otherwise we need to create a larger cover to remove other symmetries. We start by calculating the branching size k of the largest branched $\sigma_i\sigma_j$ cycle in the symbol. Then we create a k -fold cover of the oriented symbol. For this we make k copies of the fundamental patch and we denote the k copies of a flag $d \in \mathcal{D}$ by d^i with $1 \leq i \leq k$. Next we try to add the missing edges in the fundamental patch in all the possible ways, so that we have a symbol without any missing edges or branched cycles. This means that when a flag d_1 is connected to a flag d_2 by a σ_j -edge that is missing in the fundamental patch, we try to connect d_1^1 to any d_2^i that is still available, and then the same for d_1^2, \dots, d_1^k . When there is a contradiction we backtrack.

There is a relation between fundamental patches and tile-transitive tilings. A fundamental patch corresponds to the smallest region of a tiling we need to reproduce the complete tiling. If we realize the fundamental patch of a tiling and then copy this realization and paste them together based on the symmetries at the edge of the

fundamental patch we get a realization of the complete tiling. Thus we can view the fundamental patch as a sort of ‘super-tile’ which is part of a tiling with only one kind of tiles, i.e., a **tile-transitive tiling**.

**tile-
transitive
tiling**

There are only two possible types of fundamental patches for a Delaney-Dress symbol that only contains translational symmetries, i.e., a quadrangle and a hexagon. This follows from the fact that the only two possibilities for tile-transitive tilings with only translational symmetries are the square grid with the square extracted and the honeycomb with the hexagon extracted. That these are the only two possibilities, can easily be proven with the use of Delaney-Dress symbols.

Theorem 1.3.34 *The only two tile-transitive, equivariant tilings containing only translation symmetries are the square grid with the square extracted and the honeycomb with the hexagon extracted.*

Proof: Since the tiling is tile-transitive, its Delaney-Dress symbol can only have one $\sigma_0\sigma_1$ orbit (and thus the function m_{01} is constant for the complete symbol and we use the same notation for the value of this constant) and because the tiling may only contain translation symmetry, this orbit will contain exactly $2m_{01}$ chambers.

So we can fill in this information into the formula for the curvature of the tiling:

$$\sum_{d \in \mathcal{D}} \left(\frac{1}{m_{01}(d)} + \frac{1}{m_{12}(d)} - \frac{1}{2} \right) = \frac{2m_{01}}{m_{01}} + \left(\sum_{d \in \mathcal{D}} \frac{1}{m_{12}(d)} \right) - \frac{2m_{01}}{2} = 0.$$

This means that

$$\sum_{d \in \mathcal{D}} \frac{1}{m_{12}(d)} = m_{01} - 2.$$

Suppose the number of $\sigma_1\sigma_2$ orbits is k and denote the constant value of the function m_{12} on the i th orbit by m_{12}^i . Since the tiling may only contain translation symmetry, the i th orbit will contain $2m_{12}^i$ chambers. Substituting this information in the equation above, we find

$$m_{01} - 2 = \frac{2m_{12}^1}{m_{12}^1} + \cdots + \frac{2m_{12}^k}{m_{12}^k} = 2k,$$

or

$$m_{01} = 2k + 2.$$

The $\sigma_1\sigma_2$ orbits form a partition of the Delaney-Dress graph and thus we also have the following equality:

$$m_{01} = m_{12}^1 + \cdots + m_{12}^k.$$

At least 3 faces (or edges) meet in a vertex. This means that each of the m_{12}^i is at least 3. So we can rewrite the equality above as

$$m_{01} = (3 + l_1) + \cdots + (3 + l_k) = 3k + l,$$

with $l_i, l \in \mathbb{N}$. Combining the equalities we find that

$$2k + 2 = 3k + l,$$

or,

$$-k + 2 = l,$$

with $l \in \mathbb{N}$ and $k \in \mathbb{N}^*$. This equation has only two solutions for (k, l) and those are $(1, 1)$ and $(2, 0)$.

The first solution $(1, 1)$ corresponds to a quadrangular tile with exactly 1 $\sigma_1\sigma_2$ -component and for this component the value of m_{12} is equal to 4. The $\sigma_0\sigma_1$ -component for this symbol is shown in Figure 1.8.a. The symbol must be orientable, hence the graph has to be bipartite. The $\sigma_0\sigma_2$ -components form 4-cycles. This means that in a valid symbol the oppositely coloured endpoints of two σ_0 edges are connected. If the edge A would be connected to the edge B or D , there would be a branched $\sigma_1\sigma_2$ -component. Therefore the only possibility is that A is connected to C and B is connected to D . The resulting symbol can be seen in Figure 1.9.a. This is the Delaney-Dress symbol of the square grid with the square extracted.

The second solution $(2, 0)$ corresponds to a hexagonal tile with exactly 2 $\sigma_1\sigma_2$ -components and for both the value of m_{12} is 3. The $\sigma_0\sigma_1$ -component for this

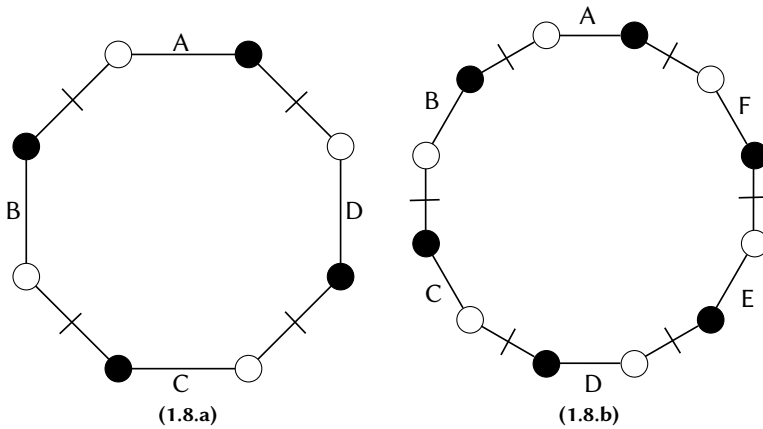


Figure 1.8: The two possible $\sigma_0\sigma_1$ orbits corresponding to a quadrangle(a) and a hexagon (b).

symbol is shown in Figure 1.8.b. Once again the symbol must be orientable, and thus the graph has to be bipartite. If A is connected to B or F , there would be a branched $\sigma_1\sigma_2$ -component. If A is connected to C , then D must be connected to F and B to E , but this means that there would be two $\sigma_1\sigma_2$ -components with impossible sizes. If A is connected to E a similar reasoning can be made. Thus the only remaining possibility is that A is connected to D , B to E and C to F . The resulting symbol can be seen in Figure 1.9.b. This is the Delaney-Dress symbol of the honeycomb with the hexagon extracted.

■

1.4 Generation algorithms

The goal of this section is to offer a high-level view of the different techniques that are used in structure generation algorithms. For several of these methods, details will be explained later on in a more concrete setting when they are applied to specific

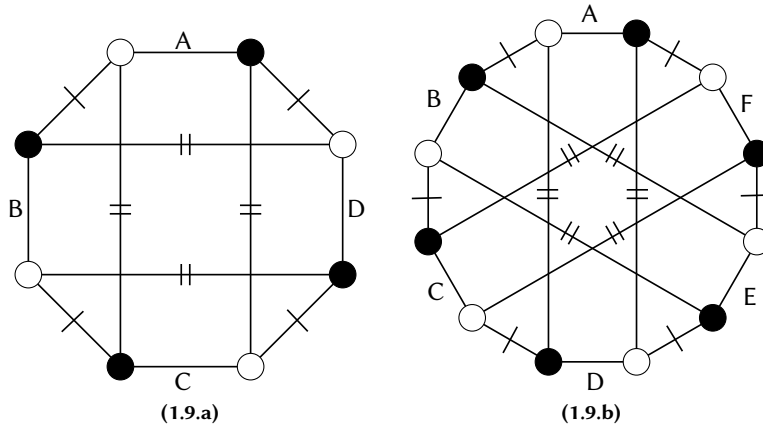


Figure 1.9: *The Delaney-Dress symbols of the only two tile-transitive tilings containing only translation symmetry.*

situations in different chapters. An introductory overview of these techniques can also be found in [44].

1.4.1 Construction operations

search space

construction operation

First the set of structures of interest must be defined. This is usually the set of all graphs that share a common property, e.g., all graphs, all cubic graphs, ... This set is called the **search space**. In case of an infinite search space it is often subdivided by an extra parameter, e.g., all graphs with n vertices, ... At the core almost all generation algorithms look basically the same. They define a set of operations, the **construction operations**, which, when iteratively applied to a set of initial structures, transform a structure in the search space into another structure in the search space or, through intermediary structures, finally produce a structure in the search space.

In this thesis we will be focusing on exhaustive generation algorithms, i.e., we want to be sure that we generate all structures in the search space. For this we must prove that, given the correct set of initial structures, each structure in the search space can be reached using the construction operations.

The common way to do this is by looking at the inverse operations of the construction operations. We will call these inverse operations **reduction operations**. The problem of proving that each structure can be reached using the construction operations is equivalent to proving that each structure in the search space and — if applicable — each intermediate structure, except for some (small) set of irreducible structures, can be reduced by one of the reduction operations. Usually an invariant or a combination of invariants that monotonically increases or decreases is then used to prove that this reduction process always stops, i.e., that it does not get stuck in an infinite loop that keeps cycling through a subset of the search space. Once this is proven we have proven that the set of irreducible graphs is sufficient to start the generation process and reach the entire search space.

**reduction
operation**

1.4.2 Isomorphism rejection

We are not interested in exhaustive generation algorithms alone: this thesis is in fact about isomorph-free exhaustive generation algorithms. The problem with the construction operations as described above is that there is hardly ever a way for these operations to guarantee that no two of the generated structures are isomorphic. There exist several techniques to add isomorphism rejection to the generation algorithm and this is the point on which generation algorithms mostly differ.

To filter out isomorphic copies we need a way to recognise isomorphic copies. Very often we even want to be able to decide whether an isomorphic copy exists without comparing the graphs. A key concept for this is **canonicity**. Canonicity is derived from the Greek word κανών (kanon) which means *rule* or *measuring stick*. This term is often used in mathematics to identify a unique and natural way to represent something. In the context of generation algorithms it will mostly be used to refer to a **canonical vertex labelling** or numbering. This is a fairly standard concept which we will now first define here before we will use it in the following sections to explain the different isomorphism rejection techniques.

canonicity

**canonical
vertex
labelling**

Let \mathcal{G} denote the set of all labelled graphs — or in some cases, the set of all labelled graphs satisfying certain conditions. A **canonical representative function** is a function $c : \mathcal{G} \rightarrow \mathcal{G}$ that assigns to each labelled graph another labelled graph

**canonical
representa-
tive
function**

called its canonical form. There are many different ways a specific canonical form can be defined, but a canonical form $c(G)$ of a graph G must satisfy the following two properties:

1. $\forall G \in \mathcal{G} : c(G) \cong G$
2. $\forall G, G' \in \mathcal{G} : G \cong G' \Rightarrow c(G) = c(G')$

**canonical
representa-
tive**

From this definition it follows also that $\forall G \in \mathcal{G} : c(c(G)) = c(G)$, so c is idempotent. In every isomorphism class c fixes exactly one element and this element is called the **canonical representative** of the class. An isomorphism $\phi : G \rightarrow c(G)$ is called a canonical labelling of the vertices of G . The canonical representative of a graph G is unique, but the canonical labellings for a graph G are unique only up to automorphisms of G .

There are many ways to choose the canonical representative for an isomorphism class. Although the choice should depend on the problem at hand to choose a form that is ‘easy’ to calculate, there are some techniques which are easy, common and often powerful enough for most cases. An example of a canonical form of a graph is the isomorphic labelled graph for which the binary number formed by concatenating the rows of the adjacency matrix is largest as the canonical representative.

1.4.3 Isomorphism rejection by lists

A very straight-forward and simple way to make sure that the algorithm does not output isomorphic structures is by keeping a list of output structures in memory or on file and, each time a new structure is generated, comparing it to the list. This method is called isomorphism rejection by lists.

It is possible to use isomorphism rejection by lists without using the canonical representatives, but they could decrease the necessary amount of calculations significantly. If instead of the generated structure a canonical representative is stored in the list and before a generated graph is compared to the list, its canonical representative is calculated, then comparing the two graphs becomes simpler because we are looking for equality and not just isomorphic equivalence. Moreover, the canonical

representative could also be stored in a sorted list which would mean that not every graph must be examined, and so the amount of work needed to verify whether the current graph is new is further decreased.

This method is easy to implement and usually not especially prone to error, but is unfortunately also only possible when the search space is really small. In Chapter 4 this technique is used to generate a class of graphs that contains 1274 graphs. In the other chapters we generate classes of structures that are infinite except when restricted to a certain number of vertices. In those cases isomorphism rejection by lists cannot be used. For this reason several techniques have been developed that can decide whether a structure should be output or not, without knowing the other structures.

1.4.4 Canonical representatives

A first method to solve this problem is by generating canonical representatives. This technique can be split in four steps.

1. First a code should be defined that is unique for every labelled structure that will arise during the generation process,
2. then for each structure one of the codes among all isomorphic labelled structures should be chosen as the canonical code,
3. next generate all structures and make sure that for every isomorphism class exactly one structure is generated that has the canonical code, and
4. finally accept a generated structure if the associated code is canonical and reject it otherwise.

It is no problem if not all labelled structures are generated, but it is very important that in step 3 at least all the labelled structures which lead to the canonical codes are generated. If for a certain structure the labelled version that leads to the canonical code is not generated, then all labelled versions of this structure will be rejected as there is no way to know that this structure is missing from the generated structures and thus the generation process would not be exhaustive.

The advantage of this method, as announced above, is that it can decide for each generated structure whether it should be rejected or output. This means also that this technique allows the generation process to be split into parts and divide over several computers or runs of the same program. It is however possible that each structure will be generated several times and will be rejected in all but one case. If this happens too often, it means that a lot of work goes into generating structures that will eventually be rejected. It would be advantageous to decide much earlier in the generation process whether a structure will lead to a canonical representative or not.

1.4.5 Read/Faradžev-type orderly algorithms

In the Read/Faradžev-type orderly algorithms an early bounding criterion with respect to the canonicity criterion is used. The construction operations have to be compatible with the canonical code in such a way that it is possible at an early stage to decide whether a partial structure will lead to a canonical representative or not. In case a certain partial structure can no longer lead to a canonical representative, that branch of the search space is abandoned and the algorithm can backtrack.

An obvious disadvantage is that the code and the construction operations have to be compatible to make sure that such an early bounding criterion is possible. This might lead to a canonical form which is difficult to calculate. In some cases this early bounding criterion might not even be possible that ‘early’ in the generation process and so this method might not provide the desired increase in efficiency.

This technique is used in the algorithm in Chapter 5 which is later used in Chapter 6.

1.4.6 McKay’s canonical construction path method

A technique which improves on these vulnerabilities of Read/Faradžev-type orderly algorithms is McKay’s canonical construction path method. At its core, this method is basically the same as the two previous algorithms, but it uses a general and powerful approach which grants it the right to be mentioned separately. The power comes from the fact that the effort to avoid the construction of isomorphic

copies is made in each construction step instead of waiting until the partial structures reach a point where the possible codes can be compared.

This method can be applied to each generation algorithm where recursive construction steps are used, i.e., ‘smaller’ structures are used to generate ‘larger’ structures. This method then assures that at each level only non-isomorphic structures are used and thus fixes for each generated structure a unique path from one of the initial irreducible structure to the structure itself. Hence the name of this method.

This method can also be synthesised into three simple steps.

1. First for each structure a unique reduction (i.e., a unique construction step and a unique ‘smaller’ structure) must be defined;
2. during the generation process a structure is accepted only if it is constructed by the inverse of the unique reduction (which we call the canonical operation), and
3. for each ancestor it must be checked that only ‘non-isomorphic’ construction steps are performed.

Step 3 normally requires the calculation of the automorphism group of the ancestor structure, but these ancestor structures are usually fewer and ‘smaller’ than the target structures, so this is mostly not the problem. There are also techniques to improve on this by using information from previous steps to increase the efficiency of the calculation of the automorphism group or even render it redundant in some cases.

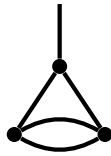
This powerful method is explained in [43]. We use this approach in Chapter 2 and Chapter 3.

Part I

Generation of mathematical structures

2

Pregraphs



This research was joint work with Tomáš Pisanski and is described in [59]. The main goal was to solve several generation problems for cubic graphs and related classes. For examples of uses for the different classes we refer to [59]. Here we will just discuss the generation of these classes.

2.1 Definitions

Pregraphs are graphs that also may contain loops, multi-edges and semi-edges (dangling edges). Loops and multi-edges have already been defined, but semi-edges introduce a new concept at this point.

pregraph

Definition 2.1.1 A **pregraph** P is an ordered pair $P = (V, \mathcal{E})$ with V the set of vertices and \mathcal{E} the multiset (E, m) of edges with $E \subseteq \binom{V}{2} \cup (V \times \{L\}) \cup (V \times \{S\})$. We have the following naming of the edges:

- $\forall \{v, w\} \in E : m(\{v, w\}) = 1 \Rightarrow \{v, w\}$ is a single edge of P
- $\forall \{v, w\} \in E : m(\{v, w\}) > 1 \Rightarrow \{v, w\}$ is a multi-edge of P
- $\forall (v, L) \in E \Rightarrow (v, L)$ is a loop of P
- $\forall (v, S) \in E \Rightarrow (v, S)$ is a semi-edge of P

degree

The **degree** is a function $d : V \rightarrow \mathbb{N}; v \mapsto d(v)$. The value $d(v)$ is called the degree of the vertex v and is defined as follows:

$$d^e(v) = \sum_{e \in E^e(v)} m(e) \text{ with } E^e(v) = \left\{ e \in E \cap \binom{V}{2} \mid v \in e \right\}$$

$$d^L(v) = \begin{cases} 0 & \text{if } (v, L) \notin E \\ 2m((v, L)) & \text{if } (v, L) \in E \end{cases}$$

$$d^S(v) = \begin{cases} 0 & \text{if } (v, S) \notin E \\ m((v, S)) & \text{if } (v, S) \in E \end{cases}$$

$$d(v) = d^e(v) + d^L(v) + d^S(v)$$

◇

For this chapter we will redefine a simple graph such that the simple graphs form a subset of the pregraphs. This definition however is completely equivalent to the usual definition of a simple graph as given in Definition 1.1.1.

Definition 2.1.2 A **simple graph** G is an ordered pair $G = (V, \mathcal{E})$ with V the set of vertices and \mathcal{E} the multiset (E, m) of edges with $E \subseteq \binom{V}{2}$ and $m : E \rightarrow \{1\}$. \diamond **simple graph**

The bijection between the set of simple graphs using this definition and the set of simple graphs using the common definition is $(V, (E, m)) \mapsto (V, E)$.

As described in [59], besides pregraphs we are also interested in some related classes which are situated somewhere between simple graphs and pregraphs.

Definition 2.1.3 A **graph with loops** G_L is a pregraph $G_L = (V, (E, m))$ with $E \subseteq \binom{V}{2} \cup (V \times \{L\})$ and $m : E \rightarrow \{1\}$. \diamond **graph with loops**

Definition 2.1.4 A **graph with semi-edges** G_S is a pregraph $G_S = (V, (E, m))$ with $E \subseteq \binom{V}{2} \cup (V \times \{S\})$ and $m : E \rightarrow \{1\}$. \diamond **graph with semi-edges**

Definition 2.1.5 A **multigraph** G_M is a pregraph $G_M = (V, (E, m))$ with $E \subseteq \binom{V}{2}$. \diamond **multigraph**

Definition 2.1.6 A **graph with loops and semi-edges** G_{LS} is a pregraph $G_{LS} = (V, (E, m))$ with $m : E \rightarrow \{1\}$.

We will also refer to this type as **simple pregraphs**. \diamond **simple pregraph**

Definition 2.1.7 A **multigraph with loops** G_{ML} is a pregraph $G_{ML} = (V, (E, m))$ with $E \subseteq \binom{V}{2} \cup (V \times \{L\})$.

We will also refer to this type as **semi-edge-free pregraphs**. \diamond **semi-edge-free pregraph**

Definition 2.1.8 A **multigraph with semi-edges** G_{MS} is a pregraph $G_{MS} = (V, (E, m))$ with $E \subseteq \binom{V}{2} \cup (V \times \{S\})$.

We will also refer to this type as **loopless pregraphs**. \diamond **loopless pregraph**

In this chapter we will generate several types of cubic pregraphs and related classes. These are the classes we will consider:

Definition 2.1.9

- The class \mathcal{C} is the class of all simple cubic graphs.
- The class \mathcal{L} is the class of all cubic graphs with loops.
- The class \mathcal{S} is the class of all cubic graphs with semi-edges.
- The class \mathcal{M} is the class of all cubic multigraphs.
- The class \mathcal{LS} is the class of all cubic simple pregraphs.
- The class \mathcal{LM} is the class of all cubic semi-edge-free pregraphs.
- The class \mathcal{SM} is the class of all cubic loopless pregraphs.
- The class \mathcal{P} is the class of all cubic pregraphs.

◇

In Figure 2.1 we show the relations between these different classes of graphs.

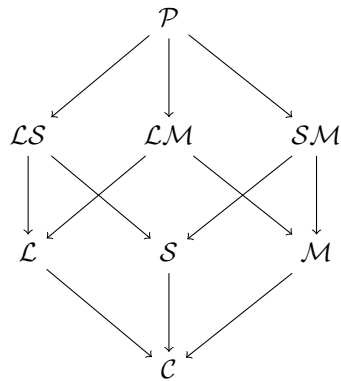


Figure 2.1: The relations between the different graph classes. In this figure $X \rightarrow Y$ means $Y \subseteq X$.

2.2 Cubic pregraph primitives

We will perform the generation in two steps. First we will generate coarser structures which we call pregraph primitives. From these pregraph primitives we will generate the pregraphs in a second step.

Definition 2.2.1 A **cubic pregraph primitive** is a multigraph $G(V, (E, m))$ such that each vertex $v \in V$ has a degree $d(v) \in \{1, 3\}$.
The set of cubic pregraph primitives is denoted by \mathcal{P}^* .

**cubic
pregraph
primitive**
 \mathcal{P}^*

We can now use this new class to define a transformation of \mathcal{P} into \mathcal{P}^* .

Definition 2.2.2 Primitivisation is a function $* : \mathcal{P} \rightarrow \mathcal{P}^*$; $G \mapsto G^*$, with G^* given by the following operations:

**primitivisa-
tion**

1. remove all loops;
2. add a vertex for each semi-edge, and replace the semi-edge by an edge connecting the new vertex and the vertex that was contained in the semi-edge.

For once, we will repeat this in a more detailed form. If G is the pregraph (V, \mathcal{E}) with

- V the set of vertices, and
- \mathcal{E} the multiset (E, m) of edges with $E \subseteq \binom{V}{2} \cup (V \times \{L\}) \cup (V \times \{S\})$ and $m : E \rightarrow \{1, 2, 3\}$,

then G^* is the multigraph (V^*, \mathcal{E}^*) with

- $V^* = V \cup \{v_1, \dots, v_s\}$, $\mathcal{E}^* = (E^*, m^*)$, and
- $E^* = (E \cap \binom{V}{2}) \cup \{e_1, \dots, e_s\}$ in which
 - $s = \sum_{v \in V} d^S(v)$,
 - $v_i \in e_i (i = 1, \dots, s)$,
 - $\forall (v, S) \in E : \exists E_v \subseteq \{e_1, \dots, e_s\} : |E_v| = d^S(v) \wedge \forall e \in E_v : v \in e$,
 - $\forall e \in E : m^*(e) = m(e)$, and
 - $\forall e \in E^* \setminus E : m^*(e) = 1$.

◇

If we extend our notation and denote the primitivisation of the sets \mathcal{LS} , \mathcal{LM} , \mathcal{SM} , \mathcal{S} , \mathcal{L} and \mathcal{M} by respectively \mathcal{LS}^* , \mathcal{LM}^* , \mathcal{SM}^* , \mathcal{S}^* , \mathcal{L}^* and \mathcal{M}^* . We have that $\mathcal{LS}^* = \mathcal{L}^* = \mathcal{S}^*$ (i.e., the cubic pregraph primitives that are simple graphs denoted by \mathcal{P}_1^*), $\mathcal{LM}^* = \mathcal{SM}^* = \mathcal{P}^*$ and $\mathcal{M}^* = \mathcal{M}$. For the cubic pregraph primitives the diagram shown in Figure 2.1 reduces to the diagram shown in Figure 2.2.

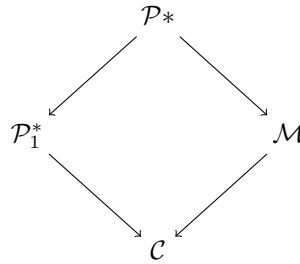


Figure 2.2: *The relations between the different pregraph primitive classes. In this figure $X \rightarrow Y$ means $Y \subseteq X$.*

2.3 Reduction operations for pregraph primitives

We will start by describing how we can reduce these cubic pregraph primitives to a smaller class of graphs. This will be the class of cubic graphs plus some extra graphs to handle some of the special cases. The inverse operations for these reductions then give us the construction operations to generate the cubic pregraph primitives from the cubic graphs and these extra graphs. We first give the operations we will be discussing.

The first operation, denoted by O_1 , (see Figure 2.3) is to unite two vertices of degree 1 that are not adjacent to the same vertices and add a new vertex that is adjacent to this newly united vertex. The reduction that corresponds to O_1 is to select a vertex x adjacent to a vertex y of degree 1 the removal of which breaks the graph into exactly two connected components, and not in three components, and

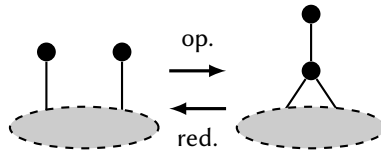


Figure 2.3: The construction operation O_1 and the corresponding reduction. The grey areas are connected.

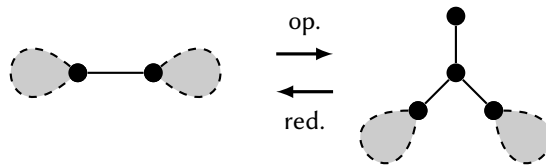


Figure 2.4: The construction operation O_2 and the corresponding reduction. The grey areas are connected.

then to remove x and y , add two new vertices x_1 and x_2 and connected each of the remaining original neighbours of x to one of these new vertices.

The second operation, denoted by O_2 , (see Figure 2.4) is to subdivide a bridge with a vertex x and connect x to a new vertex y . The reduction that corresponds to O_2 is to select a vertex x adjacent to a vertex y of degree 1 the removal of which breaks the graph into exactly three connected components. Remove the vertex y and the vertex x and connect the two remaining neighbours of x with each other.

The third operation, denoted by O_3 , (see Figure 2.5) is to subdivide an arbitrary edge (this might be a bridge) by inserting a double edge into it. If the original edge is the connection between the vertices x and y , then this edge is removed, two vertices x' and y' are added to the graph and x is connected to x' , y is connected to y' and x' is connected twice to y' . The reduction corresponding to O_3 is to select an edge with

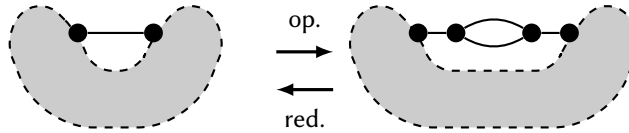


Figure 2.5: The construction operation O_3 and the corresponding reduction. The grey areas are not necessarily connected, but the whole graph is connected.

multiplicity 2 with the property that the neighbouring vertices are different. Let the neighbouring vertices be denoted by x and y , we remove the edge of multiplicity 2 and the vertices incident to it and connect the vertices x and y .

The fourth and final operation, denoted by O_4 , (see Figure 2.6) consists of replacing an edge of multiplicity 2 by another specific subgraph. Let the vertices of the edge with multiplicity 2 be x and y , and let the other neighbour of x , resp. y , be x' , resp. y' . We remove the edges between x and x' and between y and y' . We add two new vertices v and w and connect v to x' , y' and w , and connect w to x and y . The reduction that corresponds to O_4 is to select an edge with multiplicity 2 such that the two vertices x and y incident to that edge share one neighbour w , such that the other neighbour v of w has three different neighbours: w , x' and y' . We remove the vertices w and v and connect x' to x and y' to y .

We introduce a new notation: we denote a graph obtained by applying operation $x \in \{1, \dots, 4\}$ to the graph G by $O_x(G)$. In this case G is called the parent of $O_x(G)$ and $O_x(G)$ is called a child of G .

theta graph We will see that there is a special graph, namely the **theta graph**, which needs to be considered separately when working with these operations. For simplicity we will also consider the **buoy graph** a special case. These graphs are shown in Figure 2.7.

We will prove the following theorem:

Theorem 2.3.1 *Each cubic pregraph primitive can be reduced to either a cubic*

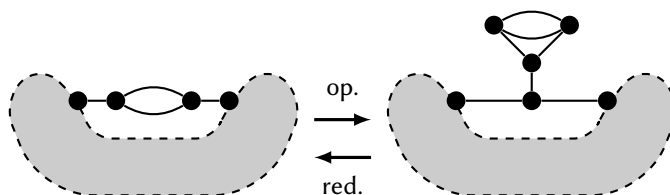


Figure 2.6: The construction operation O_4 and the corresponding reduction. The grey areas are not necessarily connected, but the whole graph is connected.

graph, K_2 or the theta graph using the reductions 1, 2, 3 and 4.

To prove this theorem we first need two lemmas.

Lemma 2.3.2 *Each cubic pregraph primitive containing a parallel edge can be reduced by reduction 3 or 4 to a cubic pregraph primitive with fewer vertices, except when it is the theta graph or the buoy graph.*

Proof: Let G be a graph different from the theta graph and the buoy graph. Consider two vertices u and v that are connected to each other by a parallel edge. Let x, y be the other two neighbours of u and v . If x and y are two distinct vertices we can apply reduction 3: we remove the vertices u and v and add an edge between x and y . This can create a new parallel edge, but the result is a cubic pregraph primitive. If $x = y$, x is adjacent to one other vertex z . Since G is different from the buoy graph, z has two other neighbours z_1 and z_2 . There are two cases for z_1 and z_2 : either they are different – in which case we can apply reduction 4 – or they are the same – in which case we can apply reduction 3 to the parallel edges between z and z_1 . ■

Lemma 2.3.3 *Each simple cubic pregraph primitive can be reduced to a cubic graph or K_2 by recursively applying reductions 1 and 2.*

(2.7.a) *Theta graph*(2.7.b) *Buoy graph*

Figure 2.7: *The theta graph and the buoy graph are two cubic pregraph primitives which will be considered separately when using our construction operations for cubic pregraph primitives.*

Proof: We will show that every cubic simple pregraph primitive G that is not K_2 and has at least one vertex u of degree 1 can be reduced to a smaller cubic simple pregraph primitive with fewer edges. Let v be the neighbour of u . Then v has degree 3, and is adjacent to two other distinct vertices x and y , because the cubic pregraph primitive contains no parallel edges. This means that we can apply either reduction 1, if the graph obtained by removing u and v is connected, or reduction 2, otherwise. In both cases we get a cubic pregraph primitive with fewer edges. ■

Now we have the tools to prove Theorem 2.3.1.

Proof of Theorem 2.3.1: The theorem follows from Lemma 2.3.2, Lemma 2.3.3 and from the fact that the buoy graph can also be reduced to K_2 by applying reduction 1 followed by reduction 3. ■

Remark 2.3.4 *The only cubic pregraph with an edge of multiplicity 3 is the theta graph. To simplify the implementation this graph is output separately in case the number of vertices is 2. In all other cases our implementation uses the buoy graph and $C_{4,d}$, i.e., a 4-cycle with additionally two opposite edges being double edges, as start graphs.*

2.4 Generation of cubic pregraph primitives

In order to assure that no isomorphic graphs are generated, we apply the canonical construction path method (see Subsection 1.4.6). Informally speaking we define a unique “canonical” last construction step for every graph and make sure that no two equivalent construction steps are performed. A graph is accepted, if and only if the last construction step is the canonical one.

In order to generate all cubic pregraph primitives with n vertices, the complete graph on 2 vertices K_2 and either the theta graph if $n = 2$ or the buoy graph and the graph $C_{4,d}$ in all other cases, the generation algorithm applies these steps recursively starting from the cubic graphs on up to n vertices. In its basic form the algorithm to generate cubic pregraph primitives with n vertices looks like this:

Let G be a graph that is to be extended. We use the word *pair* for (unordered) sets of two elements.

1. *If G has n vertices: output G . If G has no pair of degree-1 vertices that are not adjacent to the same vertex, then return.*
2. *Compute the automorphism group of G .*
3. *Compute the orbits of pairs of degree-1 vertices, the orbits of edges and the orbits of double edges.*
4. *For each orbit of pairs of degree-1 vertices, choose a representative and apply O_1 .
For each orbit of double edges, choose a representative and apply O_4 .
For each orbit of edges, choose a representative and :*
 - *if these edges are bridges, apply O_2 and O_3 ;*
 - *if these edges are not bridges, apply O_3 .*
5. *For each newly constructed graph G' compute the canonical last construction operation and accept G' if and only if the last construction operation is equivalent (details will follow) to the canonical one.*

The automorphism group is computed with *nauty* [29] — an efficient program for the computation of automorphism groups and canonical labellings of graphs. What

remains to be explained is the last step: determining the canonical last construction step.

Assume that we have a unique representative for every isomorphism class of graphs. A *canonical labelling* of a graph G is an assignment of labels to the vertices of G that produces the unique labelled representative of the isomorphism class of G . Canonical labellings are unique up to automorphisms of the graph.

We assign a pair of numbers $(n(v), l(v))$ to each vertex v of degree 1 and v is canonical if it has the lexicographically smallest pair $(n(v), l(v))$. The value $n(v)$ has been chosen to be easily computable. This value is equal to the number of vertices at distance at most 4 of v and thus lies somewhere between 2 (in case of the K_2) and 16. The value $l(v)$ is the canonical label for v . The value $n(v)$ is invariant under isomorphisms, so for all vertices v, w we have that if v and w are equivalent under the automorphism group of the graph, then $n(v) = n(w)$. Due to the definition of a canonical labelling, for all vertices v and w , we have that v and w are equivalent if and only if $l(v) = l(w)$. In most cases it is not necessary to calculate $l(v)$ in order to decide whether v is canonical.

We assign a pair of numbers $(a(e), b(e))$ to each reducible double edge e . The double edge e is canonical if it is reducible and it has the lexicographically smallest pair $(a(e), b(e))$. The value $a(e)$ is the minimum of the canonical labels of the vertices incident to e , and $b(e)$ is the maximum of these canonical labels.

The selection of the canonical last operation is in steps: If the graph contains double edges and is not the buoy graph, the last operation must be one of operations 3 or 4. Among all possible operations the one producing the canonical double edge is chosen as “the canonical last operation”. Note that since canonicity is unique only up to isomorphisms, an operation is accepted if it is equivalent to the chosen operation – that is if the double edge produced is in the orbit of the canonical double edge under the automorphism group of G' .

The canonical last operation for a simple cubic pregraph primitive with vertices of degree 1 is the one producing a degree-1 vertex in the orbit of the canonical vertex.

The algorithm is recursively applied to all base graphs (including the buoy graph, the graph $C_{4,d}$ and the complete graph on 2 vertices). Cubic graphs are obtained

from the generator in [58]. It follows from [43] (by fitting these operations into the proposed framework) but can also easily be seen and proven independently that in this way exactly one pregraph of each isomorphism class is generated.

Theorem 2.4.1 *If the algorithm described above is recursively applied, starting with all cubic graphs on up to n vertices, the complete graph on 2 vertices K_2 and either the theta graph if $n = 2$ or the buoy graph and the graph $C_{4,d}$ in all other cases, then it constructs exactly one representative of every isomorphism class of cubic pregraph primitives on n vertices.*

We will first prove some lemmas.

Lemma 2.4.2 *If the operations O_3 and O_4 are applied to one edge in each orbit of edges, respectively to one double edge in each orbit of double edges, for one representative of each isomorphism class of cubic pregraph primitives on $n - 2$ vertices, and the resulting graph G' is accepted if and only if the new double edge e has the lexicographically smallest value for $(a(e), b(e))$ among all reducible double edges in G' , then exactly one representative of each isomorphism class of cubic pregraph primitives on n vertices and containing parallel edges is accepted.*

Proof: We first observe that isomorphic graphs are constructed from the same parent. Indeed, due to the definition of $(a(e), b(e))$, if we have two isomorphic graphs G_1 and G_2 with respective new double edges e_1 and e_2 , and an isomorphism γ from G_1 to G_2 , then $\gamma(e_1)$ is in the same orbit as e_2 under the automorphism group of G_2 . Since a double edge cannot be reducible by both the inverse of O_3 and the inverse of O_4 , we have that reducing e_1 in G_1 and e_2 in G_2 produces isomorphic graphs.

Assume that a cubic pregraph primitive G on $n - 2$ vertices is the parent of two isomorphic graph G_1 and G_2 that are both accepted. We already established that this means that both G_1 and G_2 were obtained by applying the same operation to G and that there exists an isomorphism γ from G_1 to G_2 which maps the new double edge e_1 in G_1 to the new double edge e_2 in G_2 .

Assume first that G_1 , respectively G_2 , is obtained by applying operation O_3 to the edge $\{u, v\}$, respectively $\{x, y\}$, in G . This situation is illustrated in Figure 2.8 and we use the notation from this figure. The isomorphism γ maps e_1

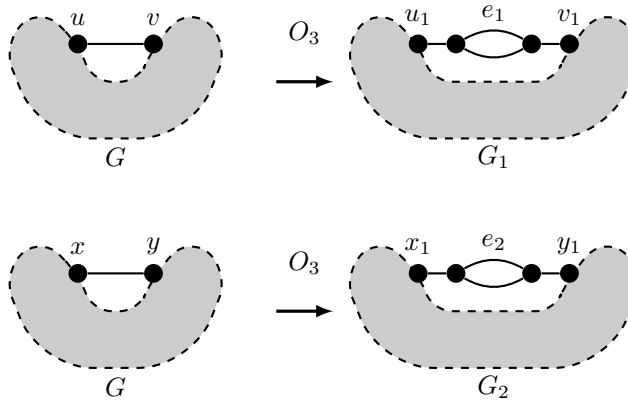


Figure 2.8: The graphs G_1 and G_2 are obtained by applying operation O_3 to G .

to e_2 and thus maps $\{u_1, v_1\}$ to $\{x_1, y_1\}$. This isomorphism induces an automorphism of G if we only consider the action on the vertices not contained in e_1 and e_2 . This means that $\{u, v\}$ and $\{x, y\}$ are in the same orbit of the automorphism group of G . This is, however, in contradiction with the procedure we used.

The case where the new double edges were obtained by applying operation O_4 is completely analogue. Assume that G_1 , respectively G_2 is obtained by applying operation O_4 to the double edge $\{u, v\}$, respectively $\{x, y\}$, in G . This situation is illustrated in Figure 2.9 and we use the notation from this figure. The isomorphism γ maps e_1 to e_2 and thus maps $\{u_1, v_1\}$ to $\{x_1, y_1\}$. This isomorphism induces an automorphism of G if we only consider the action on the vertices at distance at least 3 from e_1 and e_2 , and define the action on u and v as follows: u is mapped to x if u' is mapped to x' , and otherwise u is mapped to y ; v is mapped to y if v' is mapped to y' and otherwise v is mapped to x . In both cases $\{u, v\}$ is mapped to $\{x, y\}$, so they are in the same orbit

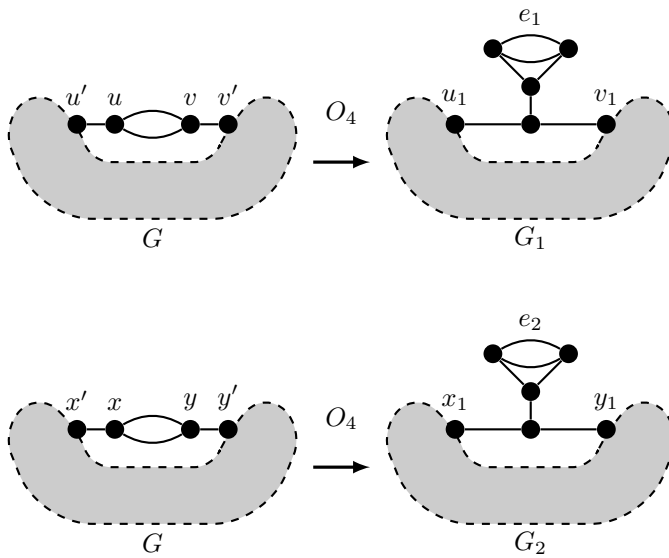


Figure 2.9: The graphs G_1 and G_2 are obtained by applying operation O_4 to G .

of the automorphism group of G . This is, however, in contradiction with the procedure we used.

That at least one representative of each isomorphism class of cubic pregraph primitives on n vertices and containing parallel edges is accepted, follows from the fact that each cubic pregraph primitive G on n vertices has a canonical double edge e the reduction of which leads to a cubic pregraph primitive G' on $n - 2$ vertices, and since applying O_3 , respectively O_4 , to edges, respectively double edges, in the same orbit gives isomorphic graphs, at some point G' was extended to form G by applying O_3 , respectively O_4 , and creating a double edge in the orbit of e under the automorphism group of G . ■

Lemma 2.4.3 *If the operations O_1 and O_2 are applied to one pair of degree-1 vertices in each orbit of pairs of degree-1 vertices, respectively to one bridge in each orbit of bridges, for one representative of each isomorphism class of simple cubic pregraph primitives on up to n vertices and for K_2 , and the resulting graph G' is accepted if and only if G' has at most n vertices, contains no parallel edges and the new degree-1 vertex v has the lexicographically smallest value for $(n(v), l(v))$ among all degree-1 vertices in G' , then exactly one representative of each isomorphism class of simple cubic pregraph primitives on at most n vertices containing at least one vertex of degree 1 is accepted, except for K_2 .*

Proof: The proof of this lemma uses more or less the same technique as used in the previous lemma. Again we first observe that also in this case isomorphic graphs are constructed from the same parent. Due to the definition of $(n(v), l(v))$, if we have two isomorphic graphs G_1 and G_2 with respective new degree-1 vertices v_1 and v_2 , and an isomorphism γ from G_1 to G_2 , then $\gamma(v_1)$ is in the same orbit as v_2 under the automorphism group of G_2 . Since a degree-1 vertex cannot be reducible by both the inverse of O_1 and the inverse of O_2 , we have that reducing v_1 in G_1 and reducing v_2 in G_2 produces isomorphic graphs.

Assume that G is a simple cubic pregraph primitive on up to n vertices or G is K_2 , so that G is the parent of two isomorphic graphs G_1 and G_2 that are both accepted. We already established that this means that both G_1 and G_2 were

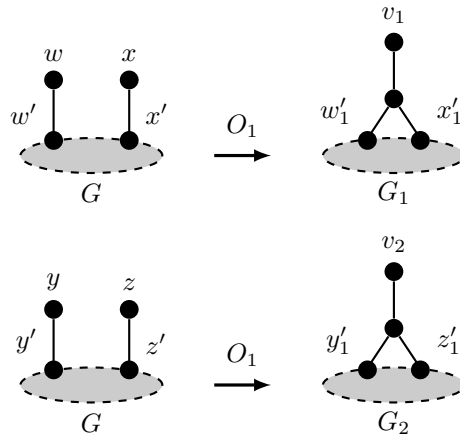


Figure 2.10: The graphs G_1 and G_2 are obtained by applying operation O_1 to G .

obtained by applying the same operation to G . This is in this case even more clear because by applying O_1 to G we obtain a graph with the same number of vertices as G , and by applying O_2 we obtain a graph with two vertices more than G . Also, we again have that there exists an isomorphism γ from G_1 to G_2 which maps the new degree-1 vertex v_1 in G_1 to the new degree-1 vertex v_2 in G_2 .

Assume first that G_1 , respectively G_2 , is obtained by applying operation O_1 to the pair of degree-1 vertices $\{w, x\}$, respectively $\{y, z\}$, in G . This situation is illustrated in Figure 2.10 and we use the notation from this figure. The isomorphism γ maps v_1 to v_2 and thus maps $\{w'_1, x'_1\}$ to $\{y'_1, z'_1\}$. This isomorphism induces an automorphism of G if we only consider the action on the vertices at distance at least 2 from v_1 and v_2 , and define the action on w and x as follows: w is mapped to y if w' is mapped to y' , and otherwise w is mapped to z ; x is mapped to z if x' is mapped to z' , and otherwise x is mapped to y . In both cases $\{w, x\}$ is mapped to $\{y, z\}$, so both pairs of degree-1 vertices are in the

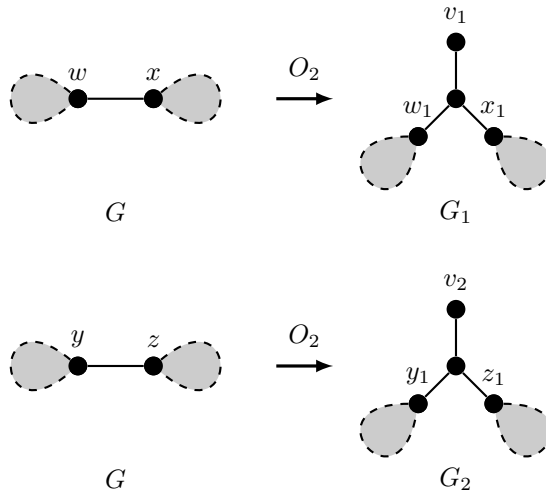


Figure 2.11: The graphs G_1 and G_2 are obtained by applying operation O_2 to G .

same orbit of the automorphism group of G . This is, however, in contradiction with the procedure we used.

Next we assume that G_1 , respectively G_2 , is obtained by applying operation O_2 to the edge $\{w, x\}$, respectively the edge $\{y, z\}$, in G . This situation is illustrated in Figure 2.11 and we use the notation from this figure. The isomorphism γ maps v_1 to v_2 and thus maps $\{w_1, x_1\}$ to $\{y_1, z_1\}$. This isomorphism induces an automorphism of G if we only consider the vertices at distance at least 2 of v_1 and v_2 . This means that $\{w, x\}$ is mapped to $\{y, z\}$, and thus both edges are in the same orbit of the automorphism group of G . This is, however, in contradiction with the procedure we used.

That at least one representative of each isomorphism class of simple cubic pregraph primitives on at most n containing at least one vertex of degree 1 is accepted, follows from the fact that each simple cubic pregraph primitive G on at most n vertices containing at least one degree-1 vertex has a canoni-

cal degree-1 vertex v the reduction of which leads to a simple cubic pregraph primitive G' on at most n vertices, and since applying O_1 , respectively O_2 , to pairs of degree-1 vertices, respectively edges, in the same orbit gives isomorphic graphs, at some point G' was extended to form G by applying O_1 , respectively O_2 , and creating a degree-1 vertex in the orbit of v under the automorphism group of G . The complete graph on 2 vertices K_2 cannot be reduced and thus will also not be constructed. ■

Together these lemmas can be used to prove Theorem 2.4.1.

Proof of Theorem 2.4.1: This theorem follows from Lemma 2.4.2 and Lemma 2.4.3, together with the observations that a graph G containing parallel edges is not accepted if it is obtained by applying operation O_1 or O_2 , that each graph which can be reduced to the theta graph is first reduced to $C_{4,d}$ and that all cubic pregraph primitives that are simple cubic graphs on n vertices are immediately output in the first step of the algorithm. ■

2.5 Generation of cubic pregraphs

In the previous sections we described a set of construction/reduction operations together with a generation algorithm which can be used to generate all cubic pregraph primitives starting from some special graphs and all cubic simple graphs. What remains to be described, is how to get the cubic pregraphs from these cubic pregraph primitives.

In case we are generating cubic multigraphs without loops and semi-edges, the cubic pregraph primitives are exactly the cubic pregraphs.

For the cases with loops, but without semi-edges, there is a one-to-one correspondence between the cubic pregraph primitives on n vertices and the cubic pregraphs with n vertices. For the cases with semi-edges, but without loops, there is a one-to-one correspondence between the cubic pregraph primitives with n vertices of degree

3 and the cubic pregraphs with n vertices. In all these cases we can just apply the inverse of the primitivisation function. If we are generating pregraphs with loops and no semi-edges, we just add a loop to each vertex of degree 1. If we are generating pregraphs with semi-edges and no loops, we remove all vertices of degree 1 and replace the edges incident to these vertices by semi-edges.

If we want both loops and semi-edges, we apply the homomorphism principle [32, 44]. If the order of the cubic pregraph primitive is n and we are generating pregraphs with n' vertices ($n' < n$) then exactly $k = n - n'$ edges incident to vertices of degree 1 must be transformed to semi-edges. We compute orbits of k -element subsets of the set of all edges to vertices of degree 1 and transform exactly one element of each orbit. Note that this step is very efficient, because in most cases the group is trivial, in which case each subset gives rise to a new graph.

2.6 3-edge-colourable pregraphs

Owing to the connection between maps and cubic pregraphs, a further class that is interesting consists of the cubic pregraphs that are 3-edge-colourable. The pregraphs with loops will never allow a proper edge-colouring, so we will not be considering them for this section and will mean loopless cubic pregraph whenever we write cubic pregraph in this section. Vizing's theorem for multigraphs gives the following restriction on the chromatic index $\chi'(G)$ of a multigraph G :

$$\Delta(G) \leq \chi'(G) \leq \Delta(G) + \mu(G)$$

where $\Delta(G)$ denotes the maximum degree of G and $\mu(G)$ denotes the maximum multiplicity of a multi-edge in G . In case of cubic pregraphs the maximum degree is always 3. The maximum multiplicity that occurs is also 3 in case of the theta graph. This graph is, however, easily verified to have a 3-edge-colouring. For all other graphs the maximum possible multiplicity is 2, so we find that

$$3 \leq \chi'(G) \leq 3 + 2 = 5$$

for a general cubic pregraph different from the theta graph.

Shannon [6] proved the following upper bound for the chromatic index:

$$\chi'(G) \leq \frac{3}{2}\Delta(G).$$

For the cubic pregraphs this upper bound is equal to 4.5, and thus each of the cubic pregraphs is either 3- or 4-edge-colourable.

There are two ways to generate all 3-edge-colourable cubic pregraphs. The first is just to generate all cubic pregraphs and filter out the 3-edge-colourable ones. The second is to modify the generation process such that we only generate the 3-edge-colourable cubic pregraphs. We will now show that it is possible to generate only the 3-edge-colourable cubic pregraphs with a minimal set of changes to the generation algorithm.

First the compatibility of the construction operations with the restriction to 3-edge-colourability will be examined and we start with operation 1. It is easy to see that a colouring of $O_1(G)$ implies a colouring of G . The other direction is not valid, because it is possible that the two edges incident to the respective vertices of degree 1 might have the same colour for all colourings of G . This means that we have the following implication:

$$G \text{ is not 3-edge-colourable} \Rightarrow O_1(G) \text{ is not 3-edge-colourable.}$$

A colouring of $O_2(G)$ implies a colouring of G , because the colours in the parts at either side of the bridge can be permuted. The other direction is also valid and uses the same argumentation. This means that we have the following implication:

$$G \text{ is 3-edge-colourable} \Leftrightarrow O_2(G) \text{ is 3-edge-colourable.}$$

In a proper 3-edge-colouring of $O_3(G)$ the two parallel edges get two different colours and thus the two edges incident to this multi-edge get the same colour. We can then colour the original edge in G with this colour and keep the rest the same. This leads to a proper 3-edge-colouring of G . Conversely, given a proper 3-edge colouring of $O_3(G)$, we keep the same colours as a proper colouring of G and colour the new multi-edge with the remaining two colours. This means that we have the following implication:

$$G \text{ is 3-edge-colourable} \Leftrightarrow O_3(G) \text{ is 3-edge-colourable.}$$

Finally we have operation 4, but due to the triangle with one edge of multiplicity 2 $O_4(G)$ will never be 3-edge-colourable.

In summary, we have found that the parent of a 3-edge-colourable graph is always 3-edge-colourable, so we only need to proceed with the generation when we have a 3-edge-colourable graph. Moreover, we have that in almost all cases a 3-edge-colourable parent leads to a 3-edge-colourable child. The only two exceptions are operations 1 and 4. Operation 4 never results in a 3-edge-colourable graph, so we just never apply this operation and the only special case remaining is operation 1. This operation can result in both 3-edge-colourable and non-3-edge-colourable graphs when applied to a 3-edge-colourable graph. So we need to perform a check whether we have a 3-edge-colourable graph after applying this operation, but to perform this check we can use stored information.

We only start the generation process with 3-edge-colourable graphs. When we have calculated a 3-edge-colouring for a start graph, we store this colouring and update it while applying the construction operations. When applying operation 1 to two edges e, e' we know that the resulting graph is 3-edge-colourable if and only if a 3-edge-colouring with e, e' of different colour exists. As a consequence of the Parity Lemma (see p.6) such a colouring never exists in the case of only two vertices with degree 1. In case of more vertices with degree 1 and operation 1 applied to two edges e, e' with the same colour, the new graph needs to be checked for 3-edge-colourability. In order to try to avoid expensive calculations for colourability, we first run a cheap test to try and change the original colouring in the original graph. We modify the original colouring by Kempe-chains starting at e or e' to check whether we can find a colouring where the colours of e and e' differ. We do not try to construct completely different colourings. In cases where even Kempe chains do not give a colouring with e and e' of different colours, we check the newly constructed graph for being 3-edge-colourable.

Using the modified generation algorithm above we can generate the following classes:

- The class \mathcal{C}_c is the class of all simple cubic graphs that are 3-edge-colourable.

- The class $\mathcal{S}c$ is the class of all simple cubic pregraphs without loops that are 3-edge-colourable.
- The class $\mathcal{M}c$ is the class of all cubic multigraphs that are 3-edge-colourable.
- The class $\mathcal{S}\mathcal{M}c$ is the class of all cubic pregraphs without loops that are 3-edge-colourable.

We can also use these results together with earlier results to determine the numbers of graphs in the following classes:

- The class $\mathcal{C}\bar{c}$ is the class of all simple cubic graphs that are not 3-edge-colourable.
- The class $\mathcal{S}\bar{c}$ is the class of all simple cubic pregraphs without loops that are not 3-edge-colourable.
- The class $\mathcal{M}\bar{c}$ is the class of all cubic multigraphs that are not 3-edge-colourable.
- The class $\mathcal{S}\mathcal{M}\bar{c}$ is the class of all cubic pregraphs without loops that are not 3-edge-colourable.

Note that we cannot use the modified generation algorithm to generate these classes.

2.7 Bipartite pregraphs

People who are interested in applications of bipartite pregraphs are referred to [59]. In this section we will show that bipartite pregraphs can be efficiently generated by a slight modification of the generation algorithm. A graph with a loop is not bipartite, so we will not consider loops here.

First the compatibility of the construction operations with the restriction to bipartite graphs will be examined. We start with Operation 1. Let v, w be the two vertices of degree 1 to which the operation is applied. Using the characterisation of bipartite graphs as graphs without odd cycles, the following equivalence is easy to see:

$$G \text{ is bipartite and } v \text{ and } w \text{ have an even distance} \Leftrightarrow O_1(G) \text{ is bipartite.}$$

As no cycles are constructed or modified, for operation 2 we immediately have that

$$G \text{ is bipartite} \Leftrightarrow O_2(G) \text{ is bipartite.}$$

For operation 3 we have

$$G \text{ is bipartite} \Leftrightarrow O_3(G) \text{ is bipartite.}$$

as the length of possible even cycles remains or changes by 2 and a new cycle of length 2 can vanish or come into existence.

Finally we have operation 4, but because of the triangle, $O_4(G)$ is not bipartite for all G .

In summary, we find that the parent of a bipartite graph is also bipartite, so we only need to start from bipartite graphs. As the algorithm described in [58] does not efficiently generate bipartite graphs, we use the algorithm from [35] for the generation of bipartite simple cubic start graphs. The child of a bipartite graph is always a bipartite graph except in the case of operation 1, where we have the extra restriction that the two vertices of degree 1 need to lie at an even distance of each other, so we only apply this operation to orbits of pairs of degree 1 vertices at even distance and we never apply operation 4.

Using the modified generation algorithm above we can generate the following classes:

- The class \mathcal{CB} is the class of all simple cubic graphs that are bipartite.
- The class \mathcal{SB} is the class of all simple cubic pregraphs without loops that are bipartite.
- The class \mathcal{MB} is the class of all cubic multigraphs that are bipartite.
- The class \mathcal{SMB} is the class of all cubic pregraphs without loops that are bipartite.

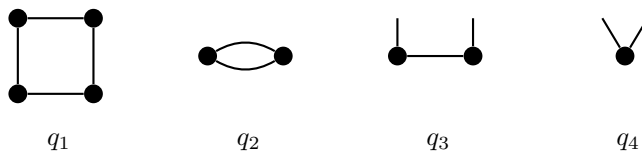


Figure 2.12: The quotients of C_4 .

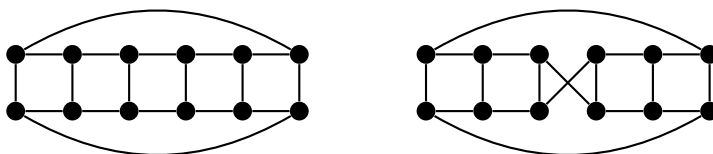


Figure 2.13: A prism (left) and a Möbius ladder (right)

2.8 Pregraphs admitting a 2-factor composed of quotients of C_4

Connected cubic pregraphs that allow a 2-factor where each component is a quotient of C_4 (see Figure 2.12) are of particular interest, because they correspond to Delaney-Dress graphs [59]. In the flag graph the σ_0, σ_2 -components are cycles of order 4, so in the Delaney-Dress graph these components have become quotients of C_4 . We use a semi-edge rather than a loop to represent the case where we have a flag d for which $\sigma_i d = d$ for a certain i . This representation has the advantage that the Delaney-Dress graph is cubic.

It is not possible to make small changes to the algorithm in order to generate only pregraphs that admit a 2-factor composed of quotients of C_4 , as was the case for 3-edge-colourable pregraphs and bipartite pregraphs. In this section we will discuss an algorithm that can efficiently filter out these graphs.

Definition 2.8.1 A **ladder** is a maximal subgraph that is isomorphic to the graph **ladder**

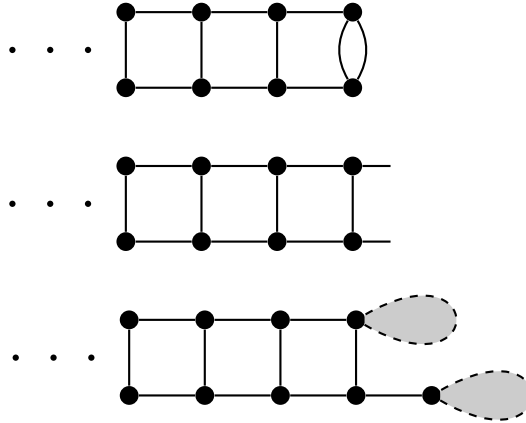


Figure 2.14: Possible end points of ladders. The grey area stands for an arbitrary structure — possibly just a semi-edge. It is also allowed that these structures are connected to each other.

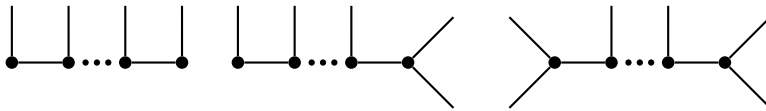


Figure 2.15: Possible types of paths with semi-edges

cartesian product of K_2 and the path P_n with n vertices for some $n \geq 2$.

prism

A **prism** is a graph that is isomorphic to the graph cartesian product of K_2 and C_n for some $n > 2$ (see the left side of Figure 2.13).

Möbius ladder

A **Möbius ladder** is a graph that is isomorphic to the graph on vertices $0, \dots, 2n-1$ with vertex i adjacent to $(i-1) \bmod 2n$, $(i+1) \bmod 2n$ and $(i+n) \bmod 2n$ where $n > 1$ (see the right side of Figure 2.13).

crown

A **crown** is a cycle with each vertex additionally incident to one semi-edge. \diamond

Lemma 2.8.2 A cubic pregraph G has a unique partition into subgraphs that are ladders, subgraphs induced by 2-cycles not contained in a ladder and the components of the subgraph induced by the complement of these.

Proof: If G is the theta graph, then the unique partition contains one part.

Assume G is not the theta graph. Since G is cubic, the intersection of two ladders in G , respectively of two 2-cycles in G , is empty. The definition of the partition does not allow a vertex to be in two different types of parts of the partition. This proves this lemma. ■

Theorem 2.8.3 *It can be tested in linear time whether a pregraph has a C_4^q -factor – a factor of quotients of C_4 .*

Proof: Pregraphs that are prisms, crowns or Möbius ladders can be detected in linear time. Prisms and Möbius ladders have a C_4^q -factor if and only if the number of vertices is a multiple of 4. Crowns have a C_4^q -factor if and only if the number of vertices is a multiple of 2.

We thus assume that G is a pregraph that is neither a prism nor a crown or a Möbius ladder. We can partition G into subgraphs induced by ladders, subgraphs induced by 2-cycles not contained in ladders and the components of the subgraph induced by the complement of these, and, due to Lemma 2.8.2, this partition is unique. This can be done in linear time in a straight forward way.

We will show that G has a C_4^q -factor if and only if each component has a C_4^q -factor. The direction that C_4^q -factors of the components imply (in fact even form) a C_4^q -factor of G is trivial; so we will concentrate on the other direction. We will prove that restricting a C_4^q -factor F of G to one of the subgraphs G_s in the partition gives a C_4^q -factor of G_s . All edges not in the partitioning subgraphs are not contained in a 2- or 4-cycle; so if they were contained in F their component would be isomorphic to q_3 or q_4 . But on the other hand they must have at least one endpoint in a 2-cycle or a ladder, which means that one endpoint is a vertex adjacent to 3 (non-semi) edges – a contradiction. Therefore all edges in F are in the partitioning subgraphs.

What remains to be shown is that the components can be checked for the existence of C_4^q -factors in linear time. Now 2-cycles are a C_4^q -factor themselves.

The existence of C_4^q -factors in a ladder depends only on the number of vertices of the ladder and the neighbours of the 4 boundary vertices (the vertices not contained in an edge that is the intersection of two 4-cycles). If either one of the endpoints of the ladder is of one of the two first types in Figure 2.14, then this ladder has a C_4^q -factor. If both endpoints are of the third type, then this ladder has a C_4^q -factor if and only if the number of vertices in this ladder is a multiple of four. Therefore this can also be tested in linear time or even constant time if some data is stored during the computation of the partition.

Now let C be a component not containing 2- or 4-cycles. The only candidate components of a C_4^q -factor are q_3 and q_4 ; so assume that each vertex has degree at least two and contains at least one semi-edge – otherwise no C_4^q -factor exists. As G is not a crown, C is a path with additionally at least one semi-edge at every vertex. The three possible types for C are shown in Figure 2.15. Such a graph has a C_4^q -factor if and only if the number of vertices is even or C contains a vertex adjacent to two semi-edges. This can again be tested in linear time or even constant time if some data is stored during the computation of the partition.

Therefore all the tests can be performed in linear time. ■

Using a program implementing the filtering algorithm as described above, we can generate – among others – the following classes:

- The class $\mathcal{C}q$ is the class of all simple cubic graphs having a C_4^q -factor.
- The class $\mathcal{S}q$ is the class of all simple cubic pregraphs having a C_4^q -factor.
- The class $\mathcal{M}q$ is the class of all cubic multigraphs having a C_4^q -factor.
- The class $\mathcal{S}\mathcal{M}q$ is the class of all cubic pregraphs without loops having a C_4^q -factor.
- The class $\mathcal{C}\mathcal{B}q$ is the class of all bipartite simple cubic graphs having a C_4^q -factor.

- The class $\mathcal{SB}q$ is the class of all bipartite simple cubic pregraphs without loops having a C_4^q -factor.
- The class $\mathcal{MB}q$ is the class of all bipartite cubic multigraphs having a C_4^q -factor.
- The class $\mathcal{SMB}q$ is the class of all bipartite cubic pregraphs without loops having a C_4^q -factor.

By slightly modifying the filtering algorithm described above, we can generate — among others — the following classes:

- The class $\mathcal{C}4$ is the class of all simple cubic graphs having a C_4 -factor.
- The class $\mathcal{S}4$ is the class of all simple cubic pregraphs having a C_4 -factor.
- The class $\mathcal{M}4$ is the class of all cubic multigraphs having a C_4 -factor.
- The class $\mathcal{SM}4$ is the class of all cubic pregraphs without loops having a C_4 -factor.
- The class $\mathcal{CB}4$ is the class of all bipartite simple cubic graphs having a C_4 -factor.
- The class $\mathcal{SB}4$ is the class of all bipartite simple cubic pregraphs without loops having a C_4 -factor.
- The class $\mathcal{MB}4$ is the class of all bipartite cubic multigraphs having a C_4 -factor.
- The class $\mathcal{SMB}4$ is the class of all bipartite cubic pregraphs without loops having a C_4 -factor.

2.9 Testing

In order to gain more certainty that not only the algorithms but also the implementations are correct, we tested as many results as possible against independently

generated data. In the following tables numbers that have been independently confirmed are given in bold font. The generator described here has been tested for each class up to 18 vertices against an independent generator. This independent generator is a slightly modified version of a generator for multigraphs — using similar techniques as the generator in [35] — that has already been extensively tested.

The 3-edge-colourable and bipartite pregraphs have been obtained in two independent ways. First by filtering all pregraphs. Second by altering the generation algorithm to only generate 3-edge-colourable pregraphs or bipartite pregraphs as was discussed in Sections 2.6 and 2.7. The numbers of structures have been compared up to 18 vertices.

The filtering program described in Section 2.8 has been compared to manual computations up to 7 vertices and tested up to 20 vertices against a different generator which will be described in the next chapter.

One of the first things to notice when looking at Table 2.1 is that the numbers for class \mathcal{L} coincide with the numbers for class \mathcal{M} , and the numbers for class \mathcal{LS} coincide with the numbers for class \mathcal{SM} except for $n = 1$. This can easily be explained by looking at Figure 2.16. This figure shows how each double edge can be transformed to a loop and vice versa. This transformation can always be performed except for the balloon graph shown in Figure 2.17. This explains the difference of 1 between \mathcal{LS} and \mathcal{SM} in case $n = 1$. This connection was only noticed by looking at the results of the generator; so this was not used when developing the generator. Since both cases are handled by different methods, they provide an extra test for the generation program.

2.10 Results

The following pages give an overview of the results obtained using the algorithms in this chapter. This algorithm was implemented as the program `pregraphs` and is available from [64]. The website also contains several other related programs such as the filters used for testing the program.

Besides the counts for the different classes we also give the timings for the program `pregraphs` and — where meaningful — the generation rate for the same program.

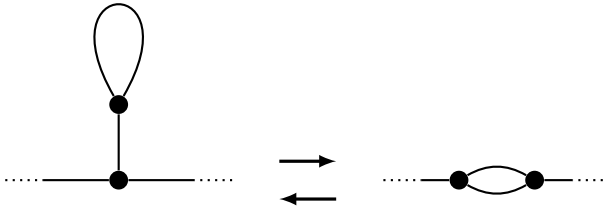


Figure 2.16: Each occurrence of a loop can be transformed into a double edge and vice versa.



Figure 2.17: The balloon graph: the only graph in which the loop cannot be transformed into a double edge. This is because there is only one vertex.

n	C	\mathcal{L}	S	\mathcal{M}	$\mathcal{L}\mathcal{S}$	$\mathcal{L}\mathcal{M}$	$\mathcal{S}\mathcal{M}$	\mathcal{P}
1	0	0	1	0	2	0	1	2
2	0	1	1	1	3	2	3	5
3	0	0	2	0	4	0	4	7
4	1	2	6	2	12	5	12	22
5	0	0	10	0	22	0	22	43
6	2	6	29	6	68	17	68	141
7	0	0	64	0	166	0	166	373
8	5	20	194	20	534	71	534	1270
9	0	0	531	0	1589	0	1589	4053
10	19	91	1733	91	5464	388	5464	14671
11	0	0	5524	0	18579	0	18579	52826
12	85	509	19430	509	68320	2592	68320	203289
13	0	0	69322	0	255424	0	255424	795581
14	509	3608	262044	3608	1000852	21096	1000852	3241367
15	0	0	1016740	0	4018156	0	4018156	13504130
16	4060	31856	4101318	31856	16671976	204638	16671976	57904671
17	0	0	16996157	0	70890940	0	70890940	253856990
18	41301	340416	72556640	340416	309439942	2317172	309439942	1139231977
19	0	0	317558689	0	1381815168	0	1381815168	5219113084
20	510489	4269971	1424644848	4269971	6310880471	30024276	6310880471	24401837085
21	0	0	6536588420	0	29428287639	0	29428287639	116278408069
22	7319447	61133757	30647561117	61133757	140012980007	437469859	140012980007	564380686932
23	0	0	146647344812	0	70671095998	0	70671095998	
24	117940535	978098997	978098997	978098997				

Table 2.1: The number of structures for each class for a given number of vertices n . The numbers in bold have been independently verified.

n	\mathcal{C}	\mathcal{L}	\mathcal{S}	\mathcal{M}	\mathcal{LS}	\mathcal{LM}	\mathcal{SM}	\mathcal{P}
10	0.0s	0.0s	0.0s	0.0s	0.1s	0.0s	0.1s	0.1s
11	0.0s	0.0s	0.1s	0.0s	0.2s	0.0s	0.3s	0.4s
12	0.0s	0.0s	0.6s	0.0s	0.8s	0.0s	1.3s	1.8s
13	0.0s	0.0s	2.2s	0.0s	3.5s	0.0s	5.4s	7.4s
14	0.0s	0.0s	9.1s	0.1s	14.9s	0.2s	22.6s	32.2s
15	0.0s	0.0s	37.3s	0.0s	64.1s	0.0s	97.2s	144.5s
16	0.0s	0.3s	158.3s	0.5s	290.1s	2.5s	427.1s	669.5s
17	0.0s	0.0s	695.9s	0.0s	1372.7s	0.0s	1931.5s	3192.3s
18	0.1s	3.0s	3182.2s	5.1s	6552.1s	31.0s	8933.5s	15725.4s
19	0.0s	0.0s	14398.5s	0.0s	32533.2s	0.0s	42194.7s	78738.8s
20	1.4s	39.0s	67781.7s	67.9s	164334.4s	441.9s	203152.1s	404351.9s
21	0.0s	0.0s	329875.5s	0.0s	853461.3s	0.0s	997604.8s	2128059.3s
22	18.6s	577.2s	1627712.4s	1044.1s	4549317.5s	7058.5s	4985448.0s	11440675.6s
23	0.0s	0.0s	8088214.3s	0.0s		0.0s		
24	298.4s	9620.6s		18022.4s		124630.6s		

Table 2.2: *The timings for the numbers in Table 2.1 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. For smaller n the CPU time is less than 0.1s.*

n	\mathcal{L}	\mathcal{S}	\mathcal{M}	\mathcal{LS}	\mathcal{LM}	\mathcal{SM}	\mathcal{P}
20	109486.4/s	21018.1/s	62886.2/s	38402.7/s	67943.6/s	31064.8/s	60348.0/s
21		19815.3/s		34481.1/s		29498.9/s	54640.6/s
22	105914.3/s	18828.6/s	58551.6/s	30776.7/s	61977.7/s	28084.3/s	49331.1/s
23		18131.0/s					
24	101667.2/s		54271.3/s		56704.4/s		

Table 2.3: *The average number of structures per second for the numbers in Table 2.1 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. (Bottom part of the table only)*

n	\mathcal{C}_c	\mathcal{S}_c	\mathcal{M}_c	\mathcal{SM}_c
1	0	1	0	1
2	0	1	1	3
3	0	2	0	3
4	1	6	2	11
5	0	9	0	17
6	2	28	5	59
7	0	59	0	134
8	5	187	16	462
9	0	501	0	1332
10	17	1679	65	4774
11	0	5310	0	16029
12	80	18989	363	60562
13	0	67461	0	225117
14	475	257738	2588	898619
15	0	997460	0	3598323
16	3848	4052146	23702	15128797
17	0	16762252	0	64261497
18	39687	71905738	263952	283239174
19	0	314293531	0	1264577606
20	496430	1414799656	3438642	5817868002
21	0	6484967876	0	27138011161
22	7174735	30479739145	50763502	129848052113
23	0	145735267008	0	
24	116214038		831898577	

Table 2.4: The number of 3-edge-colourable structures for each class for a given number of vertices n . Classes allowing loops are omitted because they never allow a proper 3-edge-colouring. The numbers in bold have been verified using a filtering program on the original set of graphs.

n	$\mathcal{C}c$	$\mathcal{S}c$	$\mathcal{M}c$	$\mathcal{S}\mathcal{M}c$
10	0.0s	0.0s	0.0s	0.1s
11	0.0s	0.2s	0.0s	0.3s
12	0.0s	0.6s	0.0s	1.2s
13	0.0s	2.4s	0.0s	4.9s
14	0.0s	9.3s	0.0s	20.6s
15	0.0s	39.2s	0.0s	88.3s
16	0.0s	164.1s	0.3s	395.7s
17	0.0s	740.1s	0.0s	1794.5s
18	0.2s	3245.6s	3.4s	8245.1s
19	0.0s	15254.9s	0.0s	39076.4s
20	3.0s	70520.4s	48.3s	191074.5s
21	0.0s	349170.5s	0.0s	932273.4s
22	47.1s	1722625.2s	791.7s	4683143.7s
23	0.0s	8491130.8s	0.0s	
24	886.3s		14271.1s	

Table 2.5: The timings for the numbers in Table 2.4 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. For smaller n the CPU time is less than 0.1s.

n	$\mathcal{S}c$	$\mathcal{M}c$	$\mathcal{S}\mathcal{M}c$
20	20062.3/s	71193.4/s	30448.2/s
21	18572.5/s		29109.5/s
22	17693.8/s	64119.6/s	27726.7/s
23	17163.2/s		
24		58292.5/s	

Table 2.6: The average number of structures per second for the numbers in Table 2.4 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. (Bottom part of the table only)

n	$\mathcal{C}\bar{c}$	$\mathcal{S}\bar{c}$	$\mathcal{M}\bar{c}$	$\mathcal{SM}\bar{c}$
1	0	0	0	0
2	0	0	0	0
3	0	0	0	1
4	0	0	0	1
5	0	1	0	5
6	0	1	1	9
7	0	5	0	32
8	0	7	4	72
9	0	30	0	257
10	2	54	26	690
11	0	214	0	2550
12	5	441	146	7758
13	0	1861	0	30307
14	34	4306	1020	102233
15	0	19280	0	419833
16	212	49172	8154	1543179
17	0	233905	0	6629443
18	1614	650902	76464	26200768
19	0	3265158	0	117237562
20	14059	9845192	831329	493012469
21	0	51620544	0	2290276478
22	144712	167821972	10370255	10164927894
23	0	912077804	0	
24	1726497		146200420	

Table 2.7: *The number of non-3-edge-colourable structures for each class for a given number of vertices n . Classes with loops are omitted because they never allow a proper 3-edge-colouring. These numbers have been obtained by subtracting the numbers in Table 2.4 from the numbers in Table 2.1.*

n	C_q	S_q	M_q	SM_q	C_4	S_4	M_4	SM_4
1		1		1				
2		1	1	3				
3		1		2				
4	1	4	2	9	1	3	2	5
5		3		7				
6	0	10	3	29				
7		9		27				
8	3	34	9	105	3	13	5	20
9		34		118				
10	0	98	14	392				
11		125		546				
12	10	367	48	1722	10	77	19	130
13		526		2701				
14	0	1352	95	7953				
15		2234		13966				
16	43	5710	331	40035	43	660	88	1194
17		10187		75341				
18	0	24938	873	210763				
19		47568		420422				
20	242	116186	3145	1162192	242	7559	553	14629

Table 2.8: *The number of structures for each class that have a C_4^q -factor and those that have a C_4 -factor for a given number of vertices n . The numbers in bold have been verified against manual computations and all numbers have been verified against another generator described in the next chapter.*

n	Cq	Sq	Mq	SMq	$C4$	$S4$	$M4$	$SM4$
10	0.0s	0.0s	0.0s	0.1s	0.0s	0.0s	0.0s	0.0s
11	0.0s	0.2s	0.0s	0.3s	0.0s	0.0s	0.0s	0.0s
12	0.0s	0.6s	0.0s	1.3s	0.0s	0.6s	0.0s	1.2s
13	0.0s	2.4s	0.0s	5.2s	0.0s	0.0s	0.0s	0.0s
14	0.0s	9.5s	0.0s	22.0s	0.0s	0.0s	0.0s	0.0s
15	0.0s	39.5s	0.0s	94.8s	0.0s	0.0s	0.0s	0.0s
16	0.0s	168.7s	0.3s	420.5s	0.0s	164.5s	0.3s	402.8s
17	0.0s	743.4s	0.0s	1903.5s	0.0s	0.0s	0.0s	0.0s
18	0.0s	3341.9s	3.8s	8850.1s	0.0s	0.0s	0.0s	0.0s
19	0.0s	15407.8s	0.0s	41812.1s	0.0s	0.0s	0.0s	0.0s
20	2.2s	72708.7s	54.0s	201745.4s	1.6s	70550.7s	49.7s	193667.8s

Table 2.9: The timings for the numbers in Table 2.8 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. For smaller n the CPU time is less than 0.1s.

n	Sq	Mq	SMq	$C4$	$S4$	$M4$	$SM4$
16	33.8/s		95.2/s	4.0/s		3.0/s	
17	13.7/s		39.6/s				
18	7.5/s	229.7/s	23.8/s				
19	3.1/s		10.1/s				
20	1.6/s	58.2/s	5.8/s	0.1/s	11.1/s	0.1/s	

Table 2.10: The average number of structures per second for the numbers in Table 2.8 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. (Bottom part of the table only)

n	CB	SB	MB	SMB	n	CB/C	SB/S	MB/M	SMB/SMB
1	0	1	0	1	1		1.000		1.000
2	0	1	1	3	2		1.000	1.000	1.000
3	0	1	0	2	3		0.500		0.500
4	0	3	1	8	4	0.000	0.500	0.500	0.667
5	0	4	0	10	5		0.400		0.455
6	1	12	3	34	6	0.500	0.414	0.500	0.500
7	0	18	0	59	7		0.281		0.355
8	1	52	6	188	8	0.200	0.268	0.300	0.352
9	0	101	0	426	9		0.190		0.268
10	2	295	15	1348	10	0.105	0.170	0.165	0.247
11	0	701	0	3631	11		0.127		0.195
12	5	2074	48	11650	12	0.059	0.107	0.094	0.171
13	0	5636	0	35038	13		0.081		0.137
14	13	17252	177	115756	14	0.026	0.066	0.049	0.116
15	0	51480	0	374569	15		0.051		0.093
16	38	164209	773	1280586	16	0.009	0.040	0.024	0.077
17	0	524392	0	4370641	17		0.031		0.062
18	149	1744885	4046	15465234	18	0.004	0.024	0.012	0.050
19	0	5874275	0	55067190	19		0.018		0.040
20	703	20354298	24759	201370245	20	0.001	0.014	0.006	0.032
21	0	71599949	0	743390634	21		0.011		0.025
22	4132	257656099	174469	2804028685	22	0.001	0.008	0.003	0.020
23	0	941820046	0	10690490079	23				
24	29579	3510119105	1387042	41516954063	24	0.000		0.001	

Table 2.11: The number of bipartite graphs for a given number of vertices n and the ratio of bipartite graphs versus the total number of graphs. The numbers in bold have been verified using a filtering program on the original set of graphs.

n	CB	SB	MB	SMB
11	0.0s	0.0s	0.0s	0.1s
12	0.0s	0.1s	0.0s	0.3s
13	0.0s	0.3s	0.0s	1.1s
14	0.0s	1.2s	0.0s	3.8s
15	0.0s	3.8s	0.0s	13.6s
16	0.0s	12.8s	0.0s	50.4s
17	0.0s	44.6s	0.0s	186.7s
18	0.0s	156.4s	0.1s	709.4s
19	0.0s	562.2s	0.0s	2727.4s
20	0.1s	2060.0s	0.8s	10663.7s
21	0.0s	7643.1s	0.0s	42106.2s
22	0.2s	28850.9s	5.9s	168857.8s
23	0.0s	111135.1s	0.0s	685140.0s
24	1.4s	432532.0s	50.2s	2819258.6s

Table 2.12: *The timings for the numbers in Table 2.11 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. For smaller n the CPU time is less than 0.1s.*

n	SB	MB	SMB
21	9367.9/s		17655.1/s
22	8930.6/s	29571.0/s	16605.9/s
23	8474.6/s		15603.4/s
24	8115.3/s	27630.3/s	14726.2/s

Table 2.13: *The average number of structures per second for the numbers in Table 2.11 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. (Bottom part of the table only)*

n	CBq	SBq	MBq	$SMBq$	n	CBq/CB	SBq/SB	MBq/MB	$SMBq/SMB$
1	1	1	1	1	1	1.000	1.000	1.000	1.000
2	0	1	1	3	2	1.000	1.000	1.000	1.000
3	1	1	1	2	3	1.000	1.000	1.000	1.000
4	0	2	1	7	4	0.667	0.667	1.000	0.875
5	2	2	6	6	5	0.500	0.500	0.667	0.600
6	0	6	2	22	6	0.000	0.500	0.667	0.647
7	6	6	21	7	7	0.333	0.333	0.667	0.356
8	1	18	4	70	8	1.000	0.346	0.667	0.372
9	19	19	83	9	9	0.188	0.188	0.400	0.195
10	0	49	6	243	10	0.000	0.166	0.400	0.180
11	63	63	351	11	11	0.090	0.090	0.333	0.097
12	2	159	16	969	12	0.400	0.077	0.333	0.083
13	227	227	1578	13	13	0.040	0.040	0.147	0.045
14	0	520	26	4096	14	0.000	0.030	0.147	0.035
15	854	854	7414	15	15	0.017	0.017	0.094	0.020
16	5	1908	73	18739	16	0.132	0.012	0.094	0.015
17	3357	3357	36177	17	17	0.006	0.006	0.037	0.008
18	0	7283	150	89612	18	0.000	0.004	0.037	0.006
19	13662	13662	182071	19	19	0.002	0.002	0.017	0.003
20	13	29435	430	446890	20	0.018	0.001	0.017	0.002

Table 2.14: The number of bipartite graphs that allow a 2-factor where each component is a quotient of C_4 for a given number of vertices n , and the ratio of bipartite pregraphs with such a 2-factor versus the total number of bipartite pregraphs..

	CBq	SBq	MBq	$SMBq$
11	0.0s	0.0s	0.0s	0.1s
12	0.0s	0.1s	0.0s	0.3s
13	0.0s	0.4s	0.0s	1.1s
14	0.0s	1.2s	0.0s	4.0s
15	0.0s	3.9s	0.0s	14.4s
16	0.0s	13.1s	0.0s	52.6s
17	0.0s	45.2s	0.0s	194.9s
18	0.0s	159.0s	0.1s	738.6s
19	0.0s	570.5s	0.0s	2832.1s
20	0.1s	2086.4s	0.8s	11080.4s

Table 2.15: *The timings for the numbers in Table 2.14 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. For smaller n the CPU time is less than 0.1s.*

	SBq	MBq	$SMBq$
16	145.6/s		356.3/s
17	74.3/s		185.6/s
18	45.8/s		121.3/s
19	23.9/s		64.3/s
20	14.1/s	537.5/s	40.3/s

Table 2.16: *The average number of structures per second for the numbers in Table 2.14 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. (Bottom part of the table only)*

n	CB_4	SB_4	MB_4	SMB_4
4	0	1	1	3
8	1	5	2	9
12	2	17	5	38
16	5	92	14	226
20	13	592	47	1704

n	CB_4/CB	SB_4/SB	MB_4/MB	SMB_4/SMB
4		0.333	1.000	0.375
8	1.000	0.096	0.333	0.048
12	0.400	0.008	0.104	0.003
16	0.132	0.001	0.018	0.000
20	0.018	0.000	0.002	0.000

Table 2.17: The number of bipartite graphs that allow a 2-factor where each component is a C_4 for a given number of vertices n and the ratio of bipartite pregraphs with such a 2-factor versus the total number of bipartite pregraphs.

n	CB_4	SB_4	MB_4	SMB_4
12	0.0s	0.1s	0.0s	0.3s
16	0.0s	12.9s	0.0s	51.1s
20	0.0s	2059.1s	0.8s	10840.6s

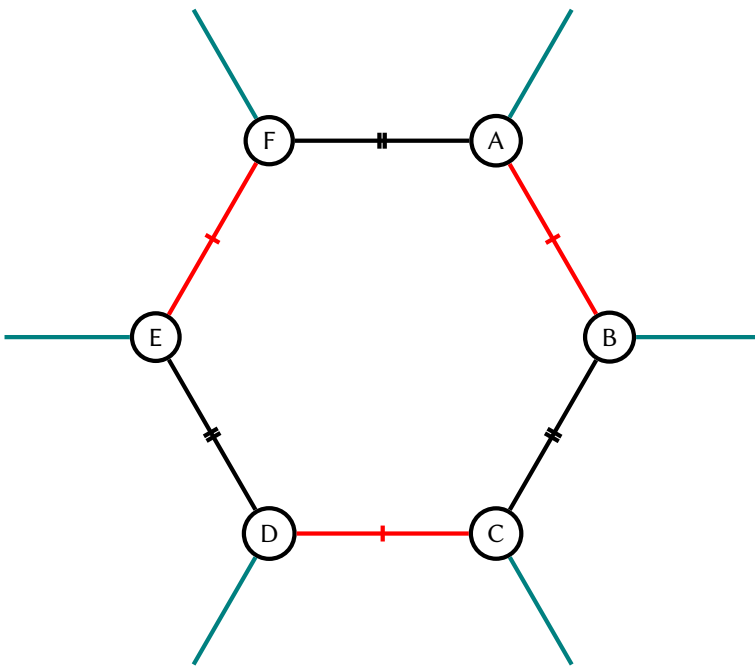
Table 2.18: The timings for the numbers in Table 2.17 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. For smaller n the CPU time is less than 0.1s.

n	CB_4	SB_4	MB_4	SMB_4
16	7.1/s		4.4/s	
20	0.3/s	58.8/s	0.2/s	

Table 2.19: The average number of structures per second for the numbers in Table 2.17 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. (Bottom part of the table only)

3

Delaney-Dress symbols



The last class of pregraphs we discussed in the previous chapter, were the pregraphs that have a C_4^q -factor. We showed that there exists an efficient algorithm which can decide in time linear in the order of the pregraph whether a pregraph belongs to this class. If we however look at the percentage of pregraphs which belong to this class, we see that this percentage is very low and decreases rapidly when the number of vertices increases. Up to 20 vertices Table 3.1 shows the number of pregraphs with a C_4^q -factor compared to the 3-edge-colourable pregraphs, i.e., the class of pregraphs generated in the previous chapter which is best used to filter for these pregraphs with a C_4^q -factor. Once we reach 20 vertices already 99.98% of the graphs that are generated do not have a C_4^q -factor.

This shows that filtering, although the test is very efficient, is not the best approach for the generation of this class of pregraphs. A specialised generation algorithm that does not need to go via the larger class of 3-edge-colourable pregraphs can be much more efficient for this generation problem. This alternative approach is the subject of the first part of this chapter. We go even further than in the previous chapter and generate all possible Delaney-Dress graphs, i.e., pregraphs with a C_4^q -factor together with a colour assignment that corresponds to a valid Delaney-Dress graph. Afterwards we use these Delaney-Dress graphs as a starting point to generate Delaney-Dress symbols of tilings of the Euclidean plane.

3.1 C_4^q -marked pregraphs

C_4^q -marked
pregraph

Definition 3.1.1 A C_4^q -*marked pregraph* is a cubic pregraph in which all the edges of a given C_4^q -factor are coloured with colour 0 and all other edges with colour 1. \diamond

The colouring in a C_4^q -marked pregraph clearly is not a proper edge-colouring as each vertex is incident to two edges of colour 0. It is also clear that the underlying uncoloured graph of a C_4^q -marked pregraph is a pregraph which has a C_4^q -factor. The following theorem shows that for a given n for almost all pregraphs on n vertices that have a C_4^q -factor there is a unique C_4^q -factor up to isomorphism.

n	colourable	has C_4^q -factor	ratio
1	1	1	100.00 %
2	3	3	100.00 %
3	3	2	66.67%
4	11	9	81.82%
5	17	7	41.18%
6	59	29	49.15%
7	134	27	20.15%
8	462	105	22.73%
9	1 332	118	8.86%
10	4 774	392	8.21%
11	16 029	546	3.41%
12	60 562	1 722	2.84%
13	225 117	2 701	1.20%
14	898 619	7 953	0.89%
15	3 598 323	13 966	0.39%
16	15 128 797	40 035	0.26%
17	64 261 497	75 341	0.12%
18	283 239 174	210 763	0.07%
19	1 264 577 606	420 422	0.03%
20	5 817 868 002	1 162 192	0.02%

Table 3.1: Comparison of the number of 3-edge-colourable pregraphs on n vertices and the number of pregraphs with a C_4^q -factor on n vertices.

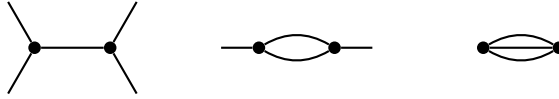


Figure 3.1: The three C_4^q -markable pregraphs on two vertices.

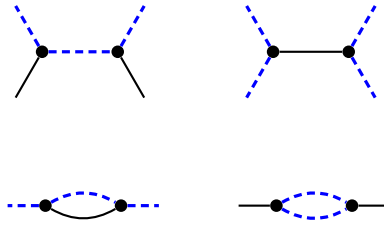


Figure 3.2: The two C_4^q -markable pregraphs on two vertices that have non-isomorphic C_4^q -factors. The marked C_4^q -factors are shown with dashed lines.

Theorem 3.1.2 For each integer $n > 0$ the number of C_4^q -markable pregraphs that have multiple pairwise non-isomorphic C_4^q -factors, depends only on $n \bmod 4$:

- $n \bmod 2 \equiv 1$: 0 pairs
- $n \bmod 4 \equiv 0$: 4 pairs
- $n \bmod 4 \equiv 2$: 2 pairs

Each C_4^q -markable pregraph has at most two non-isomorphic C_4^q -factors.

Proof: It is easily verified that there are 3 pregraphs on 2 vertices which have a C_4^q -factor (see Figure 3.1) and that there are 5 C_4^q -marked pregraphs on 2 vertices. The two pairs of C_4^q -marked pregraphs that have the same underlying uncoloured pregraph are shown in Figure 3.2.

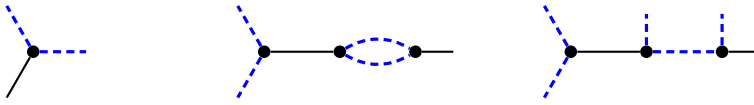


Figure 3.3: All C_4^q -marked pregraphs on one and three vertices. The marked C_4^q -factors are shown with dashed lines.

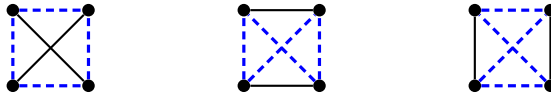


Figure 3.4: The three isomorphic C_4^q -factors in a Möbius ladder on four vertices. The marked C_4^q -factors are shown with dashed lines.

It is also easily verified that the C_4^q -factor is unique for all pregraphs on 1 and 3 vertices. Figure 3.3 shows all C_4^q -marked pregraphs on 1 and 3 vertices.

A crown on $n \geq 4$ vertices is C_4^q -markable if n is even. A C_4^q -markable crown has two isomorphic C_4^q -factors. A C_4^q -factor in a crown consists of alternating edges in the cycle together with the semi-edges incident to each of the edges. There are two isomorphic sets of alternating edges in the cycle.

A Möbius ladder on $n \geq 4$ vertices is C_4^q -markable if n is divisible by four. A Möbius ladder on four vertices has three isomorphic C_4^q -factors (see Figure 3.4). A C_4^q -markable Möbius ladder on more than four vertices has two isomorphic C_4^q -factors (see Figure 3.5).

A prism on $n \geq 4$ vertices is C_4^q -markable if n is divisible by four. A prism on four vertices does not exist. A prism on eight vertices is a cube. It is easily seen that a cube has three isomorphic C_4^q -factors: each C_4^q -factor corresponds to the edges of two opposite ‘faces’ of the cube when viewed as a solid. A

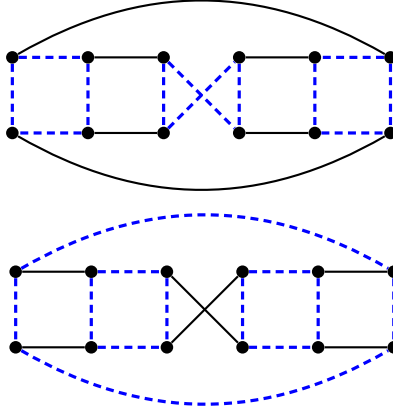


Figure 3.5: The two isomorphic C_4^q -factors in a C_4^q -markable Möbius ladder on more than four vertices (here twelve vertices). The marked C_4^q -factors are shown with dashed lines.

C_4^q -markable prism on $n > 8$ vertices has two isomorphic C_4^q -factors (see Figure 3.6).

Let P be a pregraph on $n \geq 4$ vertices which has a C_4^q -factor and is not a crown, Möbius ladder or prism. We will show that, except in a few cases, there is only one C_4^q -factor in P . In Lemma 2.8.2 we saw that P has a unique partition into subgraphs induced by ladders, subgraphs induced by digons not contained in ladders and the components of the subgraph induced by the complement of these.

If P contains a digon that is not contained in a ladder, then at least one of the two vertices x and y of the digon is incident to an edge e that is not a semi-edge and that is not contained in the digon. Since e is not contained in a C_4 or a digon, and at least one of the vertices of e is not incident to a semi-edge, e can not be part of a quotient of C_4 , and so the original digon is part of the C_4^q -factor.

If P contains a ladder, then there are two possible situations. The first case is

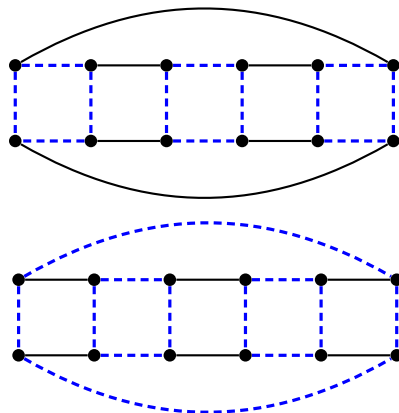


Figure 3.6: The two isomorphic C_4^q -factors in a C_4^q -markable prism on more than eight vertices (here twelve vertices). The marked C_4^q -factors are shown with dashed lines.

that there is a boundary vertex x of the ladder (a vertex not contained in an edge that is the intersection of two 4-cycles) that is incident to an edge e that is not a semi-edge and is not contained in a C_4 . Using a similar argumentation as with the digon, it is easy to see that e cannot be contained in a quotient of C_4 . This means that the C_4 which contains x is part of any C_4^q -factor of P . This also fixes any C_4^q -factor in this ladder. The second case is that the ladder contains all the vertices of P , but P is not a prism or a Möbius ladder. In this case we can look at the boundary vertices to determine the possible C_4^q -factors. Assume first that there are two boundary vertices which are connected by an edge e which is not contained in a C_4 or a digon. Again e cannot be contained in a quotient of C_4 and this implies that also in this case the C_4^q -factor is unique. There are still 3 graphs containing ladders which we haven't discussed. These graphs are shown in Figure 3.7, Figure 3.8 and Figure 3.9.

Let us first consider the graph shown in Figure 3.7. This graph is a ladder which is bounded by two digons. The edges at one side with a digon are in

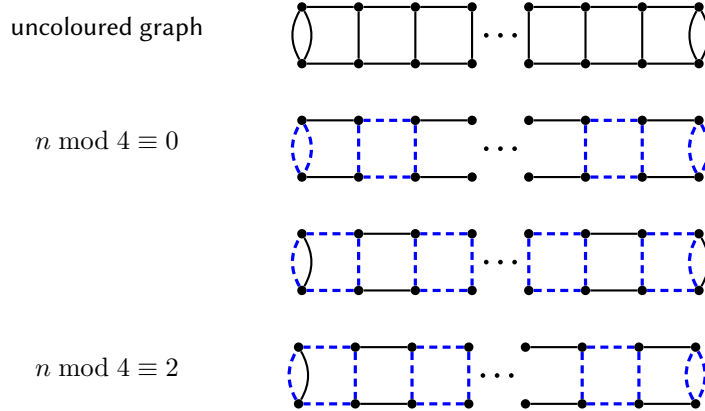


Figure 3.7: A ladder bounded by two digons has two different C_4^q -factors when its order is divisible by 4 and two isomorphic C_4^q -factors otherwise. The marked C_4^q -factors are shown with dashed lines.

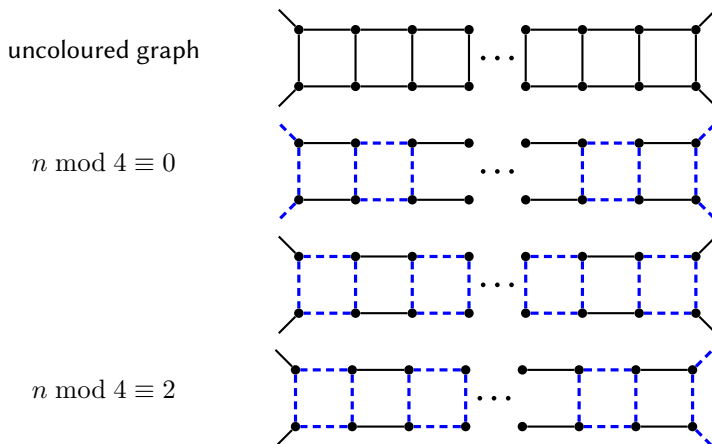


Figure 3.8: A ladder bounded by four semi-edges has two different C_4^q -factors when its order is divisible by 4 and two isomorphic C_4^q -factors otherwise. The marked C_4^q -factors are shown with dashed lines.

two ways contained in a quotient of C_4 : either the digon itself is the quotient, or one of the edges of the digon together with the rest of the C_4 in which it is contained is the quotient. In both cases the rest of the C_4^q -factor is unique for the complete graph. In case $n \bmod 4 \equiv 0$ this means that there are two non-isomorphic C_4^q -marked pregraphs, and in case $n \bmod 4 \equiv 2$ there are two isomorphic C_4^q -marked pregraphs.

The graph in Figure 3.8 is a ladder bounded by four semi-edges. Again there are two ways the edges at one side can be contained in a C_4^q -factor and also in this case this means that there are two non-isomorphic C_4^q -marked pregraphs when $n \bmod 4 \equiv 0$ and two isomorphic C_4^q -marked pregraphs when $n \bmod 4 \equiv 2$.

Finally we have the graph shown in Figure 3.9. In this graph, one side of the ladder is bounded by a digon, while the other side is bounded by two semi-edges. Again the edges at one side with a digon are in two ways contained in a quotient of C_4 , but here these two C_4^q -factors are non-isomorphic for all n .

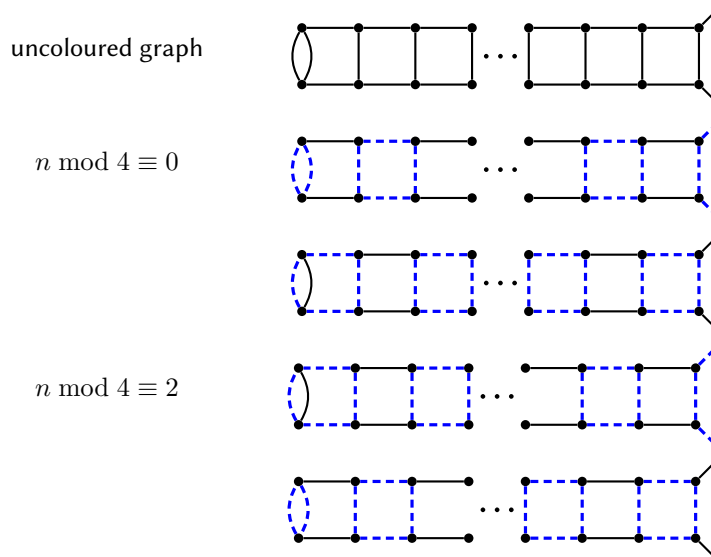


Figure 3.9: A ladder bounded by a digon and two semi-edges has two different C_4^q -factors. The marked C_4^q -factors are shown with dashed lines.

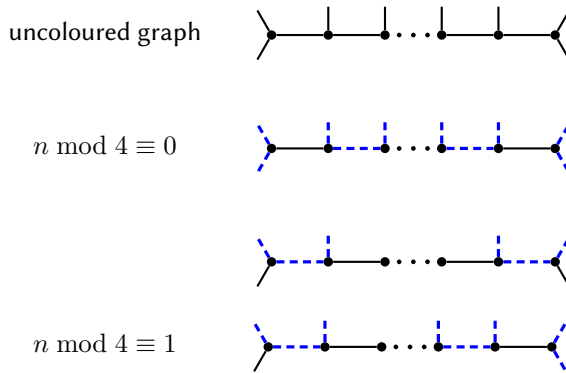


Figure 3.10: A path where each vertex is incident to at least one semi-edge and the two end vertices are incident to two semi-edges has two different C_4^q -factors when its order is even and two isomorphic C_4^q -factors otherwise. The marked C_4^q -factors are shown with dashed lines.

The C_4^q -factor is fixed in ladders and in digons that are not part of a ladder. The last step is to fix the C_4^q -factor in the rest of P . From the proof of Theorem 2.8.3 we know that this remainder consists of paths where each vertex is additionally also incident to at least one semi-edge. First assume that an end vertex x of such a path is incident to exactly one semi-edge. As we saw earlier, this implies that in P the vertex x is incident to an edge e that has a non-empty intersection with a digon or a ladder, and thus e cannot be contained in a quotient of C_4 . This again fixes the C_4^q -factor for the complete path and we find that there is a unique C_4^q -factor for P in this case.

So assume now that both end vertices are incident to exactly two semi-edges. This is only possible if P is the graph shown in Figure 3.10. If we look at one of the end vertices of P in this case, we see that this vertex is contained in a quotient of C_4 in two ways: either the two semi-edges are the quotient, or one of the two semi-edges, the third edge and the semi-edge at the neighbouring ver-

tex are the quotient. Either choice fixes the C_4^q -factor for the complete graph. In case n is even, this means that there are two non-isomorphic C_4^q -factors in this pregraph, and in case n is odd, both C_4^q -factors are isomorphic. ■

Due to the previous theorem we can easily modify a generation algorithm for C_4^q -marked pregraphs to also generate pregraphs which have a C_4^q -factor. We just need to omit the colours when outputting the graphs and make sure that we correctly handle the small number of graphs which lead to isomorphic uncoloured pregraphs.

So let us look at how we can generate C_4^q -marked pregraphs. We already established in the proof of Theorem 2.8.3 that a C_4^q -marked pregraph P has a unique partition into subgraphs induced by ladders, subgraphs induced by digons not contained in ladders and the components of the subgraph induced by the complement of these. We can refine this partition, such that each part contains only one type of quotients of C_4 .

**block
partition**

Definition 3.1.3 Given a C_4^q -marked pregraph P , a **block partition** of P is a partition of P into subgraphs of the following types:

1. maximal ladders containing only marked quotients of type q_1 ;
2. maximal subgraphs induced by marked quotients of type q_2 ;
3. maximal subgraphs induced by marked quotients of type q_3 ;
4. marked quotients of type q_4 .

Such a partition is unique for each C_4^q -marked pregraph P .

blocks

The different subgraphs in a block partition are called **blocks**. ◇

The possible blocks containing quotients of type q_1 that can occur in a block partition are shown in Figure 3.11 and Figure 3.12 and an overview is given in Table 3.2. We only show the blocks with 2 quotients of type q_1 , but most block types exist for any integer $n \geq 1$ of quotients of type q_1 – some block types do not exist for $n = 1$. This integer n is called the parameter of the block and Table 3.2 also gives the minimum value of n for each type of block. The reason that for some block types

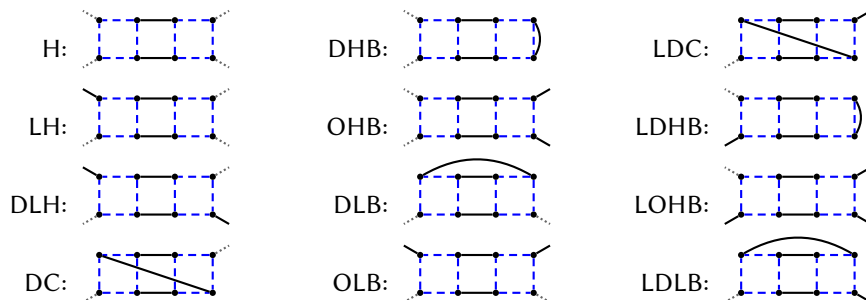


Figure 3.11: The possible blocks containing quotients of type q_1 that can occur in a block partition when the partition contains more than one block. The dotted edges mark the vertices from where connections to other blocks need to be made. The marked quotients of type q_1 are drawn with dashed lines. All the blocks that are shown here have parameter 2.

a parameter larger than 1 is required, is to prevent having different descriptions for the same block. Otherwise DLB(1) would be isomorphic to DHB(1), OLB(1) would be isomorphic to OHB(1), LDLB(1) would be isomorphic to LDHB(1), DLDLB(1) would be isomorphic to DLDHB(1) and P(1) would be isomorphic to DDHB(1). These blocks can be determined by starting from a ladder with an order that is divisible by four and adding edges between the vertices of degree 2 or adding a semi-edge to these vertices. This was done by a straightforward ad-hoc script, since no specialised techniques are needed for this small set of blocks.

The possible blocks containing quotients of type q_2 that can occur in a block partition are shown in Figure 3.13 and Figure 3.14 and an overview is given in Table 3.3. Also in this case these blocks exist for different parameters.

The possible blocks containing quotients of type q_3 that can occur in a block partition are shown in Figure 3.15 and Figure 3.16 and an overview is given in Table 3.4. Also in this case these blocks exist for different parameters.

The possible blocks containing quotients of type q_4 that can occur in a block partition are shown in Figure 3.17 and Figure 3.18 and an overview is given in Table 3.5.

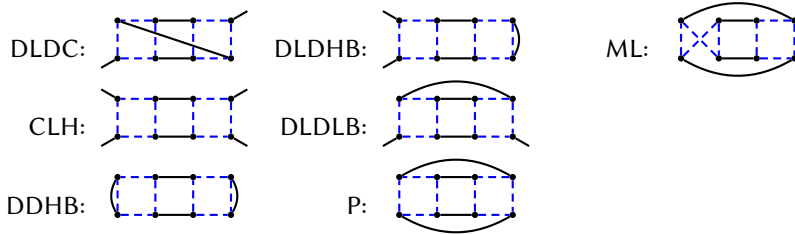


Figure 3.12: The possible blocks containing quotients of type q_1 that can occur in a block partition when the partition contains one block. The marked quotients of type q_1 are drawn with dashed lines. All the blocks that are shown here have parameter 2.

Abbreviation	Name	Connections	Minimum Parameter
H	Hub	4	1
LH	Locked Hub	3	1
DLH	Diagonally Locked Hub	2	1
DC	Diagonal Chain	2	1
DHB	Double-roof High Building	2	1
OHB	Open-roof High Building	2	1
DLB	Double-roof Long Building	2	2
OLB	Open-roof Long Building	2	2
LDC	Locked Diagonal Chain	1	1
LDHB	Locked Double-roof High Building	1	1
LOHB	Locked open-roof High Building	1	1
LDLB	Locked Double-roof Long Building	1	2
DLDC	Double-Locked Diagonal Chain	0	1
CLH	Completely Locked Hub	0	1
DDHB	Double-roof Double-floor High Building	0	1
DLDHB	Double-Locked Double-roof High Building	0	1
DLDLB	Double-Locked Double-roof Long Building	0	2
P	Prism	0	2
ML	Möbius ladder	0	1

Table 3.2: The possible blocks containing quotients of type q_1 that can occur in a block partition.



Figure 3.13: The possible blocks containing quotients of type q_2 that can occur in a block partition when the partition contains more than one block. The dotted edges mark the vertices from where connections to other blocks need to be made. The marked quotients of type q_2 are drawn with dashed lines. All the blocks that are shown here have parameter 2.



Figure 3.14: The possible blocks containing quotients of type q_2 that can occur in a block partition when the partition contains one block. The marked quotients of type q_2 are drawn with dashed lines. The DLPC block is shown with parameter 2 and the PN block is shown with parameter 5.

Abbreviation	Name	Connections	Minimum Parameter
PC	Pearl Chain	2	1
LPC	Locked Pearl Chain	1	1
DLPC	Double-Locked Pearl Chain	0	1
PN	Pearl Necklace	0	1

Table 3.3: The possible blocks containing quotients of type q_2 that can occur in a block partition.



Figure 3.15: The possible blocks containing quotients of type q_3 that can occur in a block partition when the partition contains more than one block. The dotted edges mark the vertices from where connections to other blocks need to be made. The marked quotients of type q_3 are drawn with dashed lines. All the blocks that are shown here have parameter 2.

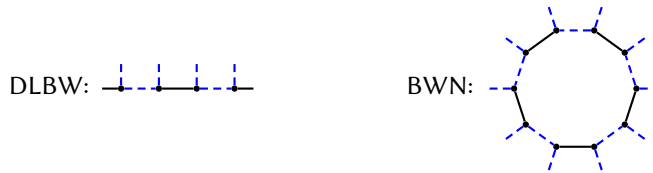


Figure 3.16: The possible blocks containing quotients of type q_3 that can occur in a block partition when the partition contains one block. The marked quotients of type q_3 are drawn with dashed lines. The DLBW block is shown with parameter 2 and the BWN block is shown with parameter 5.

Abbreviation	Name	Connections	Minimum Parameter
BW	Barbed Wire	2	1
LBW	Locked Barbed Wire	1	1
DLBW	Double-Locked Barbed Wire	0	1
BWN	Barbed Wire Necklace	0	1

Table 3.4: The possible blocks containing quotients of type q_3 that can occur in a block partition.

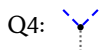


Figure 3.17: The only possible block containing a quotient of type q_4 that can occur in a block partition when the partition contains more than one block. The grey edge marks the vertex from where the connection to another block needs to be made. The marked quotient of type q_4 is drawn with dashed lines.



Figure 3.18: The only possible block containing a quotient of type q_4 that can occur in a block partition when the partition contains one block. The marked quotient of type q_4 is drawn with dashed lines.

For this type of blocks there is no parameter.

We will use block partitions to generate the C_4^q -marked pregraphs. This generation process happens in several phases. To generate all the C_4^q -marked pregraphs with n vertices we start by generating all lists of blocks such that the sum of the orders of the blocks is equal to n . This is done by a simple orderly algorithm without many optimisations since the time spent in the generation of these lists is negligible compared to the rest of the generation process. Since each C_4^q -marked pregraph corresponds to exactly one such list of blocks, different lists will result in different C_4^q -marked pregraphs.

We can perform a few tests to discard lists that are not realisable as a block

Abbreviation	Name	Connections
Q4	Q4	1
T	Tristar	0

Table 3.5: The possible blocks containing a quotient of type q_4 that can occur in a block partition.

partition of a C_4^q -marked pregraph. We can view a C_4^q -marked pregraph with a block partition as a multigraph: the vertices of the multigraph are the blocks of the block partition, the edges of the multigraph are the edges between the blocks. This means that when we have a list of blocks we can check whether the degree sequence that corresponds to that list is realisable as a multigraph without loops.

Degree sequences are well studied, and we will give a summary of the results that we use.

**degree
sequence**

Definition 3.1.4 Given a graph G with n vertices, the **degree sequence** $d = \langle d_1, d_2, \dots, d_n \rangle$ of G is the monotonic non-increasing sequence of the degrees of the vertices of G . \diamond

graphic

Definition 3.1.5 An monotonic non-increasing sequence of positive integers is called **graphic** if it can be the degree sequence of a graph. \diamond

**multi-
graphic**

Definition 3.1.6 An monotonic non-increasing sequence of positive integers is called **multigraphic** if it can be the degree sequence of a multigraph without loops. \diamond

Since the number of edges of a (multi)graph is equal to half the sum of the degrees of that graph, a necessary condition for a sequence $a = \langle a_1, a_2, \dots, a_n \rangle$ to be (multi)graphic, is that $\sum_{k=1}^n a_k$ is even.

If loops are allowed, then each sequence with an even sum is realisable as a multigraph. Such a realisation is easily constructed. There are an even number of vertices with odd degree. Divide the vertices with odd degree into pairs and connect the vertices in each pair to each other. Complete the graph by adding loops until all degree conditions are satisfied.

Erdős and Gallai proved the following characterisation of graphical sequences.

Theorem 3.1.7 (Erdős and Gallai [7]) A monotonic non-increasing sequence $a = \langle a_1, a_2, \dots, a_n \rangle$ of positive integers is graphical if and only if $\sum_{k=1}^n a_k$ is even and for each integer i , $1 \leq i \leq n - 1$,

$$\sum_{k=1}^i a_k \leq i(i-1) + \sum_{k=i+1}^n \min(i, a_k).$$

Tripathi and Vijay showed that it is not necessary to examine all possible values for i in this theorem.

Corollary 3.1.8 (Tripathi and Vijay [49]) *Let $a = \langle a_1, a_2, \dots, a_n \rangle$ be a monotonic non-increasing sequence of positive integers. Let s be the largest integer such that $a_s \geq s - 1$. Then the sequence a is graphical if and only if $\sum_{k=1}^n a_k$ is even and for each integer i , $1 \leq i \leq s$,*

$$\sum_{k=1}^i a_k \leq i(i-1) + \sum_{k=i+1}^n \min(i, a_k).$$

A loopless multigraph can be transformed into a simple graph using the following procedure: As long as there exist two vertices that are connected by at least two parallel edges, subdivide one of these edges by creating a new vertex of degree 2. This transformation forms the basis for the following result.

Theorem 3.1.9 (Owens and Trent [12]) *Let t be an integer, let $a = \langle a_1, a_2, \dots, a_n \rangle$ be a monotonic non-increasing sequence of positive integers, and let a' be the monotonic non-increasing sequence of positive integers obtained by adding t copies of the integer 2 to a , then a is the degree sequence of a loopless multigraph, whose underlying graph contains at least $\sum_{k=1}^n a_k - t$ edges if and only if the sequence a' is graphic.*

This gives us the following characterisation of multigraphical sequences.

Corollary 3.1.10 *A monotonic non-increasing sequence $a = \langle a_1, a_2, \dots, a_n \rangle$ of positive integers is multigraphical if and only if there exists a $t \geq 0$ such that the monotonic non-increasing sequence of positive integers obtained by adding t copies of the integer 2 to a is graphical.*

A list that does not satisfy the condition above will not be realisable as a multigraph and therefore will not be valid, but not all lists that satisfy this condition will occur as block partition of a C_4^q -markable pregraph. Another restriction we need to take into account when connecting the blocks is that not every connection is valid. A condition for the unicity of the block partition was that the blocks were maximal. This means that connecting a q_2 block to a q_2 block or a q_3 block to a q_3 block is not

allowed. Neither is it allowed to connect two q_1 blocks such that two of the connecting edges are part of a C_4 . So we can add the following test to discard some more blocks: if more than half of the connections are connections at q_2 blocks or more than half of the connections are connections at q_3 blocks, then this list won't be realisable.

acceptable **Definition 3.1.11** *A list L of blocks is **acceptable** if and only if*

- *the degree sequence corresponding to L is multigraphic,*
- *half or less than half of the missing connections lie in q_2 blocks, and*
- *half or less than half of the missing connections lie in q_3 blocks.*

◇

Once we have a list we need to try and add the connections in all possible ways.

partial
 C_4^q -marked
pregraph
deficient
vertex

Definition 3.1.12 *A **partial C_4^q -marked pregraph** is a not necessarily connected pregraph P in which all the edges of a given C_4^q -factor are coloured with colour 0 and all other edges with colour 1 and where all vertices have either degree 2 or degree 3.*

*The vertices with degree 2 are called the **deficient vertices** of P .*

◇

$B(P)$

Definition 3.1.13 *Denote by $B(P)$ the block list corresponding to the unique block partition of P .*

\mathcal{B}_L

Given a block list L , denote by \mathcal{B}_L the set of partial C_4^q -marked pregraphs with a block partition isomorphic to L .

◇

A block list corresponds in a trivial way to a partial C_4^q -marked pregraph. We construct a new partial C_4^q -marked pregraph by connecting two deficient vertices. The new partial C_4^q -marked pregraph has two deficient vertices less than the original graph. Once we have a partial C_4^q -marked pregraph with no deficient vertices we have found a C_4^q -marked pregraph. To generate the C_4^q -marked pregraphs we use the principle of closed structures [55][60].

marked
subgraph

Definition 3.1.14 *A **marked subgraph** of a partial C_4^q -marked pregraph P is a subgraph P_s of P such that P_s is a partial C_4^q -marked pregraph and the colours of the edges in P_s is the same as the colours of the edges in P .*

◇

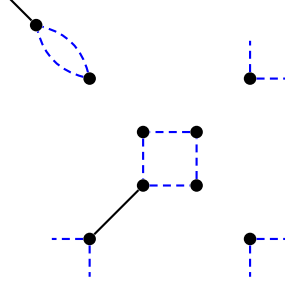


Figure 3.19: A partial C_4^q -marked pregraph that is not closed.

Definition 3.1.15 A partial C_4^q -marked pregraph P' is an **extension** of a partial C_4^q -marked pregraph P if P is a marked subgraph of P' , and P' and P have the same number of vertices. **extension** \diamond

Definition 3.1.16 A partial C_4^q -marked pregraph P is **closed** if for all extensions P_1 and P_2 of P we have that any isomorphism between P_1 and P_2 induces an automorphism of P . **closed** \diamond

The partial C_4^q -marked pregraph P in Figure 3.19 is not closed since the two extensions in Figure 3.20 are isomorphic, but the isomorphism maps the vertex in the bottom right to the vertex in the bottom left and vice versa. Clearly this does not induce an automorphism of P since these vertices have different degrees in P .

The partial C_4^q -marked pregraph corresponding to a block list L is a closed partial C_4^q -marked pregraph since any connection that would create a subgraph that is a block leads to a partial C_4^q -marked pregraph that is not in \mathcal{B}_L .

The advantage of this technique with closed structures is that if the closed partial C_4^q -marked pregraph P has a trivial symmetry group, no two different extensions of P will be isomorphic, and so no isomorphism rejection is needed once a closed graph with trivial symmetry appears in the generation process. Clearly we want to reach such a closed partial C_4^q -marked pregraph as soon as possible during the generation. The following lemma shows a way how a new closed partial C_4^q -marked pregraph can be obtained when starting with a closed partial C_4^q -marked pregraph.

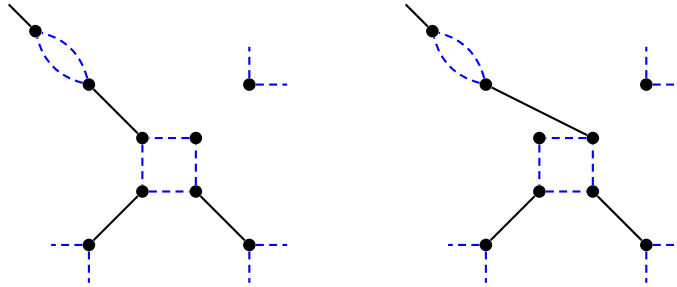


Figure 3.20: Two extensions of the partial C_4^q -marked pregraph in Figure 3.19 which are isomorphic but for which the isomorphism does not induce an automorphism of the original partial C_4^q -marked pregraph.

Lemma 3.1.17 *Let P be a closed partial C_4^q -marked pregraph. Let O be an orbit of deficient vertices under the automorphism group of P . If P' is an extension of P so that O contains no deficient vertices and no edges in $P' \setminus P$ have an empty intersection with O , then P' is also a closed partial C_4^q -marked pregraph.*

Proof: Assume we have two extensions P'_1 and P'_2 of P . We need to prove that if there is an isomorphism σ between P'_1 and P'_2 , this isomorphism induces an automorphism of P' . Since P is closed, we have that σ induces an automorphism of P .

Assume that there is an isomorphism $\sigma : P'_1 \rightarrow P'_2$ that does not induce an automorphism of P' . So there exist vertices x, y in P' , such that x and y are adjacent in P' , and $\sigma(x)$ and $\sigma(y)$ are not adjacent in P' . As P is a subgraph of P' , it cannot be that $\sigma(x)$ and $\sigma(y)$ are adjacent in P , so we have that x and y are non-adjacent in P . This means at least one of the two vertices x and y belongs to O , so assume that $x \in O$. As σ induces an automorphism on P , this means that $\sigma(x) \in O$ and so in P' $\sigma(x)$ is adjacent to another vertex, say z . Since x is adjacent to y in P'_1 , $\sigma(x)$ is adjacent to $\sigma(y)$ in P'_2 , but this contradicts that $\sigma(x)$ is not a deficient vertex in P' .

So we find that for any two extensions P'_1 and P'_2 , there is no isomorphism

between P'_1 and P'_2 that does not induce an automorphism of P' and thus P' is a closed partial C_4^q -marked pregraph. ■

We want to go from closed partial C_4^q -marked pregraphs to closed partial C_4^q -marked pregraphs as 'fast' as possible, that is why we will each time select the smallest orbit of deficient vertices and add connections to that orbit. Once we find a partial C_4^q -marked pregraph with a trivial symmetry, we can stop any isomorphism rejections and just add the remaining connections in all possible ways. We can even do this a bit sooner: it is sufficient that the symmetry group acts trivially on the deficient vertices.

Theorem 3.1.18 *Let P be a closed partial C_4^q -marked pregraph. If the automorphism group of P acts trivially on the deficient vertices of P , then all extensions of P are pairwise non-isomorphic.*

Proof: Let P_1 and P_2 be two different extensions of P . Assume that there is an isomorphism σ between P_1 and P_2 . Since P is closed, the isomorphism σ induces an automorphism of P . Let e be an edge in $P_1 \setminus P$. Both vertices incident to e are deficient vertices in P , and so they are fixed by σ . This implies that the edge e is fixed by σ , so we find that all edges in $P_1 \setminus P$ are fixed by σ , which contradicts that P_1 and P_2 are different extensions. ■

In summary, we use the following algorithm to generate the C_4^q -marked pregraphs with n vertices:

1. Generate all acceptable lists of blocks such that the sum of the orders of the blocks is n .
2. For each list construct the corresponding partial C_4^q -marked pregraph P and recursively repeat the following steps :
 - (a) If P has no deficient vertices: output P and return.

- (b) Compute the automorphism group of P and compute the orbits of deficient vertices of P .
- (c) If $\text{Aut}(P)$ acts trivial on the set of deficient vertices of P , then complete P by adding the remaining connections in all possible ways and output any complete C_4^q -marked pregraph obtained this way.
- (d) Otherwise choose the smallest orbit O and connect all vertices in O to deficient vertices in all valid ways that give non-isomorphic partial C_4^q -marked pregraph and repeat these steps for this new partial C_4^q -marked pregraph.

The fact that this algorithm moves from closed partial C_4^q -marked pregraph to closed partial C_4^q -marked pregraph is not sufficient to guarantee that no pairwise isomorphic structures are output. Although all extensions of a closed partial C_4^q -marked pregraph are pairwise non-isomorphic, it might still be possible that they are isomorphic to extensions of another partial C_4^q -marked pregraph. That this is not the case still needs to be proven.

**strongly
closed**

Definition 3.1.19 *A partial C_4^q -marked pregraph P is **strongly closed** in a set S of partial C_4^q -marked pregraphs, if all partial C_4^q -marked pregraphs in S that contain a subgraph isomorphic to P are extensions of P . \diamond*

Lemma 3.1.20 *The partial C_4^q -marked pregraphs to which the recursive step is applied in the algorithm above, are strongly closed in \mathcal{B}_L , with L the block list that is currently connected.*

Proof: Due to Lemma 3.1.17, the graphs to which the recursive step is applied are closed. It is also clear that the initial partial C_4^q -marked pregraph that corresponds to the block list L without connections is strongly closed in the set \mathcal{B}_L .

What remains to be proven is that if a partial C_4^q -marked pregraph P is strongly closed in $\mathcal{B}_{B(P)}$, O is an orbit of deficient vertices of P under the automorphism group of P and P' is an extension of P such that all vertices in O are no longer deficient and no edges were added that have an empty intersection with O , then P' is also strongly closed in $\mathcal{B}_{B(P)}$.

Given a partial C_4^q -marked pregraph P'' such that $B(P'') = B(P)$ and that P'' contains a subgraph P'_s that is isomorphic to P' . As P' is an extension of P , P'_s also contains a subgraph that is isomorphic to P , and thus P'_s is an extension of P . Since P is closed, we have that the isomorphism between P'_s and P' induces an automorphism of P . Since O is an orbit under the automorphism group of P , O is mapped onto O . This means that both P' and P'_s are extensions of P and for both partial C_4^q -marked pregraphs the same orbit of deficient vertices was chosen in step 2d of the algorithm above. In step 2d only pairwise non-isomorphic partial C_4^q -marked pregraphs are generated, so we find that $P' = P'_s$. So we have that P'' is an extension of P' , which proves that P' is strongly closed in $\mathcal{B}_{B(P)}$. ■

Theorem 3.1.21 *The algorithm above outputs exactly one representative of every isomorphism class of C_4^q -marked pregraph with n vertices.*

Proof: This theorem follows from Lemma 3.1.17 and Lemma 3.1.20, together with the fact that each C_4^q -marked pregraph has a unique block partition. ■

When we construct the partial C_4^q -marked pregraph P corresponding to a list we also construct the automorphism group of P , i.e., we construct a set of generators for the automorphism group based upon the automorphisms of the blocks and the isomorphism of similar blocks. For further computations of the automorphism group we use the program *nauty* [17].

For step 2d we use once again McKay's canonical construction path method (see Subsection 1.4.6). Given a partial C_4^q -marked pregraph P and an orbit O of deficient vertices, we first calculate the orbits of unordered pairs of deficient vertices $\{x, y\}$ such that $\{x, y\} \cap O$ is not empty. For each orbit of unordered pairs we choose one pair in that orbit and connect these vertices if this is a valid connection. There are two reasons why a connection could be invalid: it might create a new block, or it might create a subgraph which does not contain all the vertices and does not contain any deficient vertices. In case this is a valid connection, we still need to verify that

it is the canonical operation (see Subsection 1.4.6) to obtain the resulting partial C_4^q -marked pregraph P' . This is done by labelling each vertex with a 2-tuples (x_1, x_2) . In this tuple x_1 is the label of v in a canonical labelling of P and x_2 is the label of v in a canonical labelling of P' . This operation is accepted if and only if the new connection is in the orbit of connections in P' which have a non-empty intersection with O and for which the vertices have the lexicographically smallest vertex labels. It is often not needed to construct a canonical labelling of P' , since the operation can already be discarded as being not canonical based on the values of x_1 for the vertices.

Table 3.12 (see p.129) gives an overview of the numbers of block lists, the numbers of C_4^q -marked pregraphs and the numbers of C_4^q -markable pregraphs. The numbers of C_4^q -markable pregraphs have been compared to the numbers obtained in the previous chapter. Since the techniques used in both cases are very different this offers an independent test for the implementation.

3.2 Delaney-Dress graphs

Given a Delaney-Dress graph G we can easily construct a C_4^q -marked pregraph P from G by changing the colour of all edges with colour 2 to colour 0. When we want to generate Delaney-Dress graphs from C_4^q -marked pregraphs, then we want to go in the other direction, i.e., we need to assign colours 0 and 2 to the marked quotients of C_4 in the C_4^q -marked pregraph. Clearly the construction above leads to a unique C_4^q -marked pregraph corresponding to a Delaney-Dress graph, and so different C_4^q -marked pregraphs will lead to different Delaney-Dress graphs.

We need to check that different colour assignments do not lead to isomorphic Delaney-Dress graphs. In the cases where this does happen, we only accept one of these isomorphic copies.

A first observation we can make is that if we swap the colours in a quotient of type q_2 or in a quotient of type q_4 , we always get an isomorphic Delaney-Dress graph. We indeed always have the isomorphism that fixes all the vertices and all the edges that are not in that quotient and that interchanges the two edges, resp. semi-edges, in that quotient. This means that we can just choose an arbitrary colouring for these

quotients and can focus the isomorphism rejection on the quotients of type q_1 and the quotients of type q_3 .

Given a partially coloured Delaney-Dress graph D such that the uncoloured subgraphs are quotients of type q_1 and of type q_3 . The set of uncoloured quotients is denoted by U . We can define a bijection between the set of valid colour assignments for D and the set of binary vectors with length $|U|$. Choose a matching (i.e., two non-adjacent edges) in each quotient of type q_1 . For a quotient u of type q_1 we denote this matching by $m(u)$. Label the quotients in U with the numbers 1 to $|U|$.

A colouring c is mapped to a binary vector v_c as follows. The i th coordinate of v_c corresponds to the uncoloured quotient $u \in U$ that has label i . If u is of type q_1 , then the i th coordinate of v_c is equal to 0 if the edges in $m(u)$ receive colour 0, and is equal to 1 if these edges receive colour 2. If u is of type q_3 , then the i th coordinate of v_c is equal to 0 if the semi-edges in u receive colour 0, and is equal to 0 otherwise.

Given an automorphism σ of D and a binary vector v_c corresponding to a colouring c , we can easily construct the binary vector v'_c that corresponds to the colouring c' of D when we would apply σ to the coloured Delaney-Dress graph. The automorphism σ will map a quotient $u \in U$ to another quotient $u' \in U$, and clearly u and u' will be of the same type. In case u is of type q_3 , then the coordinate in v'_c corresponding to u' will have the same value as the coordinate in v_c corresponding to u . In case u is of type q_1 , then we need to check whether $m(u)$ is mapped to $m(u')$ by σ . If this is the case, then the coordinate in v'_c corresponding to u' will have the same value as the coordinate in v_c corresponding to u . Otherwise they will have different values.

This means that we can perform the orbit calculations on the set of binary vectors. We use the *union-find* algorithm on the set of all binary vectors with length $|U|$ to find which coloured Delaney-Dress graphs are isomorphic.

Table 3.13 on page 132 shows an overview of the numbers of Delaney-Dress graphs with up to $n = 30$ vertices. The numbers up to $n = 10$ vertices have been independently verified by Alen Orbančić.

3.3 Delaney-Dress symbols

In this section we will discuss how we can systematically generate Delaney-Dress symbols of equivariant tilings of the Euclidean plane. Although this work is included here, we think that even after this thesis there is still a lot of research that can be done on this topic.

A Delaney-Dress symbol consists of a Delaney-Dress graph together with two functions m_{01} and m_{12} from the vertices to the natural numbers.

Given a coloured Delaney-Dress graph \mathcal{D} . We denote by \mathcal{D}_{ij} the subgraph obtained by excluding all edges that are not coloured with colour i or colour j . The set of connected components of \mathcal{D}_{01} is denoted by F , and the set of connected components of \mathcal{D}_{12} is denoted by V . If \mathcal{D} is part of the Delaney-Dress symbol of an equivariant tiling (T, Γ) then these components will correspond to the orbits of faces, resp. of vertices, of (T, Γ) under Γ (see Section 1.3). The definition of a Delaney-Dress symbol demands the function m_{01} to be constant on elements of F and the function m_{12} to be constant on elements of V . This means that the following functions are well-defined:

$$m_F : F \rightarrow \mathbb{N}; f \mapsto m_F(f) = m_{01}(d) \text{ with } d \in f$$

and

$$m_V : V \rightarrow \mathbb{N}; v \mapsto m_V(v) = m_{12}(d) \text{ with } d \in v.$$

In order that a Delaney-Dress symbol corresponds to an equivariant tiling of the Euclidean plane, the curvature must be 0 (see Theorem 1.3.16). In terms of the newly defined functions this means

$$0 = \sum_{f \in F} \frac{|f|}{m_F(f)} + \sum_{v \in V} \frac{|v|}{m_V(v)} - \frac{|\mathcal{D}|}{2}.$$

In this chapter we will use the crystallographic restriction theorem several times. This theorem has been known since the crystallographic work of René-Just Haüy in 1822. It was described by Vaidyanathaswamy [3, 4] and later rediscovered by many authors [8, 13, 22, 20, 38].

Theorem 3.3.1 (Crystallographic restriction theorem) *The rotational symmetries in the Euclidean plane are 2-fold, 3-fold, 4-fold or 6-fold.*

The following functions are related to the crystallographic restriction:

$$\mathcal{C} : \mathbb{N} \rightarrow \mathbb{N}; n \mapsto \mathcal{C}(n) = \begin{cases} \frac{n}{6} & n \bmod 6 = 0 \\ \frac{n}{4} & n \bmod 4 = 0 \text{ and } n \bmod 3 \neq 0 \\ \frac{n}{3} & n \bmod 3 = 0 \text{ and } n \bmod 2 \neq 0 \\ \frac{n}{2} & n \bmod 2 = 0 \text{ and } n \bmod 3 \neq 0 \text{ and } n \bmod 4 \neq 0 \\ n & \text{all other cases} \end{cases}$$

and

$$\mathcal{C}^\circ : \mathbb{N} \rightarrow \mathbb{N}; n \mapsto \mathcal{C}^\circ(n) = \begin{cases} \frac{n}{3} & n \bmod 6 = 0 \\ \frac{n}{2} & n \bmod 4 = 0 \text{ and } n \bmod 3 \neq 0 \\ \frac{2n}{3} & n \bmod 3 = 0 \text{ and } n \bmod 2 \neq 0 \\ n & n \bmod 2 = 0 \text{ and } n \bmod 3 \neq 0 \text{ and } n \bmod 4 \neq 0 \\ 2n & \text{all other cases} \end{cases}$$

Lemma 3.3.2 *Let $(\mathcal{D}, m_{01}, m_{12})$ be a Delaney-Dress symbol of a tiling T . For any component O of \mathcal{D}_{ij} with $0 \leq i < j \leq 2$ we have that O contains between $\mathcal{C}(m_{ij}(d))$ and $2m_{ij}(d)$ flags with $d \in O$.*

Proof: Since \mathcal{D} is the quotient of the flag graph of T and the components of which O is the quotient contained $2m_{ij}(d)$ flags, we have that the number of flags in O is a divisor of $2m_{ij}(d)$.

As the crystallographic restriction theorem states, the rotational symmetries in the Euclidean plane are restricted to 2-fold, 3-fold, 4-fold, or 6-fold. Mirror symmetries have order 2. This means that the only possible sizes for O are $\frac{2m_{ij}(d)}{2 \cdot 6}$, $\frac{2m_{ij}(d)}{2 \cdot 4}$, $\frac{2m_{ij}(d)}{2 \cdot 3}$, $\frac{2m_{ij}(d)}{2 \cdot 2}$, $\frac{2m_{ij}(d)}{2}$, $\frac{2m_{ij}(d)}{6}$, $\frac{2m_{ij}(d)}{4}$, $\frac{2m_{ij}(d)}{3}$ and $2m_{ij}(d)$ with the extra condition of course that this needs to be a natural number. Note that several of these values are the same, they are only given here to emphasise the origin of these numbers. For given $2m_{ij}(d)$ the value $\mathcal{C}(m_{ij}(d))$ corresponds to the minimum of these values. ■

Lemma 3.3.3 *Let $(\mathcal{D}, m_{01}, m_{12})$ be a Delaney-Dress symbol of an orientable tiling T . For any component O of \mathcal{D}_{ij} with $0 \leq i < j \leq 2$ we have that O contains between $C^\circ(m_{ij}(d))$ and $2m_{ij}(d)$ flags with $d \in O$.*

Proof: Similar to proof of Lemma 3.3.2, but in this case there are no mirror symmetries possible. ■

3.3.1 Enumerating all Delaney-Dress symbols

In this section we will describe an algorithm to generate all Delaney-Dress symbols that have a Delaney-Dress graph with a given number n of vertices. By applying this algorithm for all possible values of n , one can generate all possible Delaney-Dress symbols.

Due to Lemma 3.3.2 and Lemma 3.3.3, we have that in case the component f of F , resp. v of V , is incident to a semi-edge, then the value $m_F(f)$, resp. $m_V(v)$, lies between $|f|$ and $6|f|$, resp. $|v|$ and $6|v|$. Otherwise the value $m_F(f)$, resp. $m_V(v)$, lies between $\frac{|f|}{2}$ and $3|f|$, resp. $\frac{|v|}{2}$ and $3|v|$.

We will describe how we determine all possible functions m_F . The same technique is used to determine all possible functions m_V .

Definition 3.3.4 *Given a set F and functions $m_F : F \rightarrow \mathbb{N} \cap \{\text{undefined}\}$ and $m'_F : F \rightarrow \mathbb{N} \cap \{\text{undefined}\}$, we say that (F, m_F) is isomorphic to (F, m'_F) if and only if there exists an automorphism σ of F such that*

$$\forall f \in F : m_F(f) = m'_F(\sigma(f)).$$

◇

We first define two functions: $\min(f)$ and $\max(f)$. For each f in F we set $\min(f)$ equal to $|f|$ if f contains semi-edges and $\frac{|f|}{2}$ otherwise. For each f in F we set $\max(f)$ equal to $6|f|$ if f contains semi-edges and $3|f|$ otherwise.

We will assign to each component f of F a certificate. This certificate is a n -tuple, where n is the depth of the recursive calls. This means that we first assign a 1-tuple to each component $f \in F$. To do this we calculate the orbits of components

under the automorphism group of \mathcal{D} and then we sort these orbits in descending order according to the size of the components they contain. We then assign to each component the number of its orbit as certificate and apply the following algorithm.

1. Choose a component f which has the lexicographically smallest certificate and has not yet received a value for m_F . We denote by O the orbit to which f belongs. If all components have received a value for m_F , output this graph together with the function m_F .
2. Repeat the following steps until $\min(f) > \max(f)$:
 - (a) Assign to f the value $\min(f)$.
 - (b) Set the value of \min to $\min(f)$ for all components in O .
 - (c) Recalculate the orbits of components of F taking also the already assigned values for m_F into account.
 - (d) Order the orbits in descending order according to the size of the components they contain and by giving priority to orbits for which the value for m_F is assigned, and append the number of each orbit to the certificate of each component in that orbit.
 - (e) Recursively apply this algorithm to the Delaney-Dress graph together with the new m_F function.
 - (f) Increment $\min(f)$ by one.

Theorem 3.3.5 *All structures output by the algorithm above are pairwise non-isomorphic.*

Proof: Let F be a set of components and let m_F and m'_F be two different functions which were generated by the algorithm above. Assume that there is an isomorphism σ from (F, m_F) to (F, m'_F) . This implies that σ is an automorphism of F and that the following equality holds for all components $f \in F$:

$$m_F(f) = m'_F(\sigma(f)). \quad (3.1)$$

It is clear that σ cannot be the identity, so there exists a component $f \in F$ such that $m_F(f) \neq m'_F(f)$. Let f_1 be the first component that receives different values for m_F and m'_F by the algorithm. Without loss of generality, we can assume that

$$m_F(f_1) < m'_F(f_1). \quad (3.2)$$

We denote $\sigma(f_1)$ by f'_1 .

When the algorithm assigns the value of m'_F for f_1 , we have that f_1 and f'_1 are in the same orbit, and thus

$$m'_F(f_1) \leq m'_F(f'_1). \quad (3.3)$$

Combining these results

$$m_F(f_1) \stackrel{3.1}{=} m'_F(f'_1) \stackrel{3.3}{\geq} m'_F(f_1) \stackrel{3.2}{>} m_F(f_1).$$

This contradiction proves that the algorithm does not output isomorphic structures. ■

The algorithm also tries all possible values, so all valid structures are output by the algorithm.

Since the orbits after assigning a value for m_F are subsets of the orbits before that assignment, we do not need to recalculate the orbits once the automorphism group acts trivially on the components which have not yet received a value for m_F .

If a graph is output by this algorithm, we then apply a similar algorithm to calculate the value of m_V . If a graph is then output by that algorithm, we can calculate the curvature to determine if the assignment is valid for a tiling of the Euclidean plane.

There is at the moment no independent implementation to verify this algorithm. The results of this generator have been used to generate tilings which satisfy certain conditions. This provide some ways to test the implementation of this algorithm as will be described in the next subsection.

Property	Description	Default
$m_{\mathcal{F}}$	The minimum face size in the tiling	3
$M_{\mathcal{F}}$	The maximum face size in the tiling	$6 * \text{MAXN}$
$m_{\mathcal{V}}$	The minimum vertex degree in the tiling	3
$M_{\mathcal{V}}$	The maximum vertex degree in the tiling	$6 * \text{MAXN}$
$m_{ F }$	The minimum number of face orbits in the tiling	1
$M_{ F }$	The maximum number of face orbits in the tiling	MAXN
$m_{ V }$	The minimum number of vertex orbits in the tiling	1
$M_{ V }$	The maximum number of vertex orbits in the tiling	MAXN

Table 3.6: *The properties of tilings which we want to be able to specify and the corresponding default values. The value MAXN is implementation specific.*

3.3.2 Restricting the tilings

In this subsection we will discuss how the generation process can be bounded in order to generate Delaney-Dress symbols for equivariant tilings with specified properties.

The properties of tilings which we consider are listed in Table 3.6 together with their default values. The minima can only be increased and the maxima can only be decreased. In principle the algorithm would work without the maxima having a default value, or having infinity as the default value, but in a program a maximum is enforced by limitations of the computer. To prevent any confusion we will refer to the vertices of the Delaney-Dress graph as flags for the rest of this section.

The generation algorithm for Delaney-Dress symbols works in different phases:

1. Generate all acceptable lists of blocks.
2. Given a list of blocks L , generate all partial C_4^q -marked pregraphs in \mathbb{B}_L .
3. Given a partial C_4^q -marked pregraph P , generate all Delaney-Dress graphs based on P .
4. Given a Delaney-Dress graph D , generate all Delaney-Dress symbols that have

D as Delaney-Dress graph.

In each phase we will try to discard as many structures as possible that do not lead to a Delaney-Dress symbol that satisfies the restrictions.

3.3.2.1 Refining the restrictions

The restrictions we allow to be specified are not all independent of each other. Before we start to bound the generation process we first try to refine the restrictions given. Using the curvature we have

$$\sum_{d \in \mathcal{D}} \left(\frac{1}{M_{\mathcal{F}}} + \frac{1}{M_{\mathcal{V}}} - \frac{1}{2} \right) \leq \sum_{d \in \mathcal{D}} \left(\frac{1}{m_{01}(d)} + \frac{1}{m_{12}(d)} - \frac{1}{2} \right) \leq \sum_{d \in \mathcal{D}} \left(\frac{1}{m_{\mathcal{F}}} + \frac{1}{m_{\mathcal{V}}} - \frac{1}{2} \right),$$

which is equivalent to

$$|\mathcal{D}| \left(\frac{1}{M_{\mathcal{F}}} + \frac{1}{M_{\mathcal{V}}} - \frac{1}{2} \right) \leq \sum_{d \in \mathcal{D}} \left(\frac{1}{m_{01}(d)} + \frac{1}{m_{12}(d)} - \frac{1}{2} \right) \leq |\mathcal{D}| \left(\frac{1}{m_{\mathcal{F}}} + \frac{1}{m_{\mathcal{V}}} - \frac{1}{2} \right).$$

Since $|\mathcal{D}|$ is always strictly positive and the curvature is equal to 0, this gives some simple restrictions on the minimum and maximum face sizes and vertex degrees:

$$0 \leq 2m_{\mathcal{V}} + 2m_{\mathcal{F}} - m_{\mathcal{V}}m_{\mathcal{F}}, \quad (3.4)$$

$$0 \geq 2M_{\mathcal{V}} + 2M_{\mathcal{F}} - M_{\mathcal{V}}M_{\mathcal{F}}. \quad (3.5)$$

An overview of which combinations are allowed by these two inequalities, is given in Table 3.7 and Table 3.8.

3.3.2.2 Possible orders of Delaney-Dress graphs

As described above, the generation of the Delaney-Dress symbols is done by first generating the possible Delaney-Dress graphs and then looking for the functions m_{01} and m_{12} . Limiting the number of Delaney-Dress graphs is therefore an important part of making the generation of the Delaney-Dress symbols as efficient as possible.

	3	4	5	6	7
3	3	2	1	0	-1
4	2	0	-2	-4	-6
5	1	-2	-5	-8	-11
6	0	-4	-8	-12	-16
7	-1	-6	-11	-16	-21

Table 3.7: An overview of the inequality 3.4 calculated for several different combinations of minima. The shaded cells correspond to combinations of minima that are allowed.

	3	4	5	6	7
3	3	2	1	0	-1
4	2	0	-2	-4	-6
5	1	-2	-5	-8	-11
6	0	-4	-8	-12	-16
7	-1	-6	-11	-16	-21

Table 3.8: An overview of the inequality 3.5 calculated for several different combinations of maxima. The shaded cells correspond to combinations of maxima that are not allowed.

A first thing we can try to bound is the size of possible Delaney-Dress graphs. The formula for the curvature is

$$K = \sum_{d \in \mathcal{D}} \left(\frac{1}{m_{01}(d)} + \frac{1}{m_{12}(d)} - \frac{1}{2} \right) = 0 \quad (3.6)$$

Due to Lemma 3.3.2 and since for each $d \in \mathcal{D}$ it is so that $m_{12}(d) \leq m_{\mathcal{V}}$, we find that

$$2|F| + |\mathcal{D}| \left(\frac{1}{m_{\mathcal{V}}} - \frac{1}{2} \right) \geq 0 \quad (3.7)$$

which can be rewritten as

$$|\mathcal{D}| \leq \left(\frac{4m_{\mathcal{V}}}{m_{\mathcal{V}} - 2} \right) |F|.$$

This implies the following bound for the number of flags in the Delaney-Dress graph:

$$|\mathcal{D}| \leq \left(\frac{4m_{\mathcal{V}}}{m_{\mathcal{V}} - 2} \right) M_{|F|}.$$

In the formula for the curvature the roles of m_{01} and m_{12} are interchangeable. Using a similar calculation as above but interchanging vertices by faces, we get

$$|\mathcal{D}| \leq \left(\frac{4m_{\mathcal{F}}}{m_{\mathcal{F}} - 2} \right) M_{|V|}.$$

If we use the upper bound from Lemma 3.3.2 for the number of flags for both the vertex orbits and face orbits and substitute these into the curvature formula, we get

$$|\mathcal{D}| \leq 4(|V| + |F|). \quad (3.8)$$

This formula implies the following upper bound on the number of flags in a Delaney-Dress graph:

$$|\mathcal{D}| \leq 4(M_{|V|} + M_{|F|}).$$

The lower bound of Lemma 3.3.2 gives us a minimum amount of flags that each face orbit contributes to the Delaney-Dress graph. By multiplying this by the minimum of face orbits required, we can find the following lower bound for the number of flags:

$$|\mathcal{D}| \geq m_{|F|} \min\{\mathcal{C}(n) | m_{\mathcal{F}} \leq n \leq M_{\mathcal{F}}\}.$$

In case of orientable tilings we can use Lemma 3.3.3 and improve this bound to

$$|\mathcal{D}| \geq m_{|F|} \min\{\mathcal{C}^\circ(n) | m_{\mathcal{F}} \leq n \leq M_{\mathcal{F}}\}.$$

Analogously we can find the following lower bound for the number of flags:

$$|\mathcal{D}| \geq m_{|V|} \min\{\mathcal{C}(n) | m_{\mathcal{V}} \leq n \leq M_{\mathcal{V}}\}.$$

In case of orientable tilings we can improve this bound to

$$|\mathcal{D}| \geq m_{|V|} \min\{\mathcal{C}^\circ(n) | m_{\mathcal{V}} \leq n \leq M_{\mathcal{V}}\}.$$

If we use the upper bound of Lemma 3.3.2 and the required maxima for the number of orbits, we can make a similar argumentation and get the following upper bounds on the number of flags:

$$|\mathcal{D}| \leq 2M_{|F|}M_{\mathcal{F}}$$

and

$$|\mathcal{D}| \leq 2M_{|V|}M_{\mathcal{V}}.$$

Another restriction we might want to specify for a tiling is a list of required face sizes and vertex degrees together with a required minimum multiplicity, and a list of forbidden face sizes and vertex degrees. Since several face orbits, resp. vertex orbits, might have the same size, resp. degree, these lists will be represented by multisets. For the forbidden face sizes and vertex degrees of course a set suffices. We denote these multisets as follows:

- $\mathcal{R}_{\mathcal{F}}$ Multiset of required face sizes
- $\mathcal{R}_{\mathcal{V}}$ Multiset of required vertex degrees
- $\mathcal{U}_{\mathcal{F}}$ Set of forbidden (unwanted) face sizes
- $\mathcal{U}_{\mathcal{V}}$ Set of forbidden (unwanted) vertex degrees

Given these multisets and sets we can rewrite the last six bounds:

$$|\mathcal{D}| \geq \sum_{f \in \mathcal{R}_{\mathcal{F}}} \mathcal{C}(f) + (m_{|F|} - |\mathcal{R}_{\mathcal{F}}|) \min\{\mathcal{C}(n) | m_{\mathcal{F}} \leq n \leq M_{\mathcal{F}} \wedge n \notin \mathcal{U}_{\mathcal{F}}\}$$

$$|\mathcal{D}| \geq \sum_{v \in \mathcal{R}_V} \mathcal{C}(v) + (m_{|V|} - |\mathcal{R}_V|) \min\{\mathcal{C}(n) | m_V \leq n \leq M_V \wedge n \notin \mathcal{U}_V\}$$

$$|\mathcal{D}| \geq \sum_{f \in \mathcal{R}_F} \mathcal{C}^\circ(f) + (m_{|F|} - |\mathcal{R}_F|) \min\{\mathcal{C}^\circ(n) | m_F \leq n \leq M_F \wedge n \notin \mathcal{U}_F\}$$

$$|\mathcal{D}| \geq \sum_{v \in \mathcal{R}_V} \mathcal{C}^\circ(v) + (m_{|V|} - |\mathcal{R}_V|) \min\{\mathcal{C}^\circ(n) | m_V \leq n \leq M_V \wedge n \notin \mathcal{U}_V\}$$

$$|\mathcal{D}| \leq 2 \left(\sum_{f \in \mathcal{R}_F} f \right) + 2(M_{|F|} - |\mathcal{R}_F|)M_F$$

$$|\mathcal{D}| \leq 2 \left(\sum_{v \in \mathcal{R}_V} v \right) + 2(M_{|V|} - |\mathcal{R}_V|)M_V$$

There are several classes of tilings that have been extensively studied and of course we want this generator to be able to generate these specific classes. The class of tile- N -transitive tilings can be generated by setting $m_{|F|} = M_{|F|} = N$. Another class of tilings which has received a lot of attention is the class of heaven-and-hell tilings.

heaven-and-hell tiling
regular heaven-and-hell tiling
semi-regular heaven-and-hell tiling

Definition 3.3.6 *A tiling is a **heaven-and-hell tiling** if its faces can be coloured black and white such that no two faces of the same colour share an edge.*

*A **regular heaven-and-hell tiling** is a heaven-and-hell tiling which has exactly two orbits of faces if we add the restriction that faces with different colours cannot be mapped to each other.*

*A **semi-regular heaven-and-hell tiling** is a heaven-and-hell tiling of which the faces can be coloured black and white such that there is exactly one orbit of black faces if we add the restriction that faces with different colours cannot be mapped to each other. \diamond*

These classes of tilings got their name from the famous pictures of M.C. Escher.

In [23] an upper bound for the size of the Delaney-Dress graph in case of semi-regular heaven-and-hell tilings is given. This upper bound is in some cases stricter than the upper bounds we have obtained until this point. This bound is also based on the curvature. If a tiling is semi-regular and heaven-and-hell, the implication for

the Delaney-Dress graph is that we can partition the flags into two equal-sized sets \mathcal{D}_B and \mathcal{D}_W such that there is no σ_2 -edge from a flag in \mathcal{D}_B to a flag in \mathcal{D}_W .

$$0 = \sum_{d \in \mathcal{D}} \left(\frac{1}{m_{01}(d)} + \frac{1}{m_{12}(d)} - \frac{1}{2} \right) =$$

$$\sum_{d \in \mathcal{D}_B} \left(\frac{1}{m_{01}(d)} \right) + \sum_{d \in \mathcal{D}_W} \left(\frac{1}{m_{01}(d)} \right) + \sum_{d \in \mathcal{D}} \left(\frac{1}{m_{12}(d)} - \frac{1}{2} \right)$$

Using the fact that there is only one orbit of black faces, we get

$$0 \leq 2 + \frac{|\mathcal{D}_W|}{m_{\mathcal{F}}} + \frac{|\mathcal{D}|}{m_{\mathcal{V}}} - \frac{|\mathcal{D}|}{2},$$

and because $|\mathcal{D}_W| = |\mathcal{D}_B| = \frac{|\mathcal{D}|}{2}$

$$0 \leq 2 + |\mathcal{D}| \left(\frac{m_{\mathcal{V}} + 2m_{\mathcal{F}} - m_{\mathcal{V}}m_{\mathcal{F}}}{2m_{\mathcal{V}}m_{\mathcal{F}}} \right).$$

The smallest possible value for $m_{\mathcal{F}}$ is 3 and for $m_{\mathcal{V}}$ is 4, so $m_{\mathcal{V}} + 2m_{\mathcal{F}} - m_{\mathcal{V}}m_{\mathcal{F}}$ will always be negative and we find

$$|\mathcal{D}| \leq \frac{4m_{\mathcal{V}}m_{\mathcal{F}}}{m_{\mathcal{V}}m_{\mathcal{F}} - m_{\mathcal{V}} - 2m_{\mathcal{F}}}.$$

3.3.2.3 Rejecting lists of blocks

Once we have the possible orders of valid Delaney-Dress graphs the first step in the generation process is to generate lists of blocks for each of these orders. In this section we will be looking at when we can decide that a lists of blocks can never lead to a Delaney-Dress graph that can be used for a valid Delaney-Dress symbol given the restrictions which were fixed earlier. Table 3.9 lists some of the properties of a list of blocks which we will use to reject some of the lists. The value S_1 is already known during this phase since no semi-edges are added while connecting the list of blocks to form a C_4^g -marked pregraph.

A PC block or a LPC block with parameter p (see Figure 3.13) always corresponds to a component of \mathcal{D}_{01} and a component of \mathcal{D}_{12} both having more than $2p$ flags

Property	Description
S_1	The number of semi-edges with colour 1.
Q_3	The number of marked C_4 -quotients of type q_3 .
Q_4	The number of marked C_4 -quotients of type q_4 .

Table 3.9: Some properties of lists of blocks which are useful to reject such a list.

(more since the endpoints of a block may not be connected to each other). Due to Lemma 3.3.2 such a block is not possible in a tiling which has $M_{\mathcal{F}} \leq p$ or $M_{\mathcal{V}} \leq p$. For a LPC block with parameter p we can even improve this to $M_{\mathcal{F}} \leq 2p$ or $M_{\mathcal{V}} \leq 2p$.

Given a Delaney-Dress graph \mathcal{D} , the components of \mathcal{D}_{ij} with $0 \leq i < j \leq 2$ are either cycles or paths where each end point is incident to a semi-edge or a single vertex which is incident to two semi-edges. The semi-edges in a component of \mathcal{D}_{ij} are called the end points of that component and each component either has zero or two endpoints. Each semi-edge with colour 1 is an endpoint of a component of \mathcal{D}_{01} and of a component of \mathcal{D}_{12} . The semi-edges in each quotient of type q_4 also are the endpoints respectively of a component of \mathcal{D}_{01} and of a component of \mathcal{D}_{12} . This gives us the following two inequalities which is valid for any Delaney-Dress graph:

$$|\mathcal{V}| \geq \left\lceil \frac{S_1 + Q_4}{2} \right\rceil$$

$$|\mathcal{F}| \geq \left\lceil \frac{S_1 + Q_4}{2} \right\rceil$$

This gives us the following restrictions for $M_{|\mathcal{V}|}$ and $M_{|\mathcal{F}|}$:

$$M_{|\mathcal{V}|} \geq \left\lceil \frac{S_1 + Q_4}{2} \right\rceil$$

$$M_{|\mathcal{F}|} \geq \left\lceil \frac{S_1 + Q_4}{2} \right\rceil$$

and $\left\lceil \frac{S_1 + Q_4}{2} \right\rceil$ is a possible improvement for both $m_{|\mathcal{F}|}$ and $m_{|\mathcal{V}|}$.

Each quotient of type q_3 contains two semi-edges and two vertices. Per quotient these two semi-edges will always receive the same colour: either 0 or 2. This means that at this point we cannot decide whether these semi-edges are the endpoint of a component of \mathcal{D}_{01} or of a component of \mathcal{D}_{12} , but we do know that together all these semi-edges are the endpoints of at least Q_3 components of \mathcal{D}_{01} and \mathcal{D}_{12} , so the following inequality holds for any Delaney-Dress graph:

$$2 \left\lceil \frac{S_1 + Q_4}{2} \right\rceil + Q_3 \leq |\mathcal{V}| + |\mathcal{F}|,$$

which gives us the following restriction

$$2 \left\lceil \frac{S_1 + Q_4}{2} \right\rceil + Q_3 \leq M_{|\mathcal{V}|} + M_{|\mathcal{F}|}.$$

In case we only want to generate Delaney-Dress symbols for orientable tilings, we only need bipartite Delaney-Dress graphs without semi-edges. So we can exclude any lists of blocks that contain a block that has a semi-edge in it or has an odd cycle.

3.3.2.4 Rejecting Delaney-Dress graphs

It is important to avoid Delaney-Dress graphs which will not be used, since trying to find the possible functions m_{01} and m_{12} for each Delaney-Dress graph is an expensive part of the algorithm. In Table 3.10 we see that trying to reject Delaney-Dress graphs as soon as possible is a very useful thing to do: only a very small percentage of Delaney-Dress graphs appear in Delaney-Dress symbols of tilings of the Euclidean plane, and probably even less will satisfy the specified restrictions.

When looking for the functions m_{01} and m_{12} we already have more information at hand and we can improve the bounds to reduce the number of functions which need to be checked. Once we have a Delaney-Dress graph the number of \mathcal{D}_{ij} -components with $0 \leq i < j \leq 2$ and their respective orders are known.

We can, e.g., revisit the inequality 3.7:

$$2|F| + \frac{|\mathcal{D}|}{m_{\mathcal{V}}} - \frac{|\mathcal{D}|}{2} \geq 0.$$

n	used	unused	ratio used
1	1	0	100.00%
2	7	0	100.00%
3	3	0	100.00%
4	20	2	90.91%
5	7	6	53.85%
6	35	35	50.00%
7	18	49	26.87%
8	90	225	28.57%
9	63	330	16.03%
10	163	1414	10.34%
11	161	2354	6.40%
12	452	9028	4.77%
13	436	16769	2.53%
14	1089	60505	1.77%
15	1323	122630	1.07%
16	2997	430033	0.69%
17	3747	927982	0.40%
18	8048	3188793	0.25%

Table 3.10: *The number of Delaney-Dress graphs that appear in Delaney-Dress symbols of a tiling of the Euclidean plane and those that do not appear in such symbols.*

which can be rewritten as

$$|\mathcal{D}| \geq \left(\frac{|\mathcal{D}|}{2} - 2|F| \right) m_{\mathcal{V}}.$$

In case

$$\left(\frac{|\mathcal{D}|}{2} - 2|F| \right) m_{\mathcal{V}} > |\mathcal{D}| \quad (3.9)$$

this gives us that the Delaney-Dress graph will not lead to a valid Delaney-Dress symbol. By interchanging faces and vertices in the previous formulas we find that a Delaney-Dress graph with

$$\left(\frac{|\mathcal{D}|}{2} - 2|V| \right) m_{\mathcal{F}} > |\mathcal{D}| \quad (3.10)$$

will not lead to a valid Delaney-Dress symbol.

Another useful formula is inequality 3.8:

$$|\mathcal{D}| \leq 4(|V| + |F|).$$

This formula does not give any improvements on the bounds we impose on the face sizes and vertex degrees, but it does give us a criterion to exclude Delaney-Dress graphs which have too few \mathcal{D}_{01} - and \mathcal{D}_{12} -components for their order, i.e., we can reject a Delaney-Dress graph in case

$$|\mathcal{D}| > 4(|V| + |F|). \quad (3.11)$$

In Table 3.11 we give an overview of how many Delaney-Dress graphs can be rejected based upon 3.9, 3.10 and 3.11.

3.4 Results

Table 3.12 shows the number of block lists, C_4^q -marked pregraphs and C_4^q -markable pregraphs with a given number of vertices. Table 3.13 shows the number of Delaney-Dress graphs with a given number of vertices. Table 3.14 gives the number of Delaney-Dress symbols of the Euclidean plane with a Delaney-Dress graph with a given number of vertices.

n	only (3.9)	only (3.10)	only (3.11)	only (3.9) and (3.10)	only (3.9) and (3.11)	only (3.10) and (3.11)	(3.9), (3.10) and (3.11)	Total
9	0.000%	0.000%	0.031%	0.000%	0.000%	0.000%	0.000%	0.031%
10	0.000%	0.000%	0.040%	0.000%	0.000%	0.000%	0.000%	0.040%
11	0.000%	0.000%	0.014%	0.000%	0.000%	0.000%	0.000%	0.014%
12	0.000%	0.000%	0.021%	0.000%	0.000%	0.000%	0.000%	0.021%
13	0.048%	0.048%	0.000%	0.000%	0.025%	0.025%	0.008%	0.153%
14	0.051%	0.051%	0.000%	0.000%	0.027%	0.027%	0.010%	0.166%
15	0.034%	0.034%	0.000%	0.000%	0.015%	0.015%	0.004%	0.101%
16	0.037%	0.037%	0.000%	0.000%	0.016%	0.016%	0.006%	0.113%
17	0.013%	0.013%	0.042%	0.000%	0.020%	0.020%	0.002%	0.109%
18	0.014%	0.014%	0.040%	0.000%	0.023%	0.023%	0.003%	0.117%
19	0.010%	0.010%	0.027%	0.000%	0.012%	0.012%	0.001%	0.072%
20	0.011%	0.011%	0.027%	0.000%	0.015%	0.015%	0.002%	0.080%
21	0.003%	0.003%	0.086%	0.000%	0.012%	0.012%	0.001%	0.117%
22	0.004%	0.004%	0.083%	0.000%	0.014%	0.014%	0.001%	0.120%
23	0.003%	0.003%	0.060%	0.000%	0.008%	0.008%	0.000%	0.081%
24	0.003%	0.003%	0.059%	0.000%	0.010%	0.010%	0.001%	0.085%
25	0.026%	0.026%	0.044%	0.000%	0.042%	0.042%	0.010%	0.190%
26	0.026%	0.026%	0.041%	0.000%	0.043%	0.043%	0.012%	0.191%

Table 3.11: *The percentage of Delaney-Dress graphs that satisfy the conditions 3.9, 3.10 and/or 3.11 and thus can be rejected. On up to 8 vertices, no Delaney-Dress graph satisfies any of these conditions.*

Table 3.12: An overview of the number of block lists, the number of C_4^q -marked pregraphs and the number of C_4^q -markable pregraphs with n vertices. For each column the time needed to generate those structures using the program `ddgraphs` is given. For the C_4^q -markable pregraphs also the time needed by `pregraphs` is given. All timings were done on a 2.40 GHz Intel Xeon.

n	lists	C_4^q -marked		C_4^q -markable		pregraphs	
		ddgraphs	time	ddgraphs	time	ddgraphs	time
1	1	1	0.0s	1	0.0s	1	0.0s
2	5	5	0.0s	5	0.0s	3	0.0s
3	2	2	0.0s	2	0.0s	2	0.0s
4	13	13	0.0s	13	0.0s	9	0.0s
5	7	7	0.0s	7	0.0s	7	0.0s
6	31	31	0.0s	31	0.0s	29	0.0s
7	25	27	0.0s	27	0.0s	27	0.0s
8	103	109	0.0s	109	0.0s	105	0.0s
9	86	118	0.0s	118	0.0s	118	0.0s
10	311	394	0.0s	394	0.0s	392	0.1s
11	260	546	0.0s	546	0.0s	546	0.3s
12	938	1726	0.0s	1726	0.0s	1722	1.3s
13	763	2701	0.0s	2701	0.1s	2701	5.2s
14	2521	7955	0.0s	7955	0.3s	7953	22.0s

Continued on next page

Table 3.12: An overview of the number of block lists, the number of C_4^q -marked pregraphs and the number of C_4^q -markable pregraphs with n vertices. (Continued)

n	lists	C_4^q -marked		C_4^q -markable		pregraphs
		time	ddgraphs	time	ddgraphs	
15	1968	0.0s	13966	0.4s	13966	94.8s
16	6776	0.0s	40039	1.4s	40035	420.5s
17	5171	0.0s	75341	2.3s	75341	1903.5s
18	16557	0.0s	210765	8.1s	210763	8850.1s
19	12321	0.0s	420422	13.9s	420422	41812.1s
20	40622	0.1s	1162196	46.5s	1162192	201745.4s
21	29843	0.1s	2419060	86.8s	2419060	86.7s
22	93166	0.2s	6626610	273.8s	6626608	273.7s
23	67345	0.2s	14292180	551.5s	14292180	551.9s
24	213822	0.5s	38958571	1694.1s	38958567	1704.0s
25	153388	0.5s	86488183	3583.6s	86488183	3586.2s
26	467050	1.2s	235004260	10709.5s	235004258	10714.7s
27	331411	1.2s	534796010	23645.3s	534796010	23619.7s
28	1018009	3.0s	1450990715	69131.1s	1450990711	69251.9s
29	719250	2.9s	3373088492	157144.9s	3373088492	157167.0s
30	2136996	6.8s	9147869420	455900.9s	9147869418	455606.1s

The program `ddgraphs` has been tested by comparing the results to some known enumerations of tilings.

There are 93 equivariant tile-transitive tilings of the Euclidean plane (see e.g., [34, 33]). The program `ddgraphs` can be used to generate these tilings by setting the maximum number of face orbits to 1. Based on this restriction, the program then tries to refine the restrictions and finally starts the generation with the restrictions in Table 3.15.

There are 1270 equivariant tile-2-transitive tilings of the Euclidean plane (see e.g., [31]). The program `ddgraphs` can be used to generate these tilings by setting the minimum and maximum number of face orbits to 2. Based on this restriction, the program then tries to refine the restrictions and finally starts the generation with the restrictions in Table 3.16.

There are 30 equivariant edge-transitive tilings of the Euclidean plane [15]. This number also agrees with the results obtained by the program `ddgraphs`.

In [24], the 37 minimal, non-transitive equivariant tilings of the Euclidean plane are enumerated. The program `ddgraphs` can be used to generate these tilings by setting the maximum and minimum number of face, vertex and edge orbits to 2.

In [30], the 69 tile-2-transitive tilings of the plane with all faces quadrangles and all vertices of degree 4 are enumerated. Also for this class the number of tilings coincide with the number of tilings as generated by the program `ddgraphs`.

3.5 Future work

As was shown in Table 3.10, only a very small percentage of Delaney-Dress graphs are used in the Delaney-Dress symbols of the Euclidean plane. It might be possible to discover structure in the set of Delaney-Dress graphs that are used and those that are not. This could be possible at different levels, but the most promising moments seem to be after the lists are constructed and after all colours are assigned.

Another aspect where improvement might be possible is in the refinement of the parameters. It appears that the effective maximum vertex degree and/or face size is very often smaller than the one determined at the start of the generation. In Ta-

n	Delaney-Dress graphs	time	rate
1	1	0.0s	
2	7	0.0s	
3	3	0.0s	
4	22	0.0s	
5	13	0.0s	
6	70	0.0s	
7	67	0.0s	
8	315	0.0s	
9	393	0.0s	
10	1577	0.0s	
11	2515	0.0s	
12	9480	0.1s	94800.00/s
13	17205	0.1s	172050.00/s
14	61594	0.3s	205313.33/s
15	123953	0.4s	309882.50/s
16	433030	1.6s	270643.75/s
17	931729	2.5s	372691.60/s
18	3196841	9.1s	351301.21/s
19	7258011	16.3s	445276.75/s
20	24630262	55.0s	447822.95/s
21	58309071	105.9s	550605.01/s
22	196266434	345.5s	568064.93/s
23	481330615	722.2s	666478.28/s
24	1610942856	2329.2s	691629.25/s
25	4071117829	5184.9s	785187.34/s
26	13569014653	16422.1s	826265.50/s
27	35202390477	38273.5s	919758.85/s
28	116994675348	121796.3s	960576.60/s
29	310624700725	295889.0s	1049801.45/s
30	1030455432427	949823.0s	1084892.06/s

Table 3.13: An overview of the number of Delaney-Dress graphs and the time needed by `ddgraphs` to generate these graphs when run on a 2.40 GHz Intel Xeon.

n	Delaney-Dress symbols	time
1	3	0.0s
2	15	0.0s
3	8	0.0s
4	37	0.0s
5	15	0.0s
6	86	0.0s
7	64	0.0s
8	217	0.2s
9	185	0.5s
10	527	3.8s
11	506	13.0s
12	1597	95.1s
13	1575	360.4s
14	4227	2531.5s
15	4532	10383.8s
16	12078	70331.9s
17	13105	304083.2s
18	34250	1994897.8s

Table 3.14: An overview of the number of Delaney-Dress symbols of the Euclidean plane and the time needed by `ddgraphs` to generate these symbols when run on a 2.40 GHz Intel Xeon.

	Calculated	Actual
$m_{ \mathcal{D} }$	1	1
$M_{ \mathcal{D} }$	12	12
$m_{\mathcal{F}}$	3	3
$M_{\mathcal{F}}$	144	6
$m_{\mathcal{V}}$	3	3
$M_{\mathcal{V}}$	144	12
$m_{ \mathcal{V} }$	1	1
$M_{ \mathcal{V} }$	12	4

Table 3.15: *The refinement on the parameters as calculated by `ddgraphs` in order to generate the tile-transitive tilings of the plane*

	Calculated	Actual
$m_{ \mathcal{D} }$	2	2
$M_{ \mathcal{D} }$	24	24
$m_{\mathcal{F}}$	3	3
$M_{\mathcal{F}}$	276	24
$m_{\mathcal{V}}$	3	3
$M_{\mathcal{V}}$	288	24
$m_{ \mathcal{V} }$	1	1
$M_{ \mathcal{V} }$	24	6

Table 3.16: *The refinement on the parameters as calculated by `ddgraphs` in order to generate the tile-2-transitive tilings of the plane*

ble 3.15, for instance, one can see a comparison of the values calculated for the generation of the tile-transitive tilings and the actual values. Table 3.16 shows the same values for the generation of the tile-2-transitive tilings. Since the calculation of the functions m_{01} and m_{12} is very time consuming and happens a lot, any bound that limits the number of calculations per graph can improve the speed of the program enormously.

At the moment it is not possible to limit the generation of Delaney-Dress symbols to symbols of heaven-and-hell tilings of the Euclidean plane. It is however the long-term intention to allow the inclusion of this restriction, and some preliminary versions of bounding criteria are already implemented.

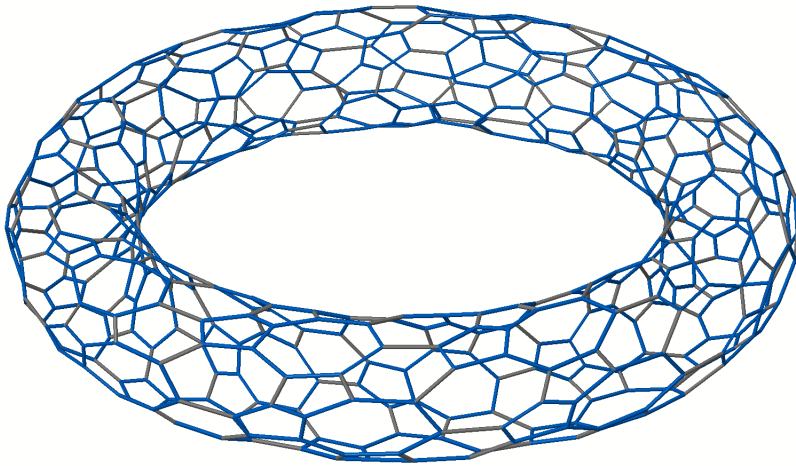
Face-, edge- and/or vertex-colourable tilings are other classes of tilings that we eventually want the generator to be able to construct, but there is still some work that needs to be done for these cases.

Part II

Generation of chemical graphs

4

Azulenoids



Jmol

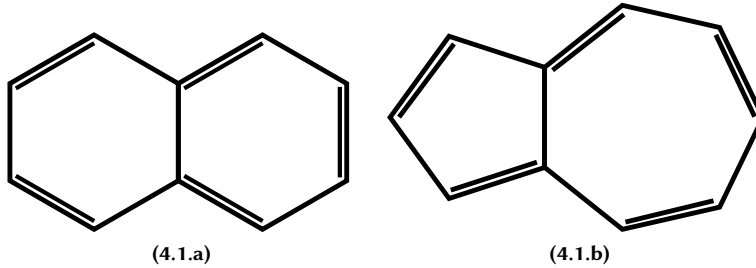


Figure 4.1: The structures of naphthalene (a) and azulene (b).

The research described in this chapter was joint work with Edward Kirby and Olaf Delgado-Friedrichs. It is described in [56].

Azulene ($C_{10}H_8$) is a molecule that is an isomer of naphthalene. It consists of a five-ring and a seven-ring that share two carbon atoms and a bond. It is not yet known how many variations of “graphite-like” networks of azulenes are theoretically possible. This research gives a partial answer to this question: we will give an enumeration of such networks with certain conditions on the symmetry of the azulenes in the network. A fullerene-style network of azulenes, also called an **azulenoid**, will be modeled by a tiling of the Euclidean plane, where the atoms correspond to the vertices, the bonds correspond to the edges and the rings correspond to the faces.

azulenoid

4.1 Definitions

azulene
graph

Definition 4.1.1 An **azulene graph** is a graph isomorphic to the graph shown in Figure 4.2. \diamond

azulenoid
tiling

Definition 4.1.2 An **azulenoid tiling** T is a tiling of the Euclidean plane \mathbb{E}^2 such that the skeleton graph S of T is a cubic graph and there exists a partition \mathcal{P} of the vertices of S such that for each $P \in \mathcal{P}$ the graph S_P induced by P in S is isomorphic to the azulene graph and the 5- and 7-cycle in S_P are facial cycles. Such a partition \mathcal{P} is called an **azulenic set** of the azulenoid tiling T . \diamond

azulenic set

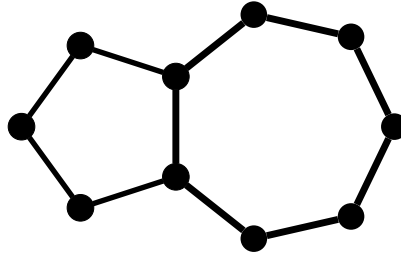


Figure 4.2: The azulene graph.

Owing to the large number of possible azulenoids we needed to impose an extra restriction on the generated set of azulenoids. Our choice was to first enumerate only those equivariant azulenoid tilings (T, Γ) that have an azulenic set \mathcal{P} with this extra condition:

$$\forall P_1, P_2 \in \mathcal{P} : \exists \gamma \in \Gamma : \gamma P_1 = P_2. \quad (\mathcal{AT})$$

In an equivariant azulenoid tiling with this property, there exists a partition of the vertices into azulenes such that each azulene can be mapped to each other azulene by a symmetry of the tiling.

Definition 4.1.3 *An equivariant azulenoid tiling that has property \mathcal{AT} is called an azulene-transitive azulenoid tiling.*

◇ azulene-transitive azulenoid tiling

4.2 The enumeration of azulenoids

4.2.1 Translation to Delaney-Dress symbols

Definition 4.2.1 A_T is the set of all pairs which consist of an azulene-transitive azulenoid tiling together with a marked azulenic set.

◇

By a marked azulenic set in the definition above we mean that the faces corresponding to the facial cycles in each set of the partition are marked.

An azulene corresponds to twenty-four flags in the flagspace distributed over two different $\sigma_0\sigma_1$ -components .

The only possible internal symmetry of an azulene in these tilings is a mirror axis. The only other symmetry that could reduce the number of flags in the Delaney-Dress graph is a glide-reflection with an axis through the center of the bond connecting the pentagon and the heptagon. These symmetries can be ‘removed’ by creating the orientable cover of the tiling as described in paragraph 1.3.7.2. In this orientable cover there is also only one orbit of azulenes.

A_T° **Definition 4.2.2** A_T° is the set of all pairs which consist of an azulene-transitive azulenoïd tiling together with a marked azulenic set where the azulenes have no internal symmetry. \diamond

We use a two-step process based on a top-down approach, where we first generate a set of coarser structures and afterwards generate all the azulenoïd tilings from the structures generated in this first step. We can group the pentagon and the heptagon into one tile. Since azulene only has eight outgoing bonds, we can transform it to an octagon (see Figure 4.3). An octagon without internal symmetries corresponds to a single $\sigma_0\sigma_1$ -component with sixteen flags. This gives us a new set of tilings.

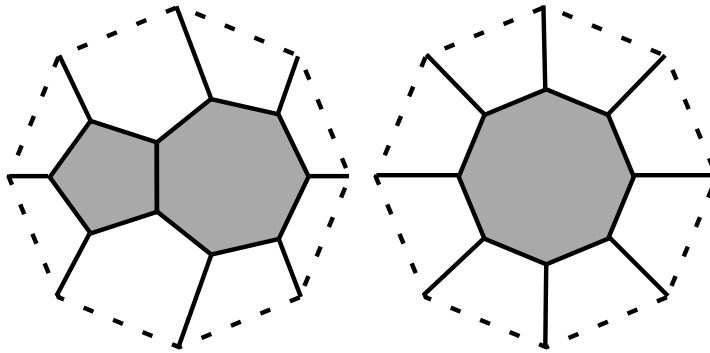


Figure 4.3: An azulene has 8 outgoing bonds.

O_T **Definition 4.2.3** O_T is the set of pairs $((T, \Gamma), \mathcal{O})$ where (T, Γ) is an equivariant tiling with a cubic skeleton graph S and \mathcal{O} is a partition of the vertices of S into facial cycles of length 8, such that for any two octagons $O_1, O_2 \in \mathcal{O}$, there exists an

action $\gamma \in \Gamma$ that maps O_1 on O_2 ; that the octagons in \mathcal{O} have no internal symmetry, i.e., that any action that maps an octagon in \mathcal{O} to itself is necessarily the identity; and that the faces corresponding to the facial cycles of length 8 in \mathcal{O} are marked. \diamond

As described above, each tiling in A_T° can be transformed into a tiling in O_T by replacing the azulenes by octagons. If we take a tiling in O_T and replace the special octagons with azulenes, we get a tiling which belongs to A_T° . We can thus first generate all the tilings in O_T and then replace the special octagons by azulenes in all possible ways. Finally we can get the tilings in A_T by calculating the minimal symbols as described in paragraph 1.3.7.1.

We can now write down the properties that a Delaney-Dress symbol must have to be the symbol of a tiling in O_T .

Property D1 expresses that there is a partition of the vertices into facial cycles of length 8 that correspond to marked octagons without internal symmetries. Property D2 expresses that the skeleton graph is cubic. Finally property D3 expresses that this Delaney-Dress symbol corresponds to a tiling of the Euclidean plane.

D1. There exists a $\sigma_0\sigma_1$ -component O with the following properties:

- $m_{01}(O) = 8$,
- O is a cycle of length 16,
- for all $\sigma_1\sigma_2$ -components V , we have $O \cap V \neq \emptyset$,
- for all $d \in O$, we have $\sigma_2 d \notin O$;

D2. for all $\sigma_1\sigma_2$ -components V , we have $m_{12}(V) = 3$;

D3. $(\mathcal{D}; m_{01}, m_{12})$ satisfies the conditions of Theorem 1.3.16 and the curvature of this symbol is equal to 0.

4.2.2 Restrictions on the Delaney-Dress symbol

4.2.2.1 Restrictions on the graph

In Chapter 3 we already discussed the restrictions that a graph needs to satisfy to be a Delaney-Dress graph. We will first summarise these results, before we give

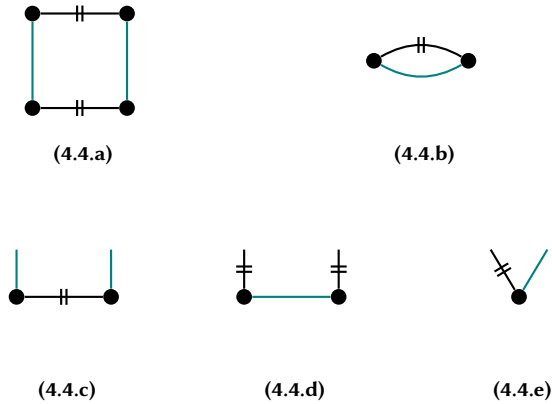


Figure 4.4: The five possible $\sigma_0\sigma_2$ -orbits.

the restrictions that are specific for this class of tilings.

A $\sigma_0\sigma_2$ -component corresponds to a 4-cycle in the flag graph. Such a cycle has only five possible quotients in the Delaney-Dress graph. These five possibilities are shown in Figure 4.4. Figure 4.4.b and Figure 4.4.e correspond to $\sigma_0\sigma_2$ -orbits of size 1 in the Delaney-Dress graph. The other three all correspond to $\sigma_0\sigma_2$ -orbits of size 2 in the Delaney-Dress graph.

All the vertices have degree 3 and this means that a $\sigma_1\sigma_2$ -orbit corresponds to a 6-cycle in the flag graph. Such a cycle has only four possible quotients in the Delaney-Dress graph. These four possibilities are shown in Figure 4.5.

An octagon in the tiling corresponds to a σ_0, σ_1 -cycle in the flag graph with 16 vertices and thus an octagon without internal symmetry corresponds to a σ_0, σ_1 -cycle with 16 vertices in the Delaney-Dress graph. Since every tiling needs to contain such a special orbit of octagons, we take this 16-cycle as starting point and call this component again O .

Every $\sigma_1\sigma_2$ -component in a tiling in O_T has a non-empty intersection with O . Since O does not contain a σ_1 -semi-edge the $\sigma_1\sigma_2$ -component shown in Figure 4.5.a is not possible. As the marked octagons do not neighbour each other, we have that

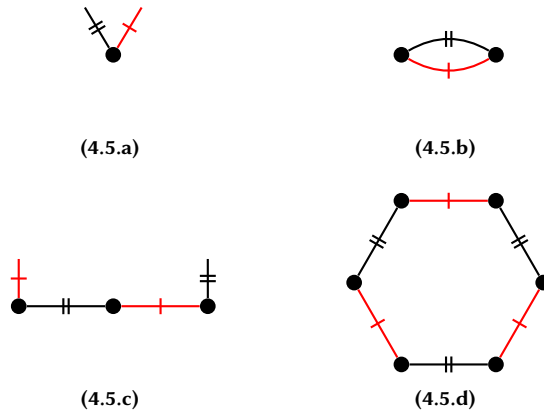


Figure 4.5: The four possible $\sigma_1\sigma_2$ -orbits.

for each flag $f \in O$, $\sigma_2(f)$ must be in another $\sigma_0\sigma_1$ -component. Figure 4.5.b corresponds to a vertex where equivalent faces meet as the σ_2 -edge does not leave the $\sigma_0\sigma_1$ -component. So the $\sigma_1\sigma_2$ -component shown in this figure is not possible. The $\sigma_1\sigma_2$ -component in Figure 4.5.c together with the fact that m_{12} is equal to 3 on this component would correspond to a vertex where two equivalent faces meet a third face. This would be possible if O corresponded to the third face. However, since the σ_1 -semi-edge may not belong to O , this is also not possible. So we find that all the $\sigma_1\sigma_2$ -components around O must be of the kind shown in Figure 4.5.d. Since only one face around each vertex can correspond to O , only one σ_1 -edge for each σ_1, σ_2 -component can be contained in O . So we can add 8 σ_1, σ_2 -components around O each of which has 2 vertices in common with O and 4 new vertices. This gives us the partial Delaney-Dress graph containing $16 + 32 = 48$ vertices that is shown in Figure 4.6.

Since each component corresponding to a vertex has a non-empty intersection with the orbit corresponding to the octagon, we have now found all the vertices, and thus also all the flags in the Delaney-Dress graph.

As the σ_0 -edges in O are already part of a $\sigma_2\sigma_0\sigma_2$ -path of length 3, the only

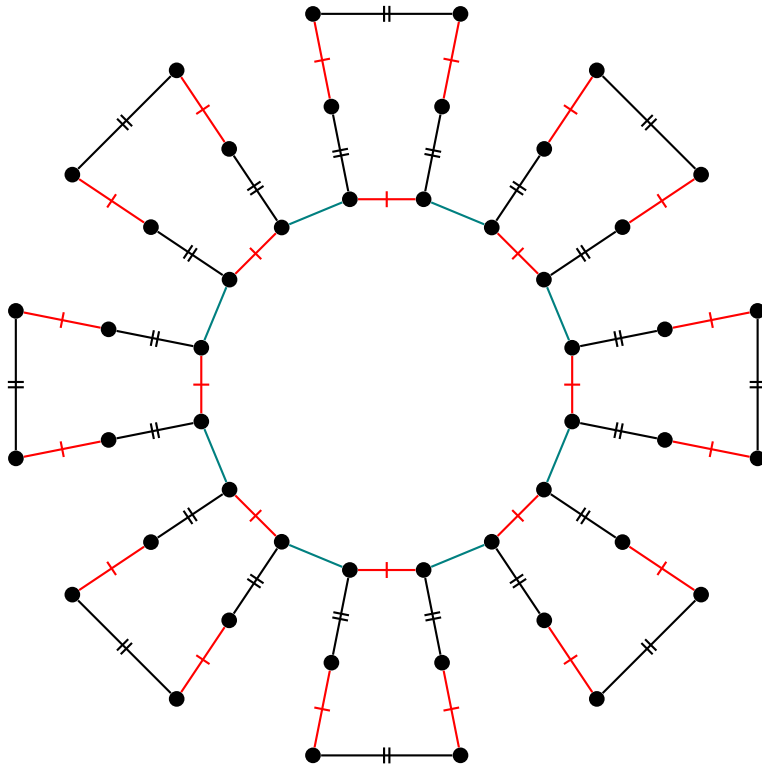


Figure 4.6: *The first partial Delaney-Dress graph \mathcal{D} after connecting all the vertex components to the octagon component.*

possible type of $\sigma_0\sigma_2$ -component that can contain these edges is the one shown in Figure 4.4.a. This can also easily be seen because the edges of the octagon corresponding to O need to separate the octagon from a face that is not equivalent and the only $\sigma_0\sigma_2$ -components that do this are the components of the type in Figure 4.4.a.

After we add these components we get the partial Delaney-Dress graph that is shown in Figure 4.7. This graph however is not yet complete: there are still some connections missing at 16 vertices of degree 2.

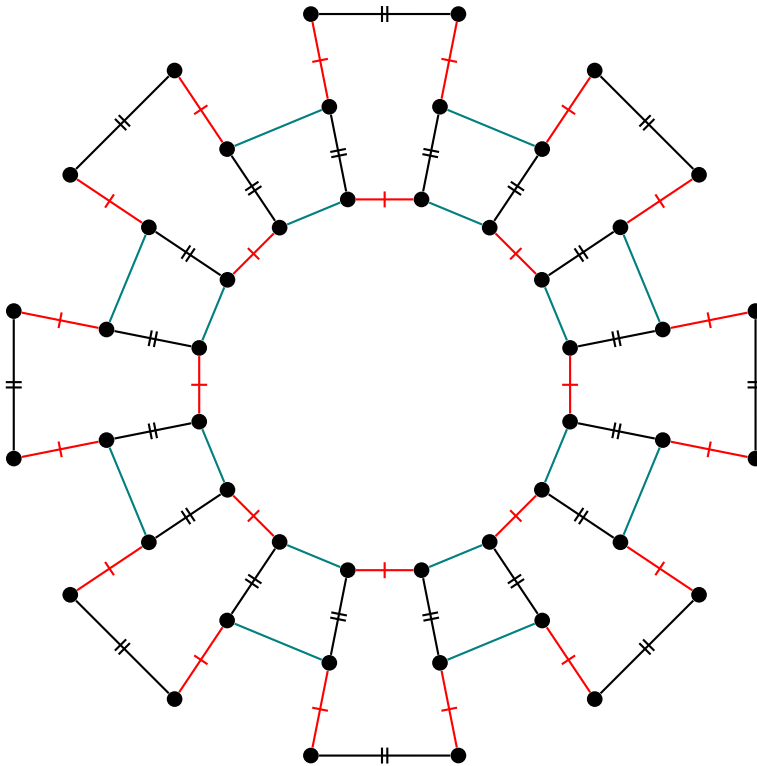


Figure 4.7: *The partial Delaney-Dress graph \mathcal{D} at the beginning of the algorithm.*

For what follows we will denote the partial Delaney-Dress graph that we have at

this point with \mathcal{D} . It is depicted in Figure 4.7.

4.2.2.2 Restrictions on the mappings

We will now try to find more restrictions for the complete Delaney-Dress symbol. We still have not used condition D3 which states, among other things, that the curvature for the symbol is zero, i.e.

$$\sum_{d \in \mathcal{D}} \left(\frac{1}{m_{01}(d)} + \frac{1}{m_{12}(d)} - \frac{1}{2} \right) = 0.$$

The partial symbol \mathcal{D} contains 48 flags and for each flag $d \in \mathcal{D}$ it is so that $m_{12}(d) = 3$. Knowing this, we can reduce the equality above to the following:

$$\sum_{d \in \mathcal{D}} \frac{1}{m_{01}(d)} + 16 - 24 = 0,$$

or

$$\sum_{d \in \mathcal{D}} \frac{1}{m_{01}(d)} = 8.$$

For the component O we also already know the value of m_{01} . These are 16 flags for which we know that for each flag $d \in O$: $m_{01}(d) = 8$. We will denote the set of remaining flags by \mathcal{D}_{32} . (This means that \mathcal{D}_{32} consists of the outer two circles in Figure 4.7 or, equivalently: the flags that do not belong to the inner octagon in Figure 4.17.) This gives us the equality

$$\sum_{d \in \mathcal{D}_{32}} \frac{1}{m_{01}(d)} = 8 - 16 \cdot \frac{1}{8} = 6.$$

The function m_{01} is constant on a $\sigma_0\sigma_1$ -component. As can be seen in Figure 4.7, this implies that the flags in \mathcal{D}_{32} are already grouped in groups of 4. There are 8 $\sigma_0\sigma_1$ -components in the partial Delaney-Dress symbol induced by \mathcal{D}_{32} . We use the notation $M_{01}(i)$ to denote the value of m_{01} on the i th $\sigma_0\sigma_1$ -component. This gives us the following equality

$$\sum_{i=1}^8 \frac{4}{M_{01}(i)} = 6 \Rightarrow \sum_{i=1}^8 \frac{1}{M_{01}(i)} = \frac{3}{2}.$$

The number of flags in a $\sigma_0\sigma_1$ -component is a divisor of $2m_{01}$. Since the flags are already grouped in $\sigma_0\sigma_1$ -components of 4 flags, a $\sigma_0\sigma_1$ -component in the partial Delaney-Dress symbol induced by \mathcal{D}_{32} in a Delaney-Dress symbol for a tiling from O_T will always contain a multiple a of 4 flags. Thus

$$4a = 2m_{01} \Rightarrow a = \frac{m_{01}}{2},$$

or in words: m_{01} has to be even. And since a face of order 2 does not exist, the value of m_{01} is at least 4.

So we need to solve the following problem: find 8 even positive integers a_i ($1 \leq i \leq 8$) greater than 2 that satisfy the condition

$$\sum_{i=1}^8 \frac{1}{a_i} = \frac{3}{2}.$$

We can already fix some integers based on this property. When we calculate the mean value ν of the inverse of the integers, we find

$$\sum_{i=1}^8 \frac{1}{a_i} = \frac{3}{2} \Rightarrow 8\nu = \frac{3}{2} \Rightarrow \nu = \frac{3}{16} = 0,1875.$$

The maximum value for $\frac{1}{a_i}$ is $\frac{1}{4} = 0,25$. The second largest value is $\frac{1}{6} = 0,1666\dots$. This implies that at least one of the eight integers must be 4. We again use ν to denote the mean of the inverse of the remaining 7 integers.

$$\sum_{i=1}^7 \frac{1}{a_i} = \frac{3}{2} - \frac{1}{4} = \frac{5}{4} \Rightarrow 7\nu = \frac{5}{4} \Rightarrow \nu = \frac{5}{28} \approx 0,18.$$

This means that still at least one of the remaining seven integers must be 4.

$$\sum_{i=1}^6 \frac{1}{a_i} = \frac{5}{4} - \frac{1}{4} = 1.$$

For the remaining 6 integers we cannot fix any more values, so we will enumerate those.

We have 24 (6×4) flags for which these integers will be used. If we assume that these six are all equal to m , then we have 24 flags c for which $m_{01}(c) = m$. Due to the crystallographic restriction theorem (see p. 113), the maximum branching of an orbit in a Delaney-Dress symbol of a tiling of the plane is 12. Therefore the 24 flags c for which $m_{01}(c) = m$, can correspond to a face of at most size $\frac{24 \cdot 12}{2} = 144$, so 144 is an upper bound for the value of m .

When we take 144 as upper bound and perform an exhaustive search, we find the sequences of integers that are shown in Table 4.1.

The integers								# circular strings	# symbols	# canonical symbols
4	4	4	4	4	6	24	24	12	6	6
4	4	4	4	4	8	12	24	21	21	21
4	4	4	4	4	8	16	16	12	30	30
4	4	4	4	4	10	10	20	12	0	0
4	4	4	4	4	12	12	12	5	34	31
4	4	4	4	6	6	8	24	54	0	0
4	4	4	4	6	6	12	12	33	2	2
4	4	4	4	6	8	8	12	54	9	9
4	4	4	4	8	8	8	8	8	140	108
4	4	4	6	6	6	6	12	19	4	3
4	4	4	6	6	6	8	8	38	26	26
4	4	6	6	6	6	6	6	4	25	19

Table 4.1: *The twelve sets of eight integers that are valid candidates for the values of m_{01} in the Delaney-Dress graph.*

4.2.3 Enumerating the octagon tilings

The next step is to take each possible distinct assignment of the sequence of eight numbers to the flags in the partial Delaney-Dress symbol. This gives us the number of canonical circular strings that are given in Table 4.1.

Afterwards we assign each of these 272 canonical circular strings to the flags of the $\sigma_0\sigma_1$ -components of the partial Delaney-Dress symbol induced by \mathcal{D}_{32} . The

symbols constructed this way are complete except for some σ_0 -edges. The only thing that remains to be done is to try to complete these partial symbols by filling in the missing σ_0 -edges.

When we compare Figure 4.4 and Figure 4.7, we see that the only possible $\sigma_0\sigma_2$ -components are of the type 4.4.a, 4.4.b or 4.4.c, because all the σ_2 -edges are already present and none of them is a semi-edge.

The final step in generating the octagon tilings is to describe the action of σ_0 for the remaining flags. Each flag d of degree 2 in \mathcal{D} has three possible types of operations that can be applied to add the σ_0 -edge:

- attach a σ_0 -semi-edge to d ,
- connect d to σ_2d , and
- connect d to a flag of degree 2 in another $\sigma_0\sigma_2$ -component of \mathcal{D} for which the function m_{01} has the same value.

We systematically try all possible σ_0 -edges and backtrack in case of contradictions like

- the number of flags in a completed $\sigma_0\sigma_1$ -component might not be a divisor of $2m_{01}$,
- in case the $\sigma_0\sigma_1$ -component still contains flags with degree 2: the number of flags in the $\sigma_0\sigma_1$ -component might be larger than $2m_{01}$.

This approach leads to isomorphic Delaney-Dress symbols. Since the final number of symbols that we find this way is quite small (297 symbols) we apply a simple filtering method afterwards to remove isomorphic copies and we finally find that O_T contains 255 tilings.

4.2.4 Inserting the azulene

As can be seen in Figure 4.8, it is possible that a tiling contains two different sets of octagons that satisfy the property that each vertex is contained in exactly one of

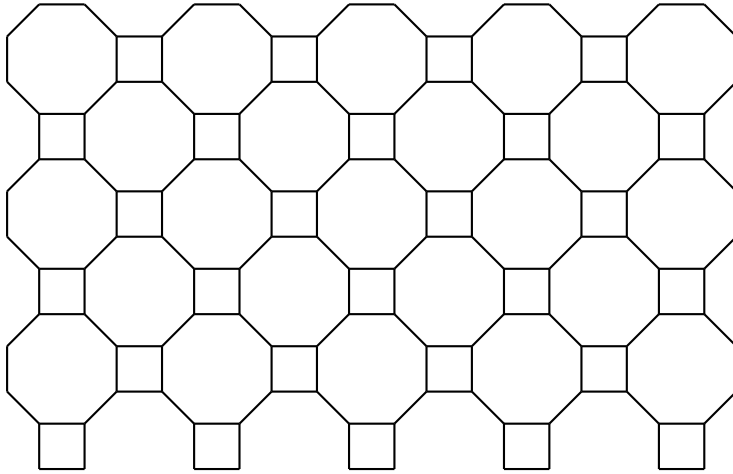


Figure 4.8: *The bathroom tiling belongs to O_T and contains two different sets of octagons that satisfy the property that each vertex is contained in exactly one of the octagons in the set.*

the octagons in the set. In such a tiling, the replacement by an azulene would in principle have to take place for every set in order to get all tilings in A_T . But in fact, the tiling in Figure 4.8 is the only tiling for which this happens and here both sets of octagons are equivalent under the symmetry group and would give isomorphic tilings of A_T .

This was just the first, coarser step in the generation. The next step is to replace those octagons with azulenes to create the azulenoid tilings. When we do this an azulene can in principle be pasted into an octagon in eight different ways by choosing two edges in the octagon that are separated by two other edges, adding a vertex to these edges and connecting these two vertices. Owing to symmetries of the octagon tiling it will be possible that two or more distinct choices for the two edges lead to the same azulenoid tiling. This could be avoided by using the double coset method (cf. Dendral [14]), but there are 255 octagon tilings and together with the 8 possible distinct choices of edges, this leads to only 2040 possible azulenoids, which

is a small enough number for us simply to try all possible choices of edges and filter out the isomorphic ones afterwards as we did earlier during the generation of the octagon tilings. Figure 4.9 shows this process of inserting two vertices in the tiling and connecting those two vertices by an edge. We see an octagon (16-cycle) and two edges (4-cycles). We add a vertex of degree 3 (6-cycle) to each of the edges, respectively a-b-c-d-e-f and g-h-i-j-k-l, and connect these two vertices by the edge (4-cycle) b-c-h-i.

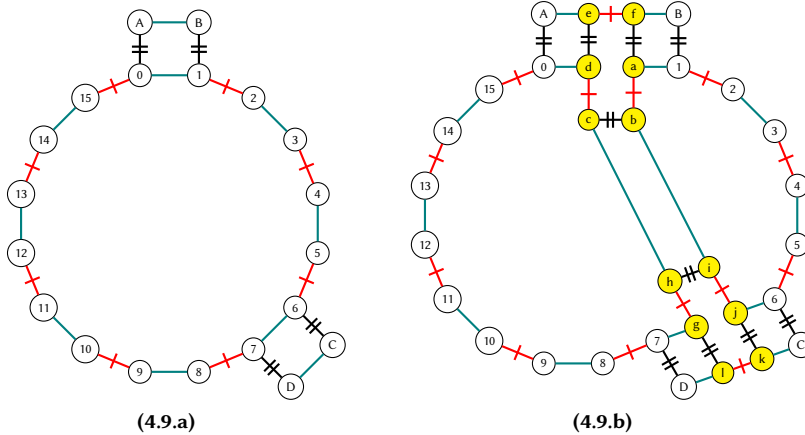


Figure 4.9: Replacing an octagon with an azulene in the partial Delaney-Dress symbol.

First of all we notice that the Delaney-Dress symbol grows. We add twelve flags to the graph. In Figure 4.18 these are denoted with small letters a through l. Table 4.2 shows the changes and the new values for the σ 's. These changes are also shown in Figure 4.9.

Since we add two new vertices to this symbol, there are several $\sigma_0\sigma_1$ -components for which the value of m_{01} has to change. First of all we will have to adjust the values for the azulene itself. The new values are shown in Table 4.3.

The value of m_{01} for the $\sigma_0\sigma_1$ -component that contains flag A also must be adjusted. We added a vertex to that face for each time that the edge of A is in that

flag	σ_0	σ_1	σ_2	flag	σ_0	σ_1	σ_2	
0	d			c	h	d	b	
1	a			d	0	c	e	
6	j			e	A	f	d	
7	g			f	B	e	a	
A	e			g	7	h	l	
B	f			h	c	g	i	
C	k			i	b	j	h	
D	l			j	6	i	k	
a	1		b	f	k	C	l	j
b	i		a	c	l	D	k	g

Table 4.2: Changes to σ_i for the partial Delaney-Dress symbol in Figure 4.9 when replacing an octagon by an azulene. The actions that are not changed are marked in gray.

orbit. Thus we the value of m_{01} is increased by twice the old value of m_{01} divided by the number of flags in the original symbol in the $\sigma_0\sigma_1$ -component that contains A.

The value of m_{01} for the $\sigma_0\sigma_1$ -component that contains flag C is adjusted in a similar manner.

It is possible that flags A and C are in the same $\sigma_0\sigma_1$ -component. This however does not change the fact that we need to increase the value of m_{01} for this component by 1 for each vertex we have added to it.

The last step is to filter out the isomorphic tilings. This is done by keeping a list of the minimal Delaney-Dress symbols of the tilings in canonical form. When a new Delaney-Dress symbol is constructed, its minimal Delaney-Dress symbol is calculated and the canonical form is compared to all the symbols found up to that point. If the new symbol is not already in the list, it is added at the end. After we filter out the isomorphic ones we find 1274 tilings.

flag	m_{01}	flag	m_{01}	flag	m_{01}
0	7	8	7	a	5
1	5	9	7	b	5
2	5	10	7	c	7
3	5	11	7	d	7
4	5	12	7	g	7
5	5	13	7	h	7
6	5	14	7	i	5
7	7	15	7	j	5

Table 4.3: *Adjusting m_{01} for the azulene*

4.3 Testing

The algorithm described above has been implemented as the C-program azul. This program takes less than 2 seconds on a 2.2 GHz Pentium Core Duo processor to generate the 1274 tilings. This confirms our claims that isomorphism rejection by lists is more than sufficient in this case.

The program has been tested against an independent program implementing a slightly different algorithm. This program was implemented by Olaf Delgado-Friedrichs. It is also based on the use of Delaney-Dress symbols, but generates in its first phase all tilings in which the tiles are octagons and all related by symmetry, but the vertex degree may be arbitrary. Relative to the approach described above, the order of adding edges to the Delaney-Dress graph (here σ_2 instead of σ_0) and generating combinations of numbers (here m_{12} instead of m_{01}) is reversed. The $\sigma_0\sigma_1$ -cycle representing that tile is then replaced by the partial Delaney-Dress graph \mathcal{D} and the σ_2 -edges by σ_0 -edges.

The results of both algorithms agree, allowing us to be very confident in the completeness of the list.

4.4 Marked and unmarked tilings

Since we calculated the minimal Delaney-Dress symbol of the symbols that we found without concerning ourselves with marked tiles, the 1274 tilings we found are unmarked tilings. This means that these are azulenoid tilings, but the azulenoid set is not fixed in these tilings. For some tilings there is more than one possible azulenoid set. Sometimes these different azulenoid sets lead to isomorphic marked tilings as shown in Figure 4.10. But this is not the rule as can be seen in Figure 4.11.

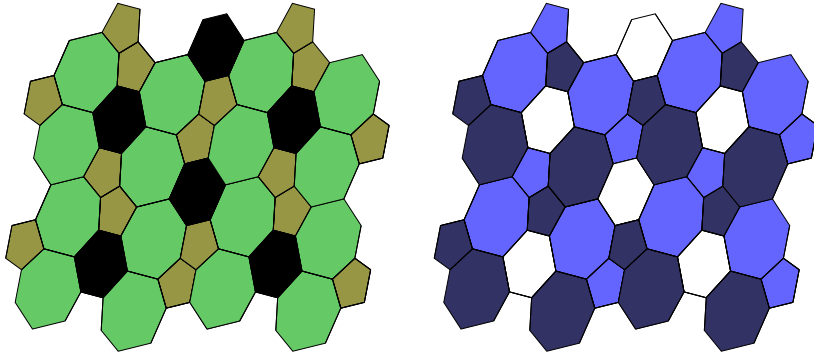
Definition 4.4.1 *Given an azulenoid tiling (T, Γ) and two azulenoid sets \mathcal{P}_1 and \mathcal{P}_2 of T , \mathcal{P}_1 and \mathcal{P}_2 are isomorphic azulenoid sets if there is an element of Γ that maps the elements of \mathcal{P}_1 on to the elements of \mathcal{P}_2 .* \diamond

When we generate marked tilings, we find that there are 1324 different marked azulenoid tilings. In this set there are 48 tilings that are equivariantly equivalent to another tiling as an unmarked tiling, but not as a marked tiling. These 48 marked tilings correspond to 24 unmarked tilings. There are 98 tilings that as an unmarked tiling have a minimal symbol that is isomorphic to another tilings (unmarked) minimal symbol, but not as marked tilings. Among these 98 there are 94 that come in pairs and thus correspond to 47 unmarked minimal tilings and 4 that come in a quartet and thus correspond to 1 unmarked minimal tiling (see Figure 4.15). So these 98 marked tilings correspond all together to 48 unmarked tilings, which gives the difference of 50 tilings.

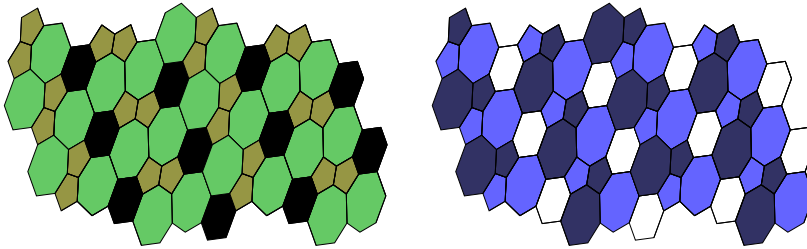
4.5 Inserting the azulene in the chamber system

Replacing the octagons by azulenes can be seen as subdividing two edges at distance 3 in an octagon, and adding an edge between the new vertices. It might be easier to visualise this operation in terms of the chamber system (which was introduced in paragraph 1.3.5), so we include some illustrations of this process for the chamber system.

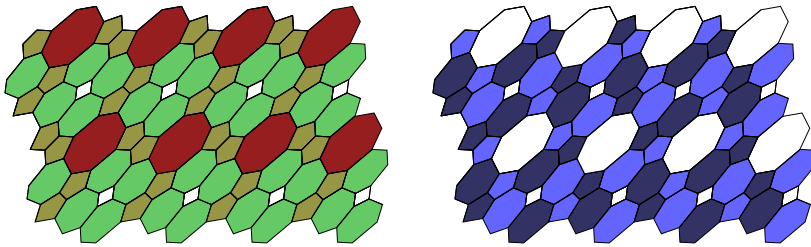
Figure 4.16 shows the local chamber systems for an octagon and a vertex of degree 3 and Figure 4.17 and Figure 4.18 show the equivalent of Figure 4.7 and Figure 4.9 in



(4.10.a)



(4.10.b)



(4.10.c)

Figure 4.10: Some tilings with two isomorphic azulenic sets. On the left hand side the tiling is shown with a separate colour for each face size. On the right hand side the two azulenic sets are highlighted in different colours and the other faces are shown in white.

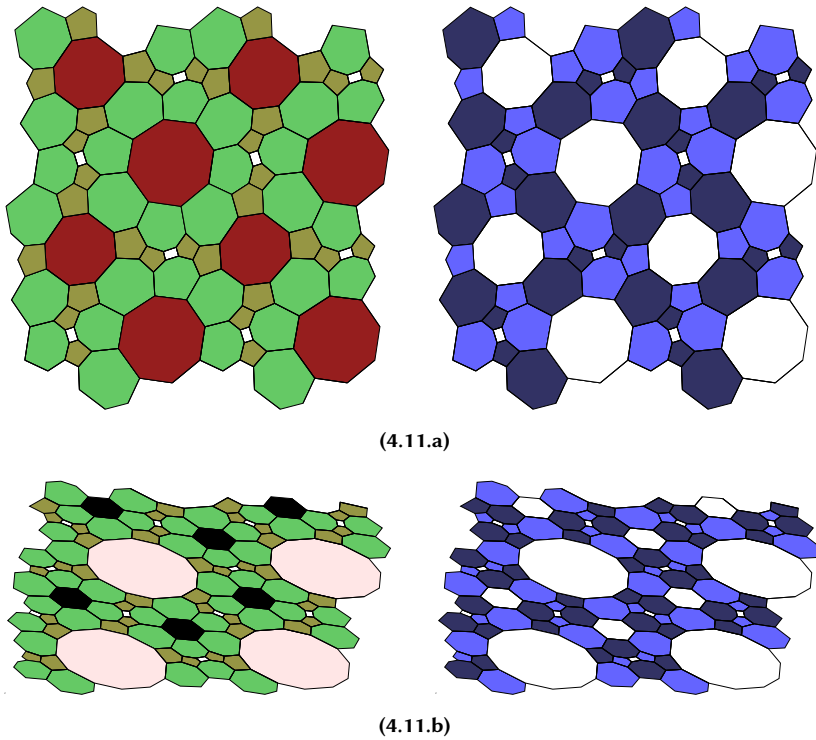
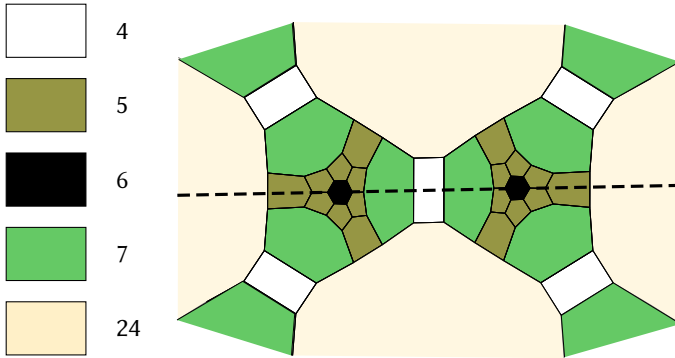
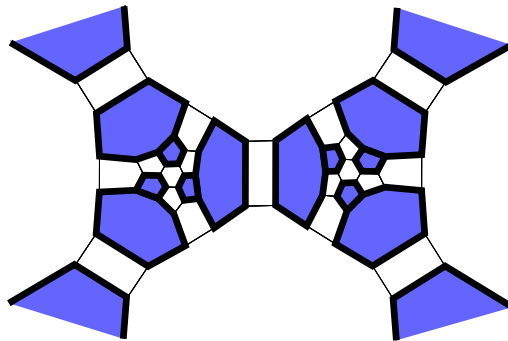


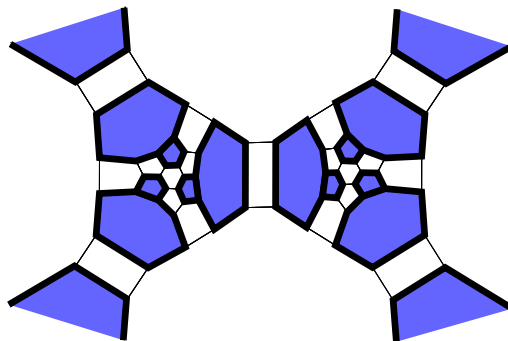
Figure 4.11: Some tilings with two nonisomorphic azulenic sets. On the left hand side the tiling is shown with a separate colour for each face size. On the right hand side the two azulenic sets are highlighted in different colours and the other faces are shown in white.



(4.12.a) An azulenoid tiling with faces of size 4, 5, 6, 7 and 24.

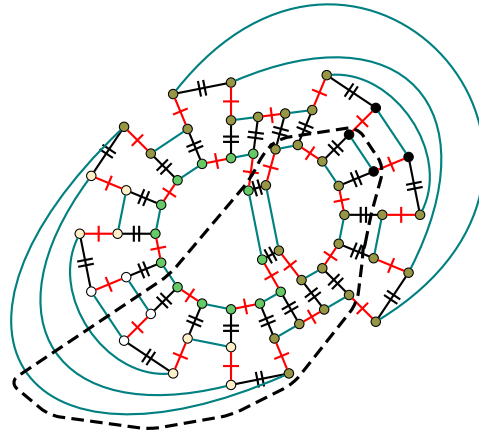


(4.12.b) There is a rotational symmetry of 180 degrees in the center of the quadrangle.

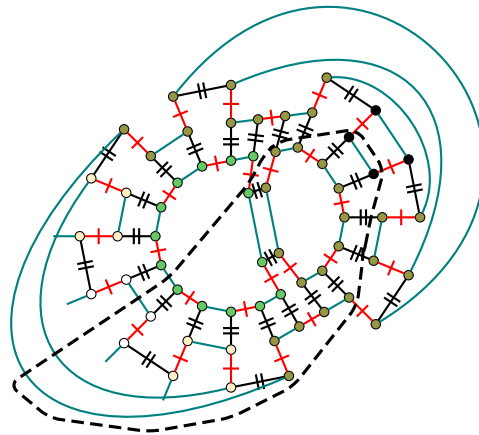


(4.12.c) There is a vertical mirror axis through the middle of the quadrangle.

Figure 4.12: Marked and unmarked tilings do not necessarily have the same minimal symbol.



(4.13.a)



(4.13.b)

Figure 4.13: *The Delaney-Dress graphs for the marked tilings in Figure 4.12.*

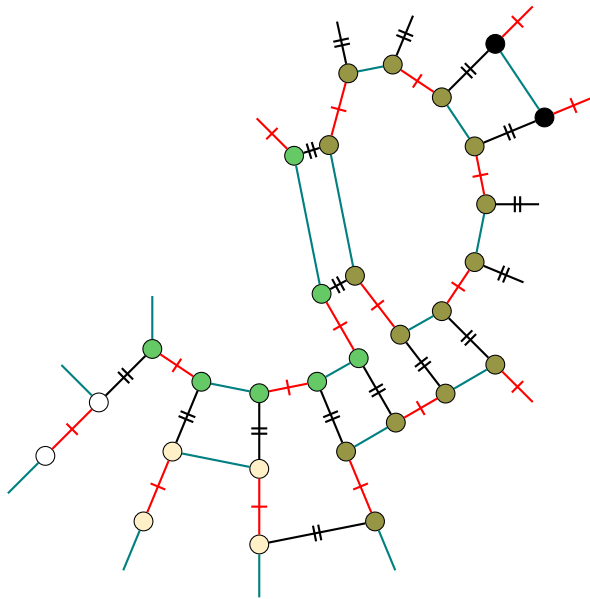
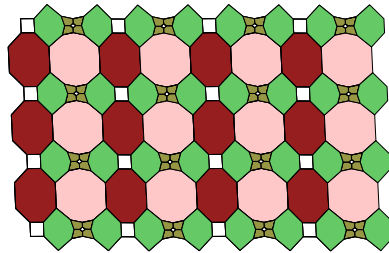
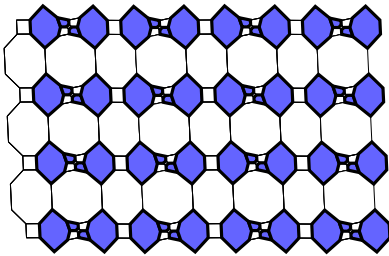


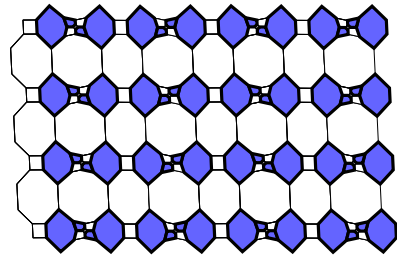
Figure 4.14: *The Delaney-Dress graphs for the minimal unmarked tiling in Figure 4.12.*



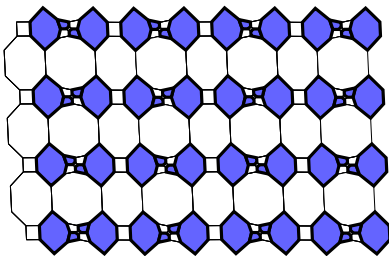
(4.15.a)



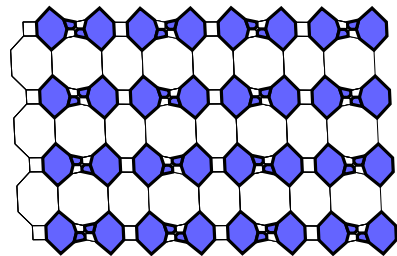
(4.15.b)



(4.15.c)



(4.15.d)



(4.15.e)

Figure 4.15: An unmarked azulene-transitive azulenoid graph (4.15.a) and the 4 marked azulene-transitive azulenoid graph (4.15.b - 4.15.e) that have this tiling as underlying unmarked tiling.

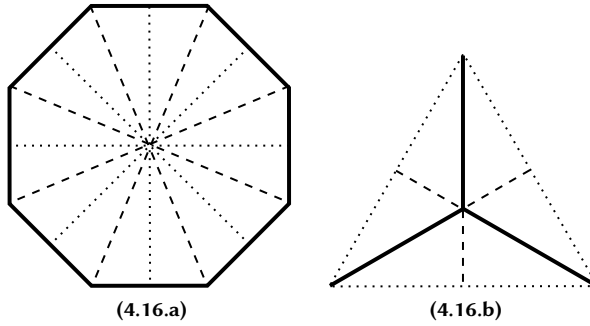


Figure 4.16: *The local chamber system of an octagon (a) and a vertex of degree three (b). For each chamber d the neighbouring chamber at the solid edge is $\sigma_0(d)$, at the dotted edge is $\sigma_1(d)$ and at the dashed edge is $\sigma_2(d)$.*

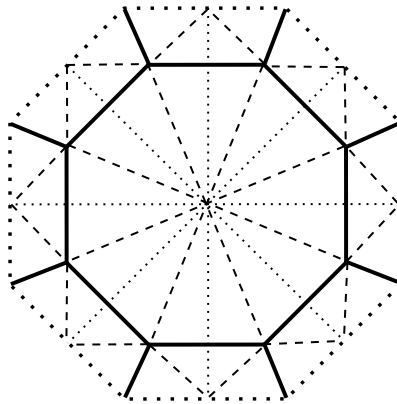


Figure 4.17: *The chamber system of our start point. For each chamber d the neighbouring chamber at the solid edge is $\sigma_0(d)$, at the dotted edge is $\sigma_1(d)$ and at the dashed edge is $\sigma_2(d)$.*

terms of the chamber system.

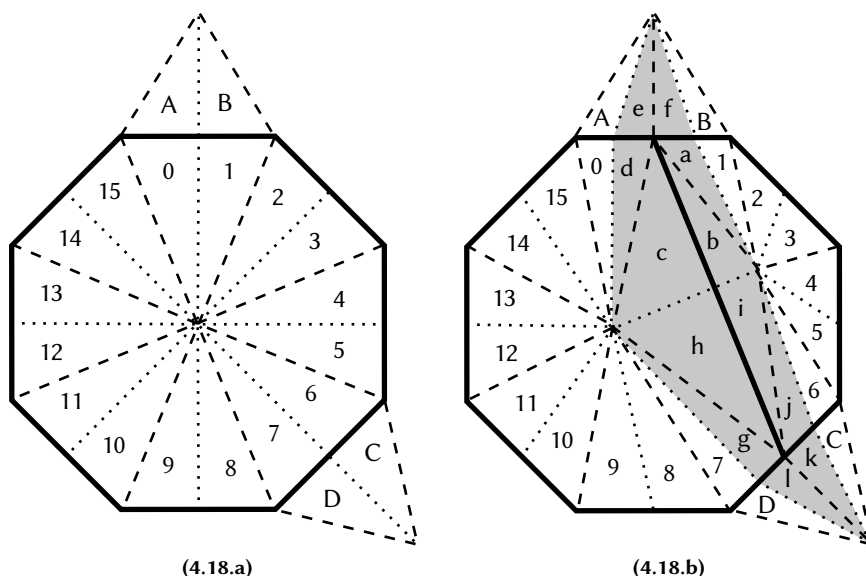


Figure 4.18: Replacing an octagon with an azulene in the Delaney-Dress symbol.

4.6 Visualising the results

The previous paragraphs have focused on the generation of azulenoids. Since this was a cooperation with chemists, we also wanted to visualise the results in a way that makes sense from a chemical point of view. We, however, found that there were no suitable visualisers of tilings for this goal. The available visualisers give mathematically correct equivariant tilings by breaking symmetry with strangely shaped edges when necessary. So we set about writing our own visualiser that uses methods that produce chemically more plausible structures. We decided to write an embedder for periodic graphs for which the isometry group only contains translations. The reason we chose this was because the restriction to translations only would be suf-

ficient to visualise the structure for chemists. We then needed only to translate our Delaney-Dress symbols to such periodic graphs (see later).

We will start by describing how the embedder works. First we needed a representation of a periodic graph.

4.6.1 Periodic graph

Definition 4.6.1 A **periodic graph** is an infinite embedded graph with two independent translational symmetries. ◇ **periodic graph**

Since we have two independent translations in the isometry group, we can construct a fundamental domain that has the shape of a parallelogram.

For the rest of the chapter we will use the convention that the X axis points right and the Y axis points downwards.

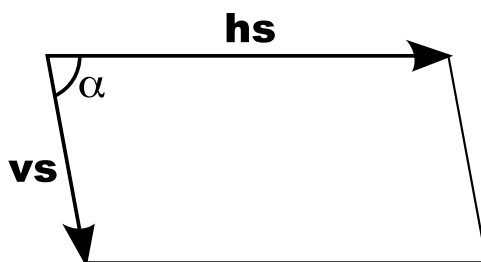


Figure 4.19: Fundamental domain

As can be seen in Figure 4.19, we need three parameters to describe our fundamental domain: an angle α , the length of the horizontal side hs and the length of the vertical side vs .

These fundamental domains form a discrete grid that we parametrise using our previously mentioned convention. The result of this coordinatization is shown in Figure 4.20.

To describe the positions of the vertices in a periodic graph we now only need to describe their positions inside a fundamental domain.

The only part of the periodic graph that still needs to be coded are the edges. For this we look at the fundamental domain with coordinates $(0, 0)$. We will describe the

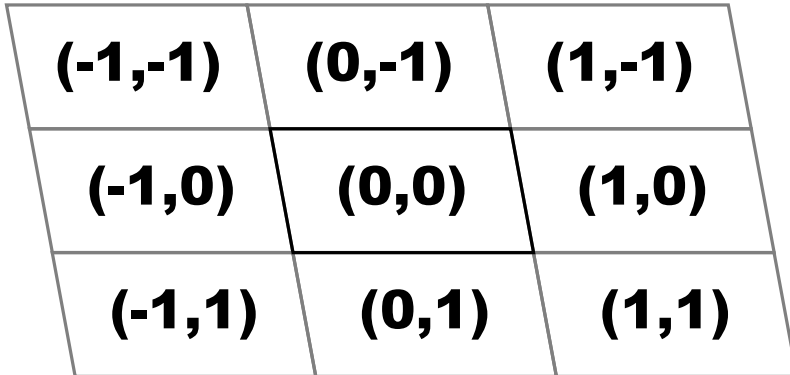


Figure 4.20: *The grid of fundamental domains*

edges for each vertex in this domain. For this we need to store the start vertex, the end vertex and the coordinates of the fundamental domain to which the end vertex belongs. In Figure 4.21 we see three edges that start in vertex 1.

Edge	Start	End	Target
a	1	2	$(0, 0)$
b	1	2	$(0, -1)$
c	1	1	$(0, 1)$

Table 4.4: *The parameters for the edges in figure 4.21*

We are explicitly saving a start and an end vertex for the edges. This means that we are saving directed edges, but since the graphs are undirected, we represent each undirected edge by two directed edges which are each others inverse. Each edge has an inverse edge that can easily be described as follows: $(\text{edge}(v_1, v_2, (x, y)))^{-1} = \text{edge}(v_2, v_1, (-x, -y))$, as can be seen in Figure 4.22. When looking at the infinite structure, we see that this inverse edge is in the same orbit as the edge that would ‘intuitively’ be defined as the inverse edge.

When we change the embedding there is no problem as long as the vertices stay

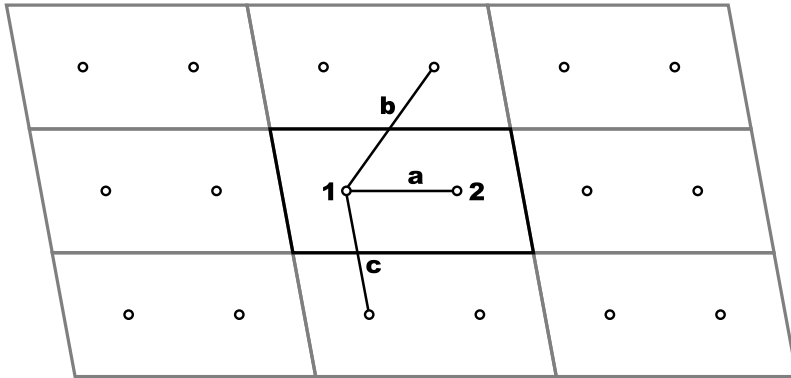


Figure 4.21: Some examples of edges in a periodic graph.

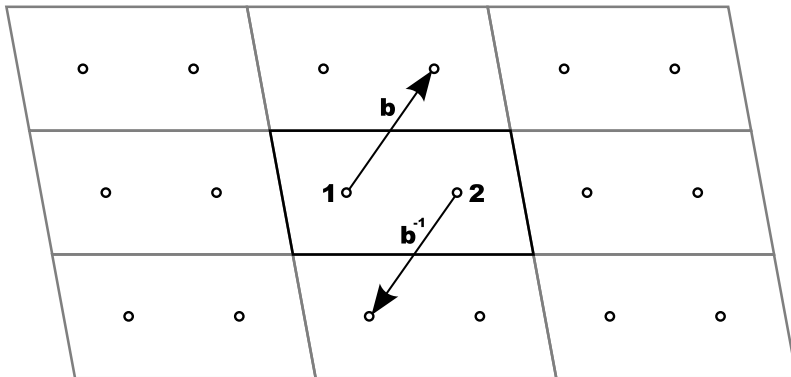


Figure 4.22: An edge and its inverse in a periodic graph.

inside the same fundamental domain. When a vertex crosses the border of the fundamental domain it 'reappears' on the other side. However we need to be careful because at that point we also need to alter the edges.

Suppose we want to move the vertex with coordinates (x_1, y_1) inside the fundamental domain with coordinates $(0, 0)$ to the position with coordinates (x'_2, y'_2) . These coordinates lie inside the fundamental domain with coordinates (X_2, Y_2) . We then calculate the coordinates of the new position as if the fundamental domain with coordinates (X_2, Y_2) was the fundamental domain with coordinates $(0, 0)$. These new coordinates are called (x_2, y_2) . Next we move the vertex to position (x_2, y_2) and for each edge starting in the vertex we change the target (X, Y) to $(X - X_2, Y - Y_2)$ and the target of the inverse edge to $(-X + X_2, -Y + Y_2)$.

For our implementation vertex coordinates are always stored as coordinates in the square with corners $(-1, -1)$, $(-1, 1)$, $(1, 1)$ and $(1, -1)$. They are then transformed to coordinates inside the fundamental domain with the following linear transformation, that can be broken down into two steps.



Figure 4.23: Rescaling of the fundamental domain.

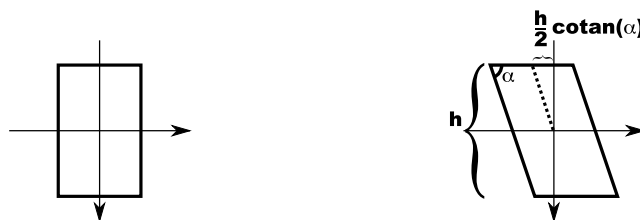


Figure 4.24: Shearing of the fundamental domain.

First the square is rescaled. The new horizontal length of the side is the same as the horizontal side hs in the target parallelogram. The new vertical side has the same length as the height h of the target parallelogram, i.e., $h = vs \cdot \sin(\alpha)$ with vs the length of the vertical side. This means that the horizontal scaling factor is $\frac{hs}{2}$ and the vertical scaling factor is $\frac{h}{2}$. The matrix for this transformation is

$$\begin{bmatrix} \frac{hs}{2} & 0 \\ 0 & \frac{h}{2} \end{bmatrix}.$$

The second step is a shearing of the thus created rectangle along the X-axis. The matrix for this transformation is

$$\begin{bmatrix} 1 & \cotan(\alpha) \\ 0 & 1 \end{bmatrix}.$$

This gives us the final transformation

$$\begin{bmatrix} \frac{hs}{2} & 0 \\ 0 & \frac{h}{2} \end{bmatrix} \cdot \begin{bmatrix} 1 & \cotan(\alpha) \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{hs}{2} & \frac{h}{2} \cotan(\alpha) \\ 0 & \frac{h}{2} \end{bmatrix},$$

and the inverse transformation

$$\frac{4}{hs \cdot h} \begin{bmatrix} \frac{h}{2} & -\frac{h}{2} \cotan(\alpha) \\ 0 & \frac{hs}{2} \end{bmatrix} = \begin{bmatrix} \frac{2}{hs} & -\frac{2}{hs} \cotan(\alpha) \\ 0 & \frac{2}{h} \end{bmatrix}.$$

4.6.2 Embedding the tiling for a Delaney-Dress symbol

We first translate the translation-only cover of the symbol to a periodic graph. In 1.3.7.2 we already described how to get the translation-only cover from the symbol. To avoid special cases we secure that no vertices lie on the edge of the fundamental patch, i.e., we make sure that no $\sigma_1\sigma_2$ -component is incomplete. This can be achieved by constructing the fundamental patch by a depth-first traversal where the children are visited in the order $\sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_0$. By choosing this order, we will primarily group the vertex components, next the face components and lastly the edge components, and thus the border of the fundamental patch will split edge components and face components.

We can easily check whether a fundamental patch corresponds to a quadrangle or a hexagon by counting the number of patches that meet at an incomplete cycle. For a quadrangle this is 4 (at a ‘corner’ of the fundamental domain) or 2 (at an ‘edge’ of the fundamental domain), for a hexagon it is 3 (at a ‘corner’ of the fundamental domain) or 2 (at an ‘edge’ of the fundamental domain). This counting can be done by choosing a flag in the fundamental patch that belongs to the border. We then look at the components through that flag in the original symbol and count how many times we cross the border in the fundamental patch when traversing those components. If this number is not equal to 2, it has to be 3 or 4. If it is 2, we choose another flag and recount for that flag. As soon as we find a 3 or a 4, we can decide whether this fundamental patch corresponds to a quadrangle or a hexagon. If we need to cross the border of the fundamental patch twice, we say that the component is cut into two parts, and analogously for three and four.

If the fundamental patch corresponds to a quadrangle we search for a flag that lies on the boundary of the fundamental patch. When we find such a flag x we construct the edges of the patch as follows. We walk along the $\sigma_0\sigma_2$ -component through x until we meet the border in the flag x_1 . Next we walk along the $\sigma_0\sigma_1$ -component through x_1 until we meet the border in the flag x_2 . We now repeat these steps until we meet an orbit that is cut into four parts. We call this component the corner b . We keep repeating the previous steps until we meet another orbit that is cut into four parts. This component will be called corner c . Finally we return to the flag x and repeat the same steps, but this time we start with the $\sigma_0\sigma_1$ -component through x . When we meet a component that is cut into four parts, we call that component corner a . Also see Figure 4.25.a. This is sufficient: we won’t need the exact other edges, because ab will be identified with dc and bc with ad . We determine the number of vertices inside this patch, i.e., the number of $\sigma_1\sigma_2$ -component s . Then we turn our attention to the edges: when a $\sigma_0\sigma_2$ -component is complete this corresponds to an edge that does not intersect with the border (i.e., target $(0, 0)$), otherwise it will intersect either ab or bc . When it intersects with ab it has target $(-1, 0)$ and when it intersects with bc it has target $(0, -1)$.

If the fundamental patch corresponds to a hexagon we proceed in a similar way:

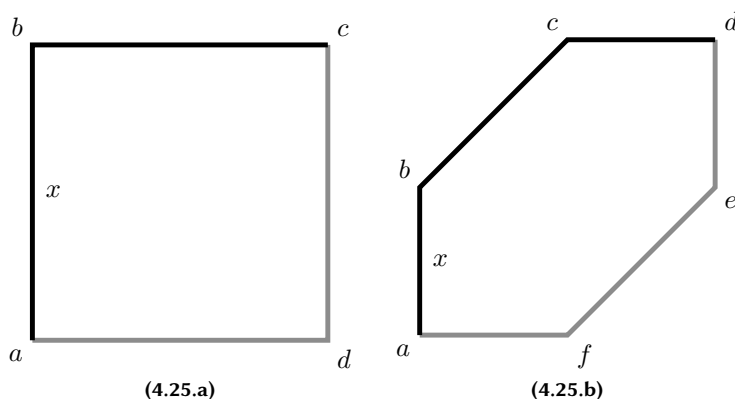
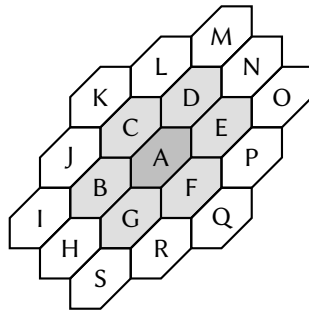


Figure 4.25: *The two types of boundaries for fundamental patches.*

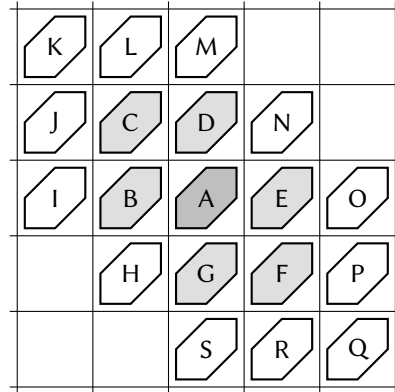
we search for a flag that lies on the boundary of the fundamental patch. When we find such a flag x we construct the edges of the patch by walking along components that are cut into two until we meet a component that is cut into three parts. This is the corner b . We then continue until we meet another corner. This is c . But now we still continue because we need the third corner d . Finally we again return to x and go in the other direction until we meet the corner a . Also see Figure 4.25.b. As before we won't need the exact other edges, because ab will be identified with ed , bc with fe and cd with af . The vertices and internal edges are handled in exactly the same way as before. The other edges will intersect either ab , bc or cd . When it intersects with ab it has target $(-1, 0)$ and when it intersects with cd it has target $(0, -1)$, just as before, and when it intersects bc it has target $(-1, -1)$.

Figure 4.26 shows a schematic view of how the hexagonal fundamental patches are translated to the quadrangular grid of the periodic graph. In Figure 4.27 this is made more concrete by giving an example with an actual tiling.

To embed the periodic graph we implemented several force-directed graph embedders such as spring embedders, Tutte embedders and embedders that change the



(4.26.a)



(4.26.b)

Figure 4.26: A schematic view of how hexagonal fundamental patches are positioned in a quadrangular grid.

fundamental domain.

A spring embedder is a well known algorithm for embedding graphs. It starts from an arbitrary embedding of the graph and regards the edges as springs pulling the vertices together or repelling them away from each other. We have used different spring embedders. Based on their respective goals, we can distinguish:

- embedders that try to get the edges to a fixed length,
- embedders that try to get the edges to equal length by taking the mean as guide value,
- embedders that try to get the edges to length zero,
- embedders that do one of the above, but also pull the vertices of a face to the center of that face.

The spring embedders that also contract faces were added because otherwise faces with a great order would grow quite large, and this counteracts that effect.

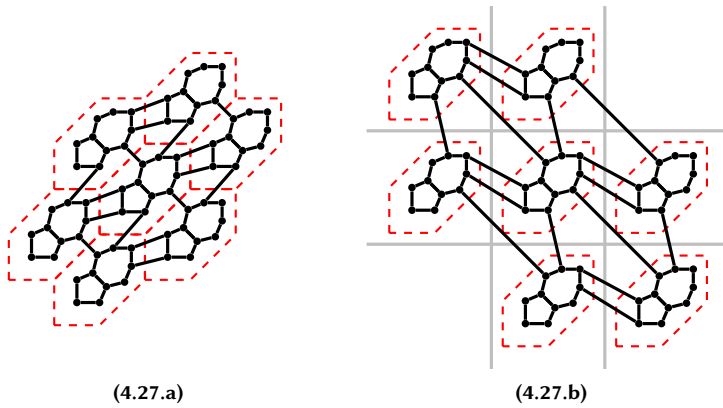


Figure 4.27: An example of a tiling with a hexagonal fundamental patch that has been translated to a periodic graph. The boundaries of the hexagonal fundamental patches are drawn in the periodic graph to clarify the grouping of the vertices. The boundaries of the fundamental patch are not used once the tiling has been translated to a periodic graph.

A Tutte embedder regards the vertices as masses. It fixes some points and then tries iteratively to place the other vertices in the barycenters of their neighbours. It was proven by Tutte [9] that when this is done for a planar graph and the vertices and edges that are fixed form a convex polygon, that the resulting embedding while have no edges crossing. Due to this last property this embedder was used to obtain the initial embedding. When translating the fundamental patch to a periodic graph, we get a cyclic order of the edges that cross the boundary of the fundamental patch. We can use this order to place the vertices which are incident to these edges and the vertices on faces that cross the boundary of the fundamental patch on a pre-embedded cycle in the fundamental domain. Afterwards we use the Tutte embedder to obtain the initial embedding. In the other embedders no special measures were taken to prevent crossing edges, simply because these embedders were written specifically for this case, and it never occurred that the graph was changed to introduce crossing edges.

Finally, we have some embedders that change the fundamental domain based on an energy function. There are two different energy functions we implemented. The first energy function is based on the angles between the edges meeting in a vertex. If we denote the degree of a vertex v by d_v and the sequence of angles between the edges at that vertex v by $\alpha_1^v, \dots, \alpha_{d_v}^v$, then the energy function is given by the sum

$$\sum_{v \in V} \sum_{i=1}^{d_v} \left(\frac{2\pi}{d_v} - \alpha_i^v \right)^2.$$

The second energy function is based on the length of the edges. If we denote the mean length of the edges by \bar{e} , then the energy function is given by the sum

$$\sum_{e \in E} (\bar{e} - e)^2.$$

These two energy functions are then used in two different embedders. The first embedder tries to optimise the angles of the fundamental domain in such a way that the energy functions is minimal. If the upper left corner of the fundamental domain is α , then this embedder calculates the energy function for this domain, and then calculates it for the domain with the angle α replaced by the angle $\alpha - \epsilon$ for a given

ϵ . Finally the angle α is changed by the amount given by the difference between the two energy functions times ϵ times a given constant k .

The second embedder tries to optimise the length of the sides of the fundamental way in such a way that the energy function is minimal. If the horizontal side of the fundamental domain is hs , then this embedder calculates the energy function for this domain, and then calculates it for the domain with the horizontal side hs replaced by the horizontal side $hs - \epsilon$ for a given ϵ . Finally the horizontal side hs is changed by the amount given by the difference between the two energy functions times ϵ times a given constant k times a variable factor such that the horizontal side is always at least a given length.

4.6.3 Periodic tilings, cylinders and tori

A cylinder is the surface formed by the points at a fixed distance of a line. If we cut a cylinder open along its axis, we get a rectangle. The other direction works as well: if we identify (read: glue together) two opposite sides of a rectangle we get a cylinder.

A torus is a closed surface that is created by rotating a circle around an axis coplanar with the circle. An example of a torus can be seen in Figure 4.28.

If we cut the torus open along the two circles drawn upon it in Figure 4.28, we get a rectangle (see Figure 4.29). The other way works as well: if we identify (read: glue together) the opposite sides of the rectangle we get a torus. Thus if we have a drawing on the torus and a way to cut it, we also have a drawing in the rectangle and the top and bottom will have to match, as well as the right and left; so we have a rectangle that can be used to build a periodic tiling.

This means that if we have the graph of a periodic tiling we can extract a graph from this that is embedded on a cylinder or a torus.

When we paste several parallelogram-shaped domains of a periodic tiling together in a $m \times n$ configuration ($m, n \in \mathbb{N}$), we get a new parallelogram-shaped domain of a periodic tiling as can be seen in Figure 4.30. When we tile the plane using these ‘supertiles’ we get the same tiling. Even if we introduce an offset when glueing these larger domains together — e.g., glue tile 1 to 2’ instead of to 1’,...—

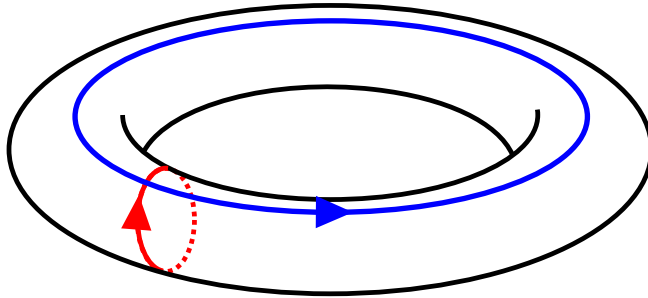


Figure 4.28: A torus with two topologically different fundamental cycles.

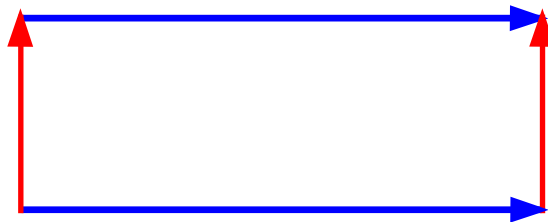


Figure 4.29: A torus cut open and shown as a rectangle.

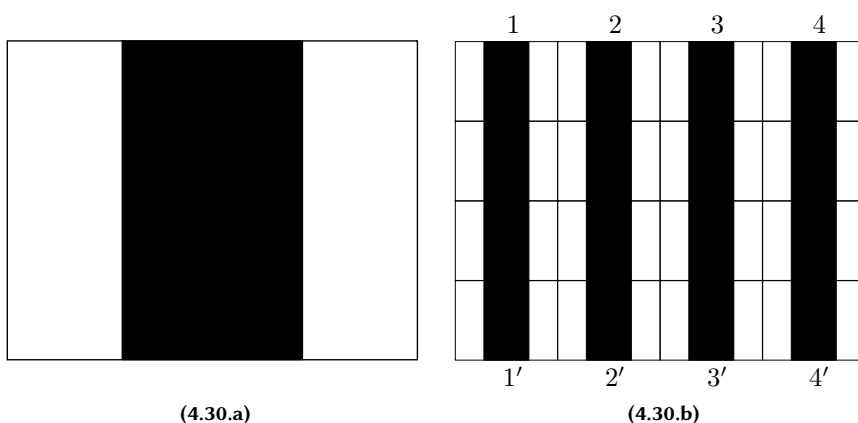


Figure 4.30: A small tile and the 4×4 large tile corresponding to this tile.

we get the same periodic tiling. On the torus and cylinder however we get different tilings. This can easily be seen by looking at Figure 4.30. If we identify the bottom and the top of the rectangle in Figure 4.30.b and let n correspond to n' ($n = 1, \dots, 4$) we get a cylinder with circles. If however we let n correspond to $(n + 1)'$ ($n = 1, \dots, 3$) we get a cylinder with a spiral on it.

4.7 Results

This algorithm was implemented as the program `azu1` and is available from [63]. The site also includes several other related resources such as the visualiser and the catalogue of tilings in several formats.

Including an image of all the tilings in the catalogue would take up a lot of space. It is also difficult to select a sample from the 1274 tilings that is interesting and at the same time small enough to be included in this thesis. We selected the six structures in which the azulenes are all equivalent under a subgroup of the symmetry group that only contains translations. These tilings are shown in Figure 4.31.

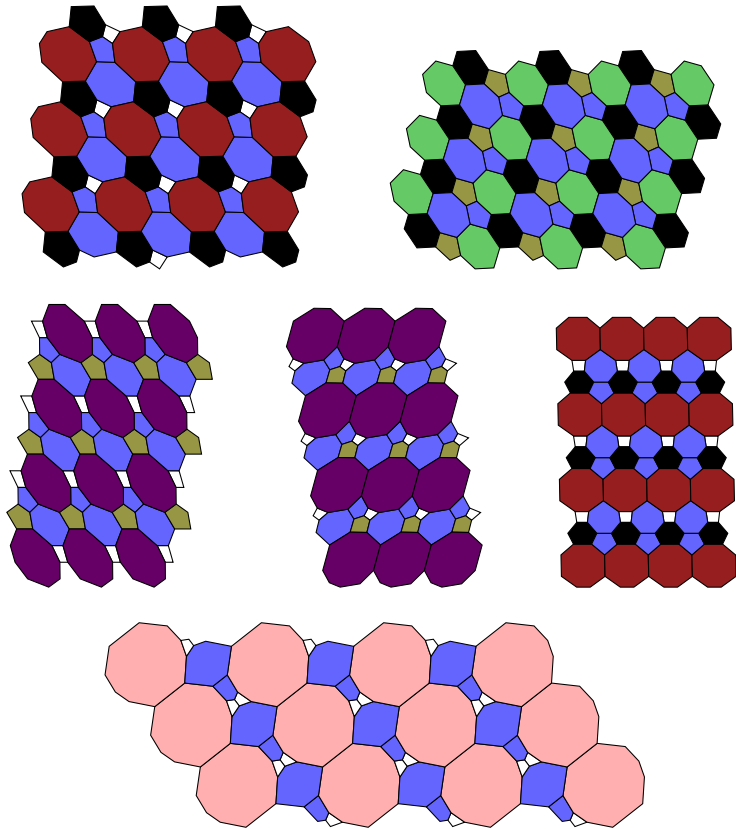
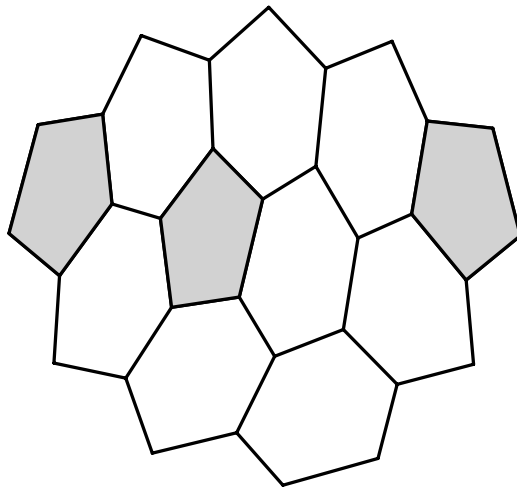


Figure 4.31: *The six structures where all the azulenes are equivalent under a group of translations.*

5

Pseudo-convex patches



5.1 Definitions

We re-use many of the definitions established in [40].

1,5-patch

Definition 5.1.1 A **1,5-patch** is a finite, bridgeless, plane graph with three kinds of faces: 1 "outer" face with unrestricted size, 1 to 5 pentagons and an unrestricted number of hexagons. Furthermore all the vertices have degree 3 except some of the vertices of the outer face which have degree 2. In this thesis we use just the word patch to mean 1,5-patch. The **boundary** of the patch is formed by the vertices and edges of the outer face.

boundary

The degrees of the boundary vertices in the order of the cycle form a cyclic sequence of 2's and 3's. If this sequence does not contain any consecutive 3's, the patch is called **pseudo-convex**.

pseudo-convex

A boundary edge that contains two vertices of degree 2 is called a **break-edge**. \diamond

break-edge

When drawing a patch as a graph and giving its boundary sequence we always choose a clockwise direction for the boundary sequence. The patch corresponding to counterclockwise direction is of course isomorphic to the patch corresponding to the clockwise direction.

When writing down the boundary sequence we will denote repetitive parts by superscripts. We use $(x_1 \dots x_n)^k$ as an abbreviation for

$$\underbrace{(x_1 \dots x_n) \dots (x_1 \dots x_n)}_{k \text{ times}}.$$

We will sometimes write a superscript 1. This is to emphasise certain structures or groupings in the boundary. An example of this notation can be seen in the caption of Figure 5.1.

Lemma 5.1.2 In a pseudo-convex patch the number of break-edges is equal to $6 - p$, where p is the number of pentagons in the patch.

This result follows easily from the Euler formula. See e.g., [47] for a proof of this lemma.

Moving to a higher level of abstraction, we can view pseudo-convex patches as polygons where the corners are formed by the break-edges, and the lengths of the

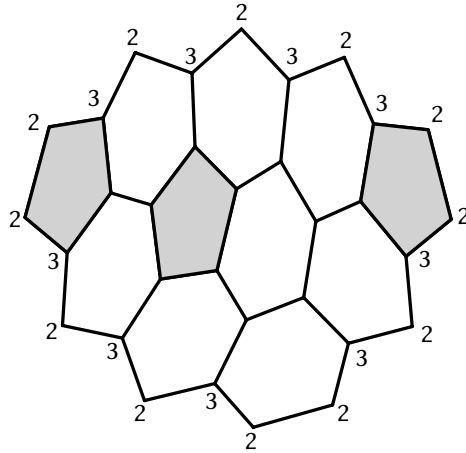


Figure 5.1: A patch with boundary $2(23)^2 2(23)^3 2(23)^4$

sides are determined by their number of 3's. This gives us yet another representation of a boundary. We can just give the cyclic sequence of lengths of the sides of this polygon (once again in clockwise direction). The $(6 - p)$ -tuple (n_1, \dots, n_{6-p}) of a boundary of a patch which is lexicographically smallest among all its cyclic permutations is called the **boundary vector** of that patch.

**boundary
vector**

5.2 Constructing pseudo-convex patches

In [54] an algorithm is described that can generate non-isomorphic patches for an arbitrary (i.e., not necessarily pseudo-convex) boundary. In that general case, however, it is difficult to predict which faces can have a disconnected intersection with the boundary, which makes the generation for this general case more difficult and less efficient.

In [40] it is shown that all pseudo-convex patches can be unwound in a spiral from the outside starting at a break-edge. This is done by selecting a break-edge and removing the face incident to that edge. Then the algorithm proceeds to remove faces

outer spiral

along the boundary in a clockwise direction. Figure 5.2 shows an example of these **outer spirals**. This implies that for a given pseudo-convex boundary and a fixed break-edge where the spiral must start, the pseudo-convex patch can be encoded in a string of constant length at most 5. This is because there are at most 5 pentagons in the patch and all other faces are hexagons. Therefore we only need to record the positions of the pentagons in the outer spiral. For a pseudo-convex patch with p pentagons we can encode the spiral as a string $[s_1, \dots, s_p]$ with s_1 the number of hexagons before the first pentagon and, for $2 \leq i \leq p$, s_i is the number of hexagons between the $(i - 1)$ -th and the i -th pentagon. For the examples in Figure 5.2 these encodings are, respectively, $[0, 4, 4]$, $[3, 3, 2]$ and $[0, 3, 4]$; so a single pseudo-convex patch might have different outerspiral codes depending on which break-edge is chosen. If one does not distinguish between mirror images (and we usually do not want to do this), there are even more different outerspiral codes because different directions for the spiral must also be taken into account.

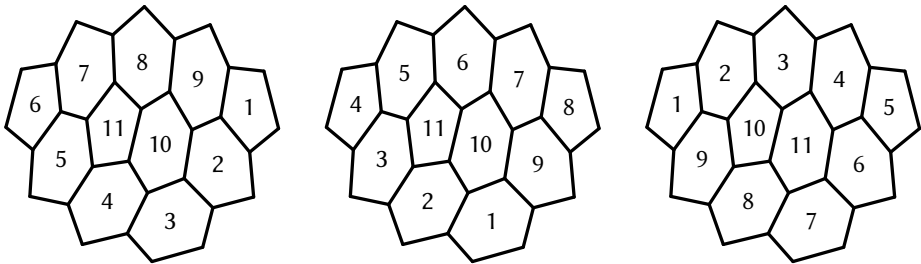


Figure 5.2: The three possible outer spirals in the pseudo-convex patch from Figure 5.1.

marked patch
marked boundary
marked break-edge
mark

Definition 5.2.1 A **marked patch** is an embedded pseudo-convex patch together with a fixed break-edge in its boundary.

A **marked boundary** is a boundary together with a fixed break-edge.

The fixed break-edge of a marked patch, respectively of a marked boundary, is called the **marked break-edge** of that marked patch, respectively of that marked boundary.

The **mark** of a marked patch or of a marked boundary, is the second (in clockwise

direction) vertex of the marked break-edge.

The **boundary sequence** of a marked patch or a marked boundary is the sequence of lengths of the sides starting with the side after the marked break-edge and proceeding in clockwise direction. **boundary sequence** \diamond

In a marked patch it makes sense to speak of the outer spiral of that marked patch. In that case we mean the unique outer spiral that starts with the face incident to the mark and then proceeds in clockwise direction.

In [40], it is also explained that while removing faces in this manner we intermediately obtain different pseudo-convex patches. Namely, if we start from a marked pseudo-convex patch and remove faces we will have a new pseudo-convex patch the moment that we remove a pentagon or the moment we remove a hexagon with a break-edge different from the mark. This allows us to define a successor for a marked patch.

Definition 5.2.2 Given a marked patch P with a boundary sequence different from $l, 0, 0, l, 0, 0$ and from $l, 0, l, 0, 0$ with $l \in \mathbb{N}$. The **successor** $s(P)$ of P is the marked patch that is obtained by removing faces along the boundary in clockwise direction starting with the face containing the mark until for the first time a pentagon or a face with a break-edge different from the marked break-edge is removed and by choosing the second vertex (in clockwise direction on the boundary) of the first break-edge (in clockwise direction starting from the old mark) in the new patch as the mark. **successor** \diamond

The successor of a patch is always defined except when all faces are removed to get to the next step in the outer spiral.

The algorithm we will discuss here is basically the same as the algorithm described in [41, 40] to solve the PentHex Puzzles (note that in [41, 40], this algorithm is not used: the authors of [41, 40] use the inverse of the algorithm in order to generate all solutions of PentHex Puzzles with any given number of hexagons). The difference from the algorithm as it is described in [41], lies in the implementation. We will give the algorithm in terms of the outer spiral code and the boundary sequence, which are both relatively short sequences of integers. Given a boundary sequence and a partial outer spiral, the operations we use, give a longer partial outer spiral and the boundary sequence of a possible successor. After applying such a step the algorithm

recursively tries to fill the new boundary sequences until a boundary sequence is obtained for which it can easily be determined whether there is no patch with this boundary, or, if there is such a patch, to list all all patches with that boundary. By enumerating the pseudo-convex patches in this way instead of by immediately generating the graph, we gain a lot of speed. Note also that for a program that uses these pseudo-convex patches by reading them through standard input or from a file, the time to transform an outer spiral code into an internal graph representations is – depending on the internal data structure used – not necessarily much more than to decode a standard graph format.

5.3 Extending the spiral

While filling the pseudo-convex patch we are dealing with two objects: an outerspiral code and a boundary sequence. Each operation takes a pair of these objects and maps them to a new pair. The rules for this mapping will be described later. When filling a pseudo-convex patch one side at a time, we allow two possible operations. The first operation is filling the first side completely with hexagons. We will denote this operation with \mathcal{H} . The second operation is adding a pentagon to the first side after i hexagons. In the non-degenerate case this means that $0 \leq i \leq s$ where s is the length of the first side. This operation will be denoted by $\mathcal{P}(i)$.

Our ultimate goal is to determine all the possible outerspiral codes that fill the original boundary of the pseudo-convex patch. To achieve this goal we consecutively apply combinations of operations \mathcal{H} and $\mathcal{P}(i)$ and thus rewrite the boundary sequence while growing the outerspiral code. We do not act directly on the outerspiral code, but use what we call an extended spiral code, which also documents the separate steps in the construction. The reason for using this data structure will become clear once we explain the canonicity check. An extended spiral is a sequence with elements from $\mathbb{N} \cup \{P\}$. The numbers record the number of hexagons added to the outer spiral and the P 's record the positions of the pentagons in the outer spiral. The outerspiral code can be obtained from an extended spiral code by summing up the numbers which are not separated by a P . E.g., the extended spiral code

$(3)(P)(3)(P)(1)(1)(P)$ corresponds to the outerspiral code $[3, 3, 2]$.

The boundary sequence is a sequence s_1, \dots, s_l with $s_i \in \mathbb{N}(1 \leq i \leq l)$. In this, l is the length of the boundary sequence.

The operations transform an extended spiral and a boundary into a longer extended spiral and a shorter remaining boundary to be filled ('shorter' means that the sum of the numbers is smaller). The fact that the remaining boundary decreases proves that this algorithm always terminates, because each side length must be at least 0.

5.3.1 The algorithm

The algorithm recursively applies the operations until the obtained boundary sequence $R = (s_1, \dots, s_l)$ encodes a single pentagon or has length 6. In the second case the partial outerspiral code encodes a pseudo-convex patch if and only if R encodes a pseudo-convex patch with only hexagons. This can be tested in constant time by testing whether R describes a closed curve in the hexagonal lattice. This is the case if the following two equations hold

$$2s_1 + s_2 - s_3 - 2s_4 - s_5 + s_6 = 0,$$

and

$$s_1 + 2s_2 + s_3 - s_4 - 2s_5 - s_6 = 0.$$

In the case when R encodes a single pentagon or a patch with 6 break-edges, it has to be tested whether the representation of the patch is canonical. This is done by computing all alternative representations of the patch and comparing these. As will be explained later on, this can be done without really building the patch in memory.

However, as will also be explained later on, we can already build the alternative representations during the construction and often already decide very early in the construction process whether the outer spiral that is being built will be canonical or not.

		length of the boundary				
		5	4	3	2	1
side lengths	$1 \neq s_2 \neq 0 \neq s_l$	5.3.3			5.3.4.2	5.3.4.1
	$s_2 = 1, s_l \neq 0$				5.3.5.2	N/A
	$s_2 = 0$	5.3.5.1	5.3.5.1	5.3.5.3	5.3.5.2	N/A
	$s_l = 0$	5.3.5.1	5.3.5.1	5.3.5.3	5.3.5.2	N/A
	$s_2 = s_l = 0$	5.3.5.1	5.3.5.4	5.3.5.3	5.3.5.2	N/A
	$s_2 = s_l = s_{l-1} = 0$	5.3.5.5	5.3.5.4	5.3.5.3	N/A	N/A

Table 5.1: An overview of the different cases for the operations to fill a pseudo-convex boundary. The non-degenerate case is shaded in white, the first class of degenerate cases is shaded in light grey, and the second class of degenerate cases is shaded in dark grey. For each case the corresponding paragraph is listed.

5.3.2 Overview of the cases

As will be explained in the following pages, there are several degenerate cases for the construction operations. This leads to more implementation work, but does not affect the efficiency of the technique.

Table 5.1 gives an overview of the different cases and the paragraphs in which they are discussed. In Subsection 5.3.3, the non-degenerate case is discussed. In this general case either a complete side of hexagons is added to the marked boundary, or a number of hexagons followed by a pentagon is added to the marked boundary. This leads to a new boundary for which most sides are the same as the previous boundary except for the side to which the faces were added and the side before and after that one. A first class of degenerate cases corresponds to those cases where the side before and after the side to which the faces are added coincide. This class of degenerate cases is discussed in Subsection 5.3.4. The calculations for the sides of the new boundary involve some subtractions. A second class of degenerate cases corresponds to those cases where some of the new boundary elements would be

smaller than zero if the rules from the previous cases were applied. This class of degenerate cases is discussed in Subsection 5.3.5.

5.3.3 The non-degenerate case

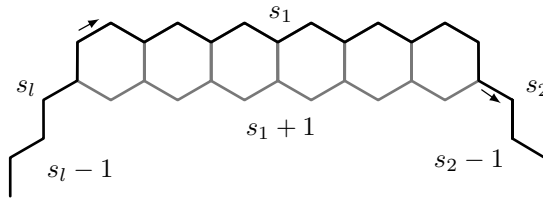


Figure 5.3: The non-degenerate case of the operation \mathcal{H} . The marked break-edge is shown with an arrow. The arrow points towards the mark.

The first operation we consider, is to fill the first side completely with hexagons. Generally speaking, this means $s_1 + 1$ times adding a hexagon. This situation is shown in Figure 5.3. If we had a partial extended spiral $\langle \text{extended spiral} \rangle$ and a boundary sequence s_1, s_2, \dots, s_l before applying this operation, then by applying this operation, we add the number $s_1 + 1$ to the extended spiral and get a new boundary sequence which is equal to $s_2 - 1, \dots, s_l - 1, s_1 + 1$. So we write down this operation as:

$$\begin{array}{ccc}
 \boxed{\begin{array}{c} \langle \text{extended spiral} \rangle \\ s_1, s_2, \dots, s_l \end{array}} & \xrightarrow{\mathcal{H}} & \boxed{\begin{array}{c} \langle \text{extended spiral} \rangle (s_1 + 1) \\ s_2 - 1, \dots, s_l - 1, s_1 + 1 \end{array}} \quad (\text{Fig. 5.3})
 \end{array}$$

The second operation is to place a number of hexagons followed by a pentagon along the first side. If this pentagon is also the last face that is added to this side, then the behaviour is slightly different. Therefore we will consider adding a pentagon after adding s_1 hexagons as a separate operation.

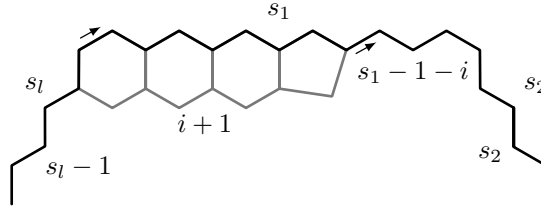


Figure 5.4: The non-degenerate case of the operation $\mathcal{P}(i)$ with $0 \leq i < s_1$. The marked break-edge is shown with an arrow. The arrow points towards the mark.

When we add a pentagon after adding i ($0 \leq i < s_1$) hexagons (see Figure 5.4), we add the number i to the extended spiral followed by a P . We also get a new boundary which is one longer than the previous boundary and which is given by $s_1 - 1 - i, s_2, \dots, s_l - 1, i + 1$. So we write down this operation as:

$$0 \leq i < s_1$$

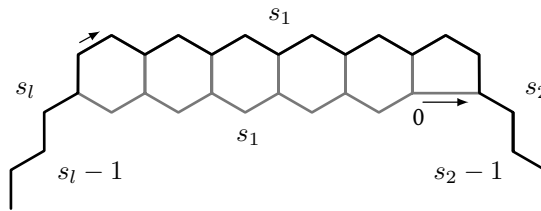
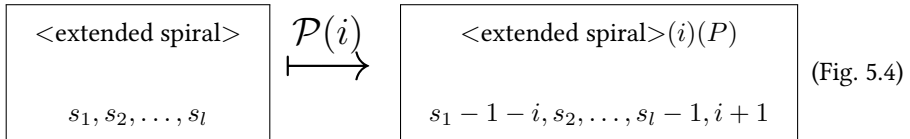


Figure 5.5: The non-degenerate case of the operation $\mathcal{P}(s_1)$. The marked break-edge is shown with an arrow. The arrow points towards the mark.

When we add a pentagon after adding s_1 hexagons (see Figure 5.5), we add the number s_1 to the extended spiral followed by a P . We also get a new boundary which

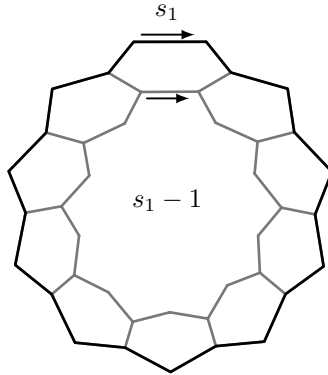


Figure 5.6: The operation \mathcal{H} in case the length of the boundary sequence is 1

is one longer than the previous boundary and which is given by $0, s_2 - 1, \dots, s_l - 1, s_1$. The reason that this operation is different from the $\mathcal{P}(i)$ -case, is that the first element of the new boundary would be smaller than zero if the previous rule was applied. We write this operation down as:

$$\begin{array}{ccc}
 \boxed{\begin{array}{c} \langle \text{extended spiral} \rangle \\ s_1, s_2, \dots, s_l \end{array}} & \xrightarrow{\mathcal{P}(s_1)} & \boxed{\begin{array}{c} \langle \text{extended spiral} \rangle_{(s_1)}(P) \\ 0, s_2 - 1, \dots, s_l - 1, s_1 \end{array}} \quad (\text{Fig. 5.5})
 \end{array}$$

5.3.4 A first class of degenerate cases

In the definitions above, each of these three operations uses up to three elements of the boundary sequence which are not simply copied in order to determine the new boundary sequence. Therefore a first class of degenerate cases might arise if there are less than three elements in the boundary sequence.

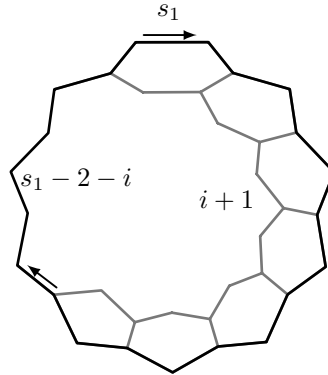
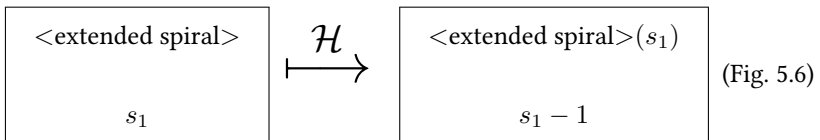


Figure 5.7: The operation $\mathcal{P}(i)$ in case the length of the boundary sequence is 1

5.3.4.1 Boundary length is 1

The situation when adding a side of hexagons is illustrated in Figure 5.6. In this case the side is already completely filled after adding s_1 hexagons. The reason is that the first hexagon lies also at the break-edge at the end of the side, since this is the same break-edge as the break-edge at the beginning of the side. This gives us the following operation:



The possibilities when adding a pentagon after first adding a number of hexagons are shown in Figure 5.7 and Figure 5.8. Note that the case where the pentagon is the last face at the current side, this time corresponds to adding a pentagon after adding $s_1 - 1$ hexagons. This gives us the following two operations:

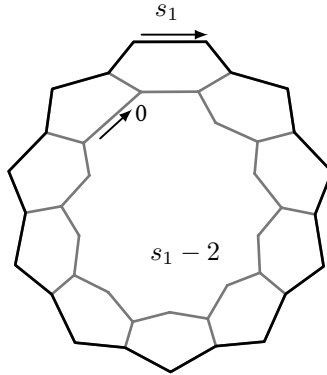
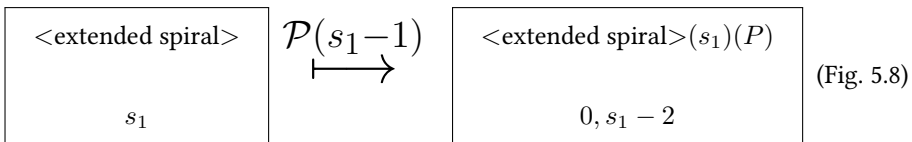
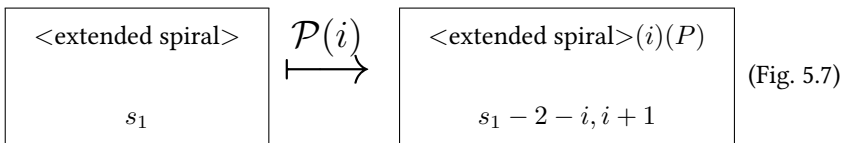


Figure 5.8: The operation $\mathcal{P}(s_1 - 1)$ in case the length of the boundary sequence is 1

$$0 \leq i \leq s_1 - 2$$



5.3.4.2 Boundary length is 2

The situation when adding a side of hexagons is illustrated in Figure 5.9. In this case the side before the side following the mark and the side after the side following the mark are the same. This gives us the following operation:

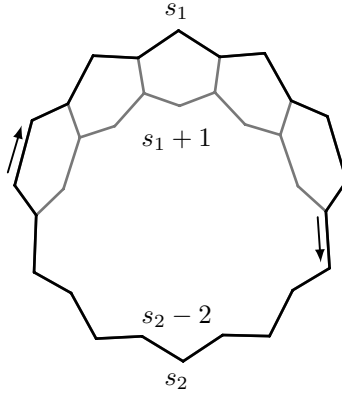
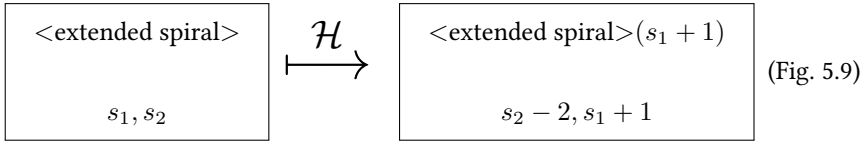
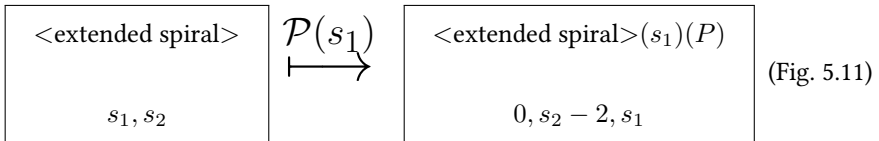
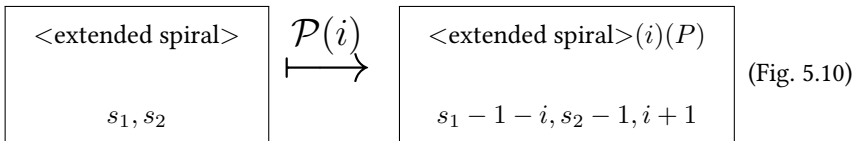


Figure 5.9: The operation \mathcal{H} in case the length of the boundary sequence is 2



The possibilities when adding a pentagon after first adding a number of hexagons are shown in Figure 5.10 and Figure 5.11. This gives us the following two operations:

$$0 \leq i \leq s_1 - 1$$



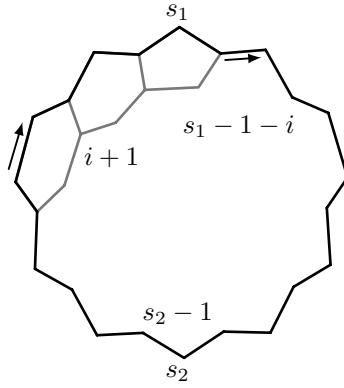


Figure 5.10: The operation $\mathcal{P}(i)$ in case the length of the boundary sequence is 2

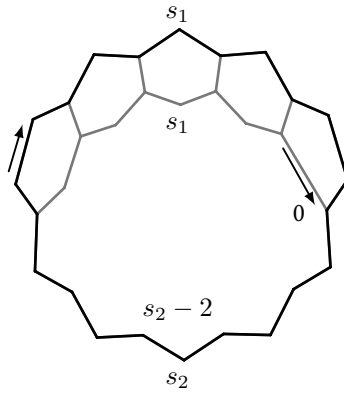


Figure 5.11: The operation $\mathcal{P}(s_1)$ in case the length of the boundary sequence is 2

5.3.5 A second class of degenerate cases

Each of the three operations take some elements of the boundary sequence and increase or decrease them to construct the new boundary sequence. The elements of the boundary sequence are natural numbers so they may increase infinitely but at the other end they are bounded by 0. Therefore a second class of degenerate cases is formed by the cases where some elements of the boundary sequence are less than the amount that will be subtracted from them, when using the previous operations to calculate the new boundary sequence.

Lemma 5.3.1 *There exists no pseudo-convex patch such that a pentagon lies at a side of length 0, except for the following cases:*

- *the boundary has the form $0, l, 0, 0, l$ with $l \in \mathbb{N}$ and the pentagon lies at the first side, or*
- *the boundary has the form $0, l, 0, l$ with $0 < l \in \mathbb{N}$.*

Proof: A pseudo-convex patch with boundary length 6 contains 0 pentagons, so we only need to examine the cases with boundary length < 6 . Assume there exists a patch P with a boundary of the form s_1, s_2, \dots, s_l with $l < 6$ and a pentagon at a side of length 0.

There is no pseudo-convex patch with a boundary of the form 0 or 0, 0, because the first would correspond to a single vertex with a loop and the second would correspond to a digon. Both are not simple graphs. There is no pseudo-convex patch with a boundary of the form 0, 0, 0, resp. of the form 0, 0, 0, 0, since this corresponds to a triangle, resp. a quadrangle. These are not pseudo-convex 1, 5-patches. So assume that P has a boundary different from these cases.

If the boundary of P contains 3 consecutive sides of length 0, then P has 5 consecutive vertices of degree 2 on the boundary. The patch P can only have a pentagon at these sides if the boundary consists of just these 5 vertices, in which case P would have a boundary of the form $0, l, 0, 0, l$ with $l = 0$. A boundary with 3 consecutive sides of length 0 cannot have another side of length 0 that is not adjacent to one of these three sides, since the maximum

length of a boundary is 5. So assume now that the boundary of P does not contain 3 consecutive sides of length 0.

Assume P has a boundary with two consecutive sides of length 0 followed and preceded by a side of length greater than 0 (possibly the same side). This configuration corresponds to 4 consecutive vertices of degree 2 on the boundary followed and preceded by a vertex of degree 3. This means that a face at this sides contains these 6 vertices, so it is a hexagon. Note that it is not possible that the following and preceding vertex of degree 3 are the same. There is an edge pointing inwards at the vertex of degree 3, and so this face at these 5 vertices would be larger than 6.

We have already proven that there exist no pseudo-convex patches with a pentagon that lies at a side of length 0 that is not followed and preceded by sides of length greater than 0, except in the case where the boundary sequence is $0, 0, 0, 0, 0$.

This means that the only cases that we still need to examine are the patches with a pentagon at the first side and with the following boundary sequences: $0, l$, or $0, l, k$, or $0, l, k, m$, or $0, l, k, m, n$.

We first look at the case with boundary sequence $0, l$ with $0 < l \in \mathbb{N}$. We can immediately also assume that $l > 1$, because a patch with boundary sequence $0, 1$ has only one vertex on the boundary with an edge pointing inwards, so this patch has a bridge. Assume the face at the side of length 0 is a pentagon, then the patch we obtain by removing that face has a boundary of the form $0, 0, l - 2$. The face at the sides of length 0, is a hexagon as was already proven. The patch obtained by removing that hexagon has a boundary of the form $0, 0, l - 4$. This argument can be used until the third side length is either 0 or 1. In the first case we have a triangular face and in the second case we have a boundary with one vertex of size 3. Both are not valid structures, so the initial assumption that the face at the side of length 0 was a pentagon was false.

Now we look at the case with boundary sequence $0, l, k$ with $0 < l, k \in \mathbb{N}$. Assume the face at the side of length 0 is a pentagon. The patch we obtain

by removing that face has a boundary of the form $0, 0, l - 1, k - 1$. Again the face at the sides of length 0, is a hexagon as was already proven. The patch obtained by removing that hexagon, has a boundary of the form $0, 0, l - 2, l - k$. This argument continues until there are at least three sides of length 0. If $l = k$ the last face would be a quadrangle, which is not possible. If $l \neq k$, we find a patch with boundary sequence $0, 0, 0, |l - k|$. Such a patch does not exist, since it would contain a face of size > 6 .

Next we look at the case with boundary sequence $0, l, k, m$ with $0 < l, m \in \mathbb{N}$ and $k \in \mathbb{N}$. Assume the face at the first side is a pentagon. The patch we obtain by removing that face has a boundary of the form $0, 0, l - 1, k, m - 1$. We already know that the face at the two consecutive sides of length 0 is a hexagon. The patch we obtain by removing that hexagon, has a boundary of the form $0, 0, l - 2, k, m - 2$. We can repeat this argument, until we obtain a patch with a boundary of the form $0, 0, 0, k, m - l$, or of the form $0, 0, l - m, k, 0$. In both cases such a patch does not exist, since it would contain a face of size > 6 , except in the case where all the side lengths are 0, i.e., when $k = 0$ and $l = m$.

Finally we look at the case with boundary sequence $0, l, k, m, n$ with $0 < l, n \in \mathbb{N}$ and $k, m \in \mathbb{N}$. Assume the face at the first side is a pentagon. The patch we obtain by removing that face has a boundary of the form $0, 0, l - 1, k, m, n - 1$. We already know that the face at the first two consecutive sides of length 0 is a hexagon. The patch we obtain by removing that hexagon, has a boundary of the form $0, 0, l - 2, k, m, n - 2$. We can repeat this argument, until we obtain a patch with a boundary of the form $0, 0, 0, k, m, n - l$, or of the form $0, 0, l - n, k, m, 0$. In both cases such a patch does not exist, since it would contain a face of size > 6 , except in the case where all the side lengths are 0, i.e., when $k = m = 0$ and $l = n$. ■

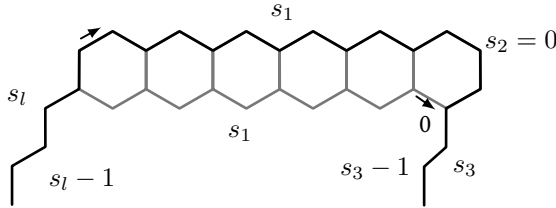
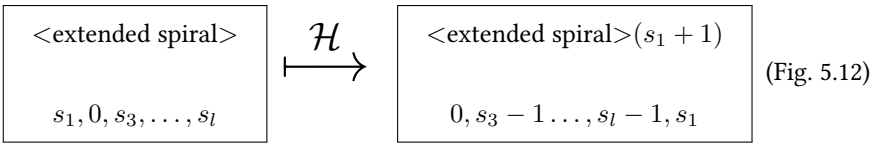


Figure 5.12: The operation \mathcal{H} in the general case of the second class of degenerate cases with $s_2 = 0$

5.3.5.1 The general case of the second class of degenerate cases

If we look at the operations in the non-degenerate case, then we see that there are three possible situations in which we have boundary elements that would be less than zero, if those operations were applied. Those cases are $s_2 = 0$, $s_l = 0$ and $s_2 = s_l = 0$.

We start by looking at the case $s_2 = 0$ and $s_l > 0$. The situation when adding s_1 hexagons to the first side is shown in Figure 5.12. The final hexagon also lies at the side which corresponds to s_3 . This gives us the following operation:



The operation of adding a pentagon after i hexagons with $0 \leq i < s_1$ is not degenerated for this case. A pentagon after s_1 hexagons is not possible due to Lemma 5.3.1 and the fact that $s_l > 0$.

Next we look at the case $s_l = 0$ and $s_2 > 0$. For all operations in this case, the first hexagon also lies at the side which corresponds to s_{l-1} . The situation when adding s_1 hexagons to the first side is shown in Figure 5.13. This gives us the following operation:

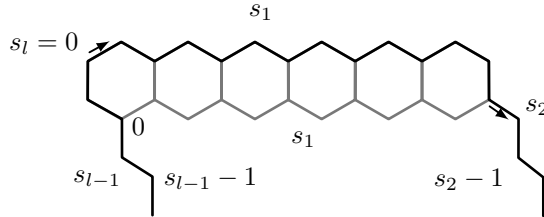


Figure 5.13: The operation \mathcal{H} in the general case of the second class of degenerate cases with $s_l = 0$

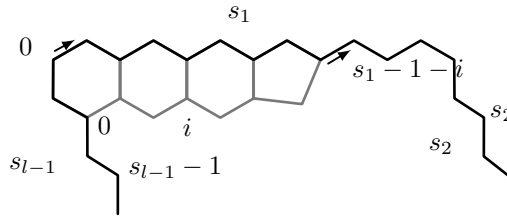
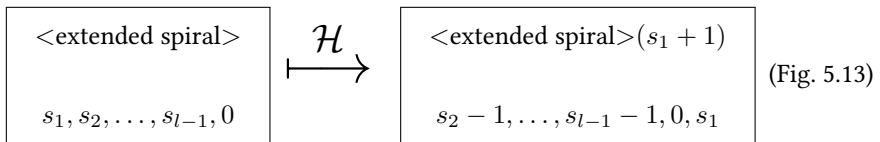


Figure 5.14: The operation $\mathcal{P}(i)$ in the general case of the second class of degenerate cases with $s_l = 0$



The situations when adding a pentagon after i hexagons is shown in Figure 5.14 and Figure 5.15. Note that due to Lemma 5.3.1 and the fact that $s_2 > 0$, adding a pentagon after 0 hexagons will not lead to a pseudo-convex patch. This gives us the following operations:

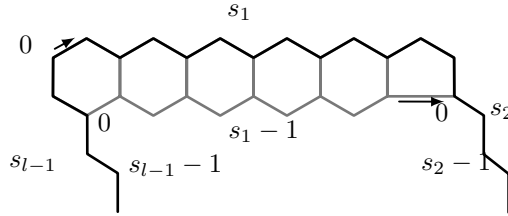


Figure 5.15: The operation $\mathcal{P}(s_1)$ in the general case of the second class of degenerate cases with $s_l = 0$

$$0 < i < s_1$$

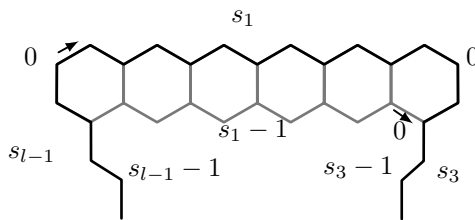
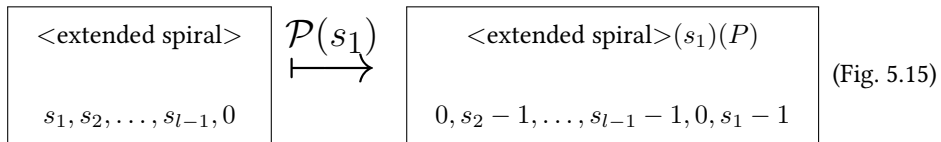
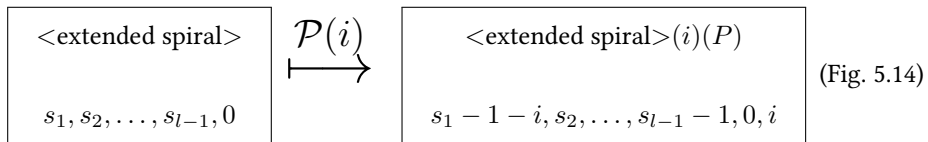


Figure 5.16: The operation \mathcal{H} in the general case of the second class of degenerate cases with $s_2 = s_l = 0$

And finally we come to the case $s_2 = s_l = 0$. The situation when adding s_1 hexagons to the first side is shown in Figure 5.16. The first hexagon also lies at the

side which corresponds to s_{l-1} and the final hexagon also lies at the side which corresponds to s_3 . This gives us the following operation:

$$\begin{array}{ccc}
 \boxed{\begin{array}{l} \langle \text{extended spiral} \rangle \\ s_1, 0, s_3, \dots, s_{l-1}, 0 \end{array}} & \xrightarrow{\mathcal{H}} & \boxed{\begin{array}{l} \langle \text{extended spiral} \rangle_{(s_1 + 1)} \\ 0, s_3 - 1, \dots, s_{l-1} - 1, 0, s_1 - 1 \end{array}}
 \end{array} \tag{Fig. 5.16}$$

The operation of adding a pentagon after i hexagons with $0 < i < s_1$ is the same for the case $s_l = 0$ and $s_2 > 0$ and the case $s_2 = s_l = 0$. Due to Lemma 5.3.1, in this case adding a pentagon after 0 hexagons will lead to a pseudo-convex patch if the boundary length is 4 and $s_3 = s_1 > 0$, or the boundary length is 5, $s_3 = 0$ and $s_4 = s_1$. These cases will be handled in respectively 5.3.5.4 and 5.3.5.5.

Due to Lemma 5.3.1, adding a pentagon after s_1 hexagons will theoretically lead to a pseudo-convex patch if the boundary length is 4 and $s_3 = s_1 > 0$, or the boundary length is 5, $s_4 = 0$ and $s_3 = s_1$. These cases will be handled in respectively 5.3.5.4 and 5.3.5.5.

The cases where s_{l-1} is equal to 0 degenerate further. These cases will be handled separately for each length of the boundary in the following paragraphs.

5.3.5.2 Boundary length is 2

As was already used in the proof of Lemma 5.3.1, there exists no patch with boundary sequence 0,0, since this patch would just be a digon. By examining the operations in the first class of degenerate cases with boundary length equal to 2, we find that those operation further degenerate when $s_2 = 0$ and also, in case of the operation of adding a side of hexagons and the operation of adding a pentagon after s_1 hexagons, when $s_2 = 1$.

We start with the case $s_2 = 0$ and first look at the operation \mathcal{H} . Here we note that the same happens as in the first class of degenerate cases with the case where the boundary length is 1. The first hexagon also lies at the end of the side that corresponds to s_1 . Therefore this operation here corresponds to adding s_1 hexagon. This situation is shown in Figure 5.17. This gives us the following operation:

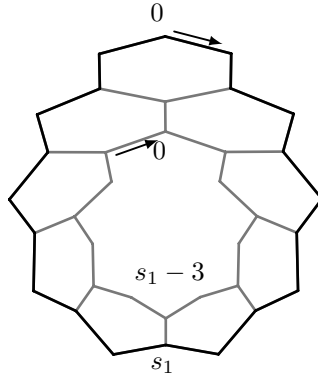
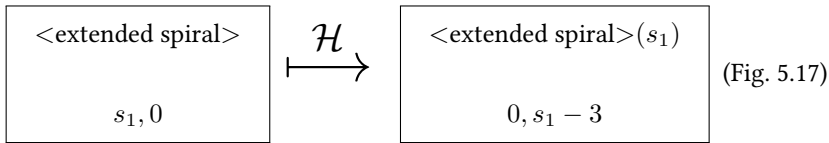


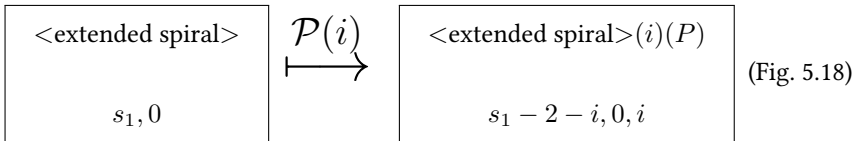
Figure 5.17: The operation \mathcal{H} in the case length = 2 of the second class of degenerate cases with $s_2 = 0$



In this case there are only s_1 faces that have a non-empty intersection with the boundary. Therefore it is not possible to add a pentagon after s_1 hexagons. The s_1 th face cannot be a pentagon either, since this face contains at least 6 vertices, as can be seen in Figure 5.17.

The situation for $\mathcal{P}(i)$ with $0 \leq i \leq s_1 - 2$ is shown in Figure 5.18. Also in this case the first hexagon also lies at the end of the side that corresponds to s_1 . This gives us the following operation:

$$0 \leq i \leq s_1 - 2$$



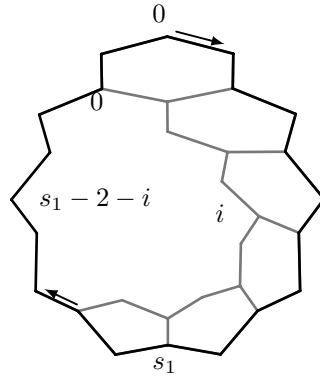


Figure 5.18: The operation $\mathcal{P}(i)$ in the case length = 2 of the second class of degenerate cases with $s_2 = 0$

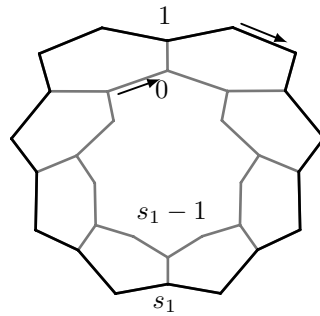


Figure 5.19: The operation \mathcal{H} in the case length = 2 of the second class of degenerate cases with $s_2 = 1$

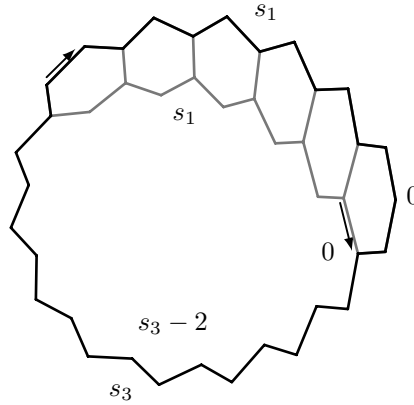
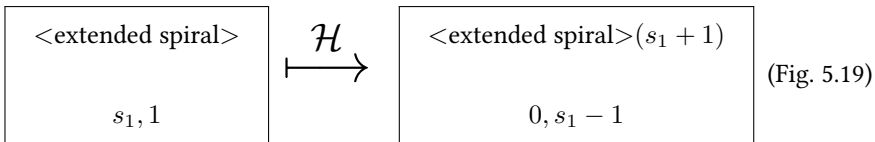


Figure 5.20: The operation \mathcal{H} in the case length = 3 of the second class of degenerate cases with $s_2 = 0$

Next we look at the case $s_2 = 1$. The situation for operation \mathcal{H} is shown in Figure 5.19. In this case the first and the last hexagon both lie at the side that corresponds to s_2 . This gives the following operation.



The operation $\mathcal{P}(i)$ with $0 \leq i \leq s_1 - 1$ is not further degenerated for this case. A pentagon after s_1 hexagons is not possible, since also in this case the last face contains at least 6 vertices, as is illustrated in Figure 5.19.

5.3.5.3 Boundary length is 3

In this section we will discuss the cases where the general case of the second class of degenerate cases degenerate further due to overlap of the boundary elements that are used to build the new boundary in the case where the boundary length is 3.

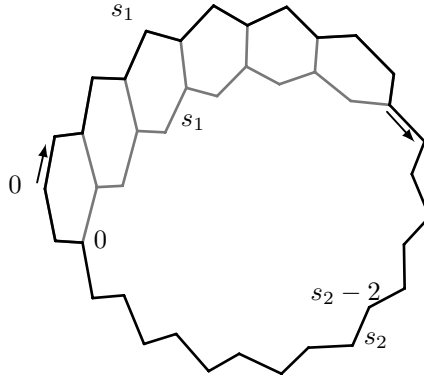
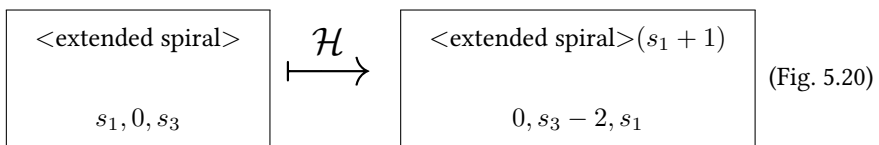


Figure 5.21: The operation \mathcal{H} in the case length = 3 of the second class of degenerate cases with $s_3 = 0$

We start with the case $s_2 = 0$. The only operation we had for this case in 5.3.5.1 was \mathcal{H} . The only difference with the situation there, is that in this case the final hexagon lies also at the side corresponding to s_3 (see Figure 5.20). This gives the following operation:



Next we turn our attention to the case $s_3 = 0$. The only two operations that further degenerate in this case are \mathcal{H} and $\mathcal{P}(s_1)$. For both operations the further degeneration stems from the fact that there is a face which lies at the beginning of the side corresponding to s_2 and a face which lies at the end of the side corresponding to s_2 . These situations are shown in Figure 5.21 and Figure 5.22. This leads to the following operations:

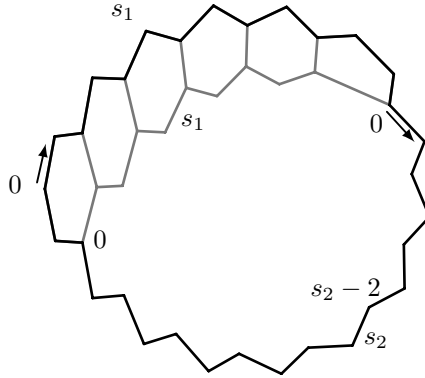


Figure 5.22: The operation $\mathcal{P}(s_1)$ in the case length = 3 of the second class of degenerate cases with $s_3 = 0$

$\langle \text{extended spiral} \rangle$ $s_1, s_2, 0$	$\xrightarrow{\mathcal{H}}$	$\langle \text{extended spiral} \rangle_{(s_1 + 1)}$ $s_2 - 2, 0, s_1$	(Fig. 5.21)
---	-----------------------------	---	-------------

$\langle \text{extended spiral} \rangle$ $s_1, s_2, 0$	$\xrightarrow{\mathcal{P}(s_1)}$	$\langle \text{extended spiral} \rangle_{(i)}(P)$ $0, s_2 - 2, 0, s_1$	(Fig. 5.22)
---	----------------------------------	---	-------------

The case $s_2 = s_3 = 0$ is not possible, since there exists no pseudo-convex patch with boundary $s_1, 0, 0$. If $s_1 = 0$, then the pseudo-convex patch would be a triangle. If $s_1 = 1$, then the pseudo-convex patch would have a bridge. If $s_1 > 1$, then the face at the two sides of length 0 is a hexagon, since it contains at least six vertices. The patch obtained by removing that face, has boundary $s_1 - 2, 0, 0$. By repeating this argument, we would eventually find a pseudo-convex patch with either boundary $0, 0, 0$ or boundary $1, 0, 0$, depending on the parity of s_1 .

Note that there is no further degeneration for these cases. Adding a side of hexagons to the side corresponding to s_1 in a boundary of the form $s_1, 0, 1$ or a

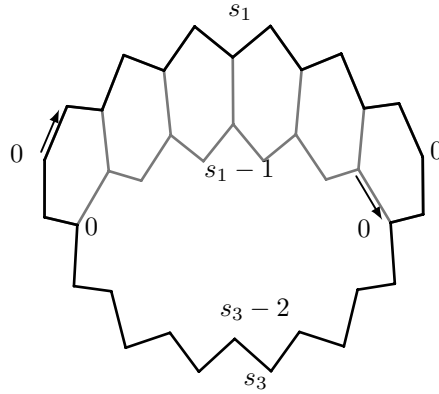


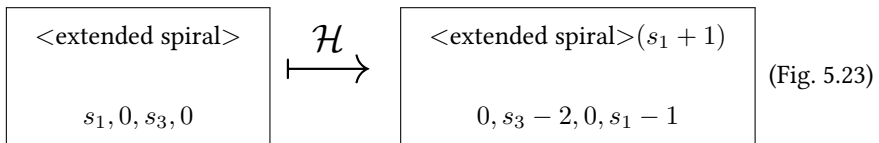
Figure 5.23: The operation \mathcal{H} in the case length = 4 of the second class of degenerate cases with $s_2 = s_4 = 0$

boundary of the form $s_1, 1, 0$ is not possible and adding a pentagon after s_1 hexagons in a boundary of the form $s_1, 1, 0$, because in all these cases, the last face cannot be placed.

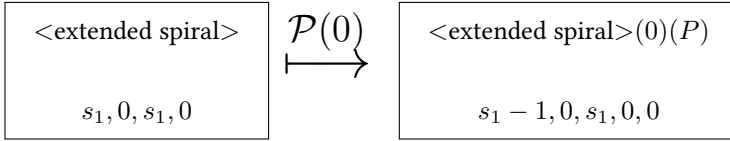
5.3.5.4 Boundary length is 4

In this section we will discuss the cases where the general case of the second class of degenerate cases degenerate further due to overlap of the boundary elements that are used to build the new boundary in the case where the boundary length is 4. The only case that deviates from the general case in 5.3.5.1 is $s_2 = s_4 = 0$.

The situation when adding hexagons to the side of length s_1 is shown in Figure 5.23. The first and the last hexagon both lie at the side of length s_3 . This gives the following operation:



Due to Lemma 5.3.1 applying operation $\mathcal{P}(0)$ to a boundary sequence $s_1, 0, s_3, 0$ will lead to a new boundary sequence that is the boundary of a pseudo-convex patch if and only if $s_1 = s_3$. The operation in this case is:



Let P be a pseudo-convex patch with a boundary of the form $l, 0, l, 0$ with $l > 0$. Assume the face at one of the sides of length 0 is a pentagon. The patch obtained by removing that pentagon has a boundary of the form $l - 1, 0, l - 1, 0, 0$. Due to Lemma 5.3.1, the face at the two consecutive sides of length 0 is a hexagon (unless $l - 1$ is 0), and the patch obtained by removing that hexagon has a boundary of the form $l - 2, 0, l - 2, 0, 0$. The same reasoning can be repeated, until we finally have a patch with a boundary of the form $0, 0, 0, 0, 0$. This patch contains one bounded face, and that face is a pentagon. So we find that for a patch with a boundary of the form $l, 0, l, 0$ with $l > 0$, if one of the faces at a side of length 0 is a pentagon, then also the other face at a side of length 0 is a pentagon. Therefore applying operation f_∞ to a boundary sequence $s_1, 0, s_1, 0$ will not lead to a pseudo-convex patch, since a hexagon is placed at one of the sides of length 0.

5.3.5.5 Boundary length is 5

The only cases that further degenerate for this boundary length, are those cases where $s_4 = s_5 = 0$, and those cases where $s_2 = s_5 = 0$ and a pentagon is placed at a side of length 0. Due to Lemma 5.3.1, this last situation won't lead to a pseudo-convex patch if the boundary is not of the form $l, 0, l, 0, 0$. So in all cases that still need to be handled, the boundary contains two consecutive sides of length 0.

Let P be a pseudo-convex patch with a boundary that contains two consecutive sides of length 0: $k, l, m, 0, 0$ with $k, l, m \in \mathbb{N}$. If $k = l = m = 0$, then P contains only one bounded face and that face is a pentagon. There is no pseudo-convex patch with boundary length smaller than or equal to 5 and three consecutive sides of length

0 and at least one other side with length greater than 0, since the three consecutive sides of length 0 preceded or followed by a side of length greater than 0, correspond to at least 5 boundary vertices of degree 2 preceded or followed by a boundary vertex of degree 3, which would mean that there is a face with size greater than 6. So assume that k and m are greater than 0. Due to Lemma 5.3.1, the face at the two consecutive sides of length 0 is a hexagon. The patch obtained by removing that face has a boundary of the form $k - 1, l, m - 1, 0, 0$. The same reasoning can be repeated, until we obtain a patch with boundary length 5 and at least three consecutive sides of length 0. However, we already showed that there is only one such patch, that it is a patch with one bounded face and that that face is a pentagon. So we find that $k = m$ and $l = 0$, and that P is up to isomorphism the only patch with that boundary.

5.4 Avoiding isomorphic copies

A representation of a pseudo-convex patch consists of a marked boundary together with an outer spiral. *In principle* we generate all possible representations of the pseudo-convex patches but accept a representation only if it is the canonical representation. A representation of a pseudo-convex patch is canonical if the boundary sequence in clockwise direction starting with the mark is equal to the boundary vector of the boundary and the outer spiral is lexicographically largest for all such possible marks.

There are several optimisations possible. E.g., we do not need to generate a representation that does not start at a break-edge which leads to the boundary vector. In many cases this will already reduce the number of possible starting points to one. We can also try to eliminate other starting points during the generation process, when we can determine that an outer spiral starting from those marks is lexicographically smaller than the current outer spiral being built, or we can abort the current generation branch and backtrack, when we can determine that an outer spiral from one of the alternative marks is lexicographically larger than the current outer spiral being built for all structures that will be constructed in this branch. Once we only have

one possible starting point we can continue with the generation without performing any checks for canonicity, since each subsequent structure in this branch will be canonical.

The following section will discuss at which points in the generation process we are able to determine parts of the outer spirals from the alternative starting points.

5.5 Splitting the patches into shells

A pseudo-convex patch is shellable, i.e., it can be subdivided into disjoint shells. But let us first define what we mean by shells.

Definition 5.5.1 *Given a pseudo-convex patch P , the **outer shell** or first shell of **outer shell** P is the set of bounded faces of P which have a non-empty intersection with the boundary of P .* \diamond

Definition 5.5.2 *Given a pseudo-convex patch P , the **n th shell** ($n > 1$) is the outer **n th shell** shell of the patch that is obtained by removing the first, second, ..., $(n - 1)$ th shell of P .* \diamond

This implies that we can see a pseudo-convex patch as a sequence of shells.

We observe that the subdivision of the patch into shells is independent of which outer spiral is used to describe the patch. Also, since the outer spiral is constructed by removing faces along the boundary in a clockwise direction, the faces in the outer spiral will be grouped together by shell, i.e., each outer spiral will first contain all the faces of the first shell, then all faces of the second shell, ... This means that we can construct the partial outer spiral for all alternative starting points up to the same point. This can be done without constructing the partial patch in memory: the extended spiral contains the necessary information to determine the partial outer spirals.

It is also not necessary to always compute the partial outer spiral starting from the break-edges on the initial boundary. If an outer spiral starting from an alternative starting point is lexicographically smaller than the current outer spiral being built, then that starting point is eliminated. If an outer spiral starting from an alternative starting point is lexicographically larger than the current outer spiral being built, then

the current branch of the generation process is terminated and we backtrack. So, if the generation process continues with still at least one alternative starting point, we can just remember the break-edges in the new boundary where the alternative partial outer spirals ended, and when the next shell is finished, just compare the part of the partial outer spirals that corresponds to that shell, since the previous parts of the outer spirals will necessarily be equal.

5.6 Isolated pentagons

As was already explained in Chapter 1, the isolated-pentagon-rule (IPR) (p. 9) is an important rule concerning the stability of fullerenes and fullerene-like structures. Therefore special attention is given to allow the generation of pseudo-convex patches with isolated pentagons.

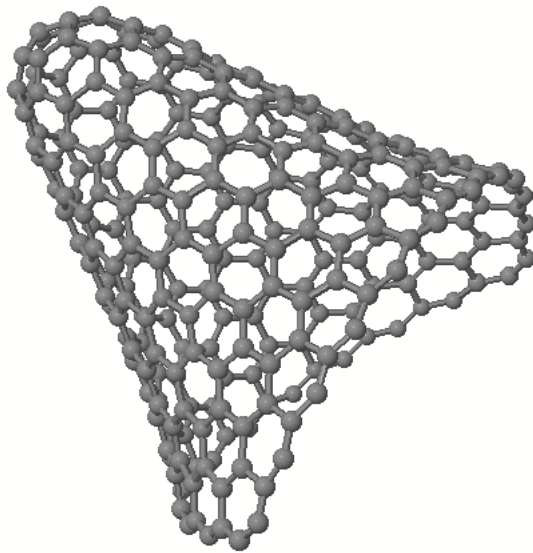
There are only two possible situations in which a face will be placed next to a pentagon. Either the previous face that was placed was a pentagon or either we reached a break-edge that corresponds to a pentagon in a previous shell. So by simply tracking the last face and the face neighbouring each break-edge we can easily prevent patches with neighbouring pentagons to be generated.

5.7 Implementation

The algorithm described in this chapter is not yet implemented as a standalone program. It is however used in the program `cone` described in the next chapter.

6

Nanocones



6.1 Introduction

Nanocoones are carbon networks conceptually situated between graphite and the famous fullerene nanotubes. It is common for all these three that the degree of an atom is 3. Graphite is a planar carbon network where the faces formed are all hexagons. Fullerene nanotubes [37] are discussed in two forms: once as the finite, closed version where, except for hexagons, there are only 12 pentagons and once as the one-side infinite version where 6 pentagons bend the molecule so that an infinite tube with constant diameter is formed. A nanocone lies between graphite and the one-side infinite fullerene nanotubes: in addition to hexagons it has between 1 and 5 pentagons, so that neither the flat shape of graphite nor the constant diameter tube of the nanotubes can be formed. Recently, the attention of the chemical world in nanocoones has strongly increased. Figure 6.1 shows an overview of these 3 types of carbon networks.

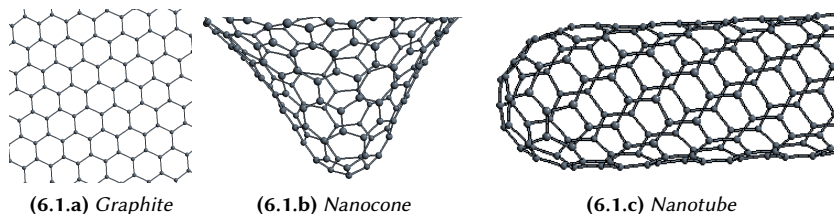


Figure 6.1: *Three types of carbon networks*

The structure of graphite is uniquely determined, but for nanotubes and nanocoones an infinite variety of possibilities exist. Algorithms to generate all combinatorially possible molecules in a given class have long been important tools in chemistry (see e.g., [11][16]). They are e.g., used to detect energetically possible or even optimal bonding structures.

There already exist fast algorithms to generate representatives of fullerene nanotubes (see [45]). These representatives of the infinite nanotubes consist of a cap containing all pentagons that can be uniquely extended to form the infinite nanotube

for which it stands. Since graphite is unique, the generation problem is trivial for this case. Our goal is now to classify the nanocones and develop a generator for them.

Nanocones were already classified by D.J. Klein in [46] and [52]. The result was established independently by C. Justus in [53]. We will give a new and simple proof of the classification result that can easily be generalised for other (cone-like) classes of infinite graphs. We will also subdivide each of the eight (infinite) classes in an infinite number of finite classes which also take the localization of the pentagons into account. These finer classes will form the basis for the generator. To the best of our knowledge, no generators for nanocones existed so far.

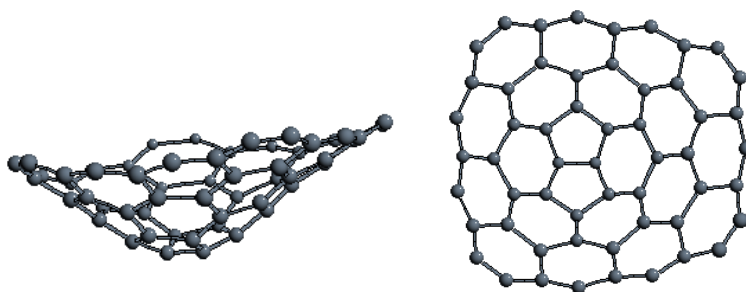


Figure 6.2: Two views of a patch with two pentagons.

The first step however is to translate these chemical molecules into combinatorial structures.

Definition 6.1.1 A **cone graph** is an infinite, cubic, 3-connected, plane graph with **cone graph** $1 \leq p \leq 5$ pentagonal faces and all remaining faces hexagonal. \diamond

6.2 Cone Patches

For computer generation of these structures, they must first be described in a finite way. We describe the infinite molecule by a unique finite structure from which the cone can be reconstructed. For this, consider a cycle in the cone graph. By the Jordan curve theorem a cycle divides the cone graph into two parts. One of those

will be finite, the other will be infinite. The finite part is called the inside of the cycle and the infinite part the outside. A piece of the cone graph can be cut out of the cone by considering only the vertices that are on the cycle or inside the cycle. We re-use much of the definitions established in [40] and already used in the previous chapter.

cone patch

Definition 6.2.1 A **(cone) patch** is a finite, 2-connected, planar graph with three kinds of faces: 1 "outer" face with unrestricted size, 1 to 5 pentagons and an unrestricted number of hexagons. Furthermore all interior vertices have degree 3. The vertices of the outer face have degree 2 or 3. The **boundary** of the patch are the vertices and edges of the outer face.

boundary

◇

When drawing a cone graph and giving its boundary sequence we always choose a clockwise direction for the boundary sequence. By choosing the counterclockwise direction we would obtain similar results.

Remark 6.2.2 Let G be a cone graph and let C be a cycle in G , such that all the pentagons lie inside the cycle. The finite subgraph G_C of the cone graph G formed by the vertices and edges that lie inside or on the cycle is a (cone) patch.

All the vertices in a cone have degree 3, so the vertices that were inside the cycle still have degree 3 and the vertices that were on the cycle have degree 2 or 3. The boundary of the patch corresponds to the vertices and edges of the cycle.

The subgraph G_C is 2-connected because its faces are cycles with the property that if they share a vertex, then they share an edge.

Definition 6.2.3 Given a cone patch P , if we encode the boundary of a cone patch as a cyclic sequence of 2's and 3's according to the degree of the vertices on the boundary, then we call it:

symmetric boundary

- a **symmetric boundary** if the encoding has the form

$$(2(23)^k)^{6-p} (k \in \mathbb{N}, 1 \leq p \leq 5);$$

near-symmetric boundary

- a **near-symmetric boundary** if the encoding has the form

$$(2(23)^{k-1})(2(23)^k)^{6-p-1} (k \in \mathbb{N}, 1 < p < 5).$$

◇

The reason to exclude $p = 1$ in case of a near-symmetric boundary will become clear later (see p. 265).

We can view the boundary of pseudo-convex cone patches as polygons where the corners are formed by the break-edges, and the lengths of the sides are determined by their number of 3's. This gives us yet another representation of a boundary. We can just give the cyclic sequence of lengths of the sides of this polygon (once again in clockwise direction). A symmetric boundary $(2(23)^k)^{6-p}$, for instance, can then be written as the cyclic $(6-p)$ -tuple (k, \dots, k) . The $(6-p)$ -tuple (n_1, \dots, n_{6-p}) of a boundary of a cone patch which is lexicographically smallest among all its cyclic permutations is called the **boundary vector** of that cone patch.

**boundary
vector**

This means that the boundary vector for a symmetric patch is the $(6-p)$ -tuple (k, \dots, k) , and for a near-symmetric patch it is the $(6-p)$ -tuple $(k-1, k, \dots, k)$.

6.3 Families of Cone Patches

Definition 6.3.1 *Given a cone patch P , the set of all faces with a non-empty intersection with the boundary of P is said to form a **layer** when the subgraph in the dual graph induced by these faces is a simple cycle.* \diamond

layer

Lemma 6.3.2 *In a layer no face contains two or more consecutive break-edges unless the layer contains only three faces.*

Proof: Suppose first that the layer contains a pentagon with two consecutive break-edges. Then it is clear that this pentagon can only have one neighbouring face. Therefore it corresponds to a vertex of degree one in the dual and thus an induced subgraph containing this vertex can never be a simple cycle.

Next suppose that the layer contains a hexagon with more than two consecutive break-edges. In this case it can only have at most one neighbouring face, and thus, again, an induced subgraph in the dual containing the vertex corresponding to this face can never be a simple cycle. If however the hexagon has exactly two consecutive break-edges, then it can have two neighbouring faces. These two neighbouring faces, however, share an edge and thus the subgraph induced by these three faces in the dual, is a triangle. This means that there

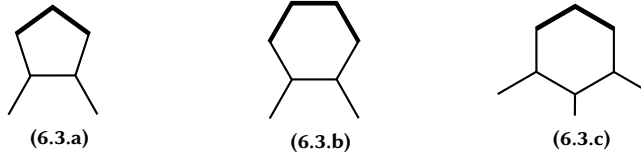


Figure 6.3: There are no consecutive break-edges in a layer with more than three faces.

cannot be any other faces in the layer, because a simple cycle only contains a triangle when it is a triangle. ■

Lemma 6.3.3 *Given a pseudo-convex cone patch P with a layer of only hexagons. A new pseudo-convex cone patch P' can be obtained from P by removing the layer of only hexagons. If P has the boundary vector*

$$(n_1, \dots, n_m),$$

then P' has the boundary vector

$$(n_1 - 1, \dots, n_m - 1).$$

Proof: In Figure 6.4 the boundary and the outer layer is shown for a cone patch with three pentagons and at least one layer of only hexagons. If we look at the relation between the boundary of the original patch and the boundary of the patch obtained by removing the layer of hexagons, we see that in the old patch the new 2's are neighbours of the old 3's and the new 3's come from the old 2's that lie between two 3's on the old boundary. Furthermore the rotational order around the patch is preserved, so each side contains one 3 less. ■

Above we have described the removal of a layer of only hexagons. We will use the symbol ρ for this operation. So the notation

$$P' = \rho P,$$

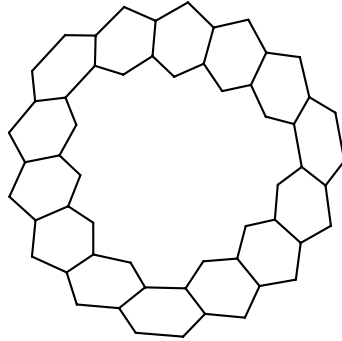


Figure 6.4: The boundary of a cone patch with 3 pentagons and a layer of hexagons. The cone patch that is obtained by removing the layer of hexagons is omitted in the picture. The degrees of the boundary vertices in the inner patch can be easily derived from the degrees of the inner cycle: a vertex of degree 2 corresponds to a vertex of degree 3 and vice versa.

means that P' can be obtained from P by the removal of a layer of only hexagons. The addition of such a layer is defined as the inverse operation of ρ , and thus will be written as ρ^{-1} .

Corollary 6.3.4 *Given a pseudo-convex cone patch P with boundary vector*

$$(n_1, \dots, n_m),$$

then $\rho^{-1}P$ has the boundary vector

$$(n_1 + 1, \dots, n_m + 1).$$

Extending the notation, we will write ρ^{-a} with $a \in \mathbb{N}$ to mean the addition of a layers of hexagons.

These operations enable us to define the following relation between two pseudo-convex cone patches P_1 and P_2 :

$$P_1 \sim_F P_2 \iff \exists k \in \mathbb{Z} : P_1 \cong \rho^k P_2.$$

Note that k is in \mathbb{Z} , and thus can be negative.

Lemma 6.3.5 *The relation \sim_F is an equivalence relation.*

Proof: This follows from the fact that the operation ρ can be inverted in a unique way. ■

These last two lemmas justify the following definitions.

Definition 6.3.6 *Given a pseudo-convex cone patch with boundary vector (n_1, \dots, n_{6-p}) . The **boundary signature** of that cone patch is given by the $(6-p)$ -tuple $(0, n_2 - n_1, \dots, n_{6-p} - n_1)$.*

*An equivalence class associated with the equivalence relation \sim_F is called a **family of patches**. Due to Lemma 6.3.3 the boundary signature is an invariant of a family of patches.*

*A family of patches is said to be a **canonical family** if it has a symmetric or a near-symmetric boundary. ◇*

The boundary vector (n_1, \dots, n_{6-p}) is lexicographically smallest among all its cyclic permutations, and thus for each i with $1 \leq i \leq 6-p$, we have that $n_1 \leq n_i$. This means that the elements of the boundary signature are all elements of \mathbb{N} . A symmetric boundary has the boundary signature $(0, \dots, 0)$ and a near-symmetric boundary has the boundary signature $(0, 1, \dots, 1)$.

Definition 6.3.7 *Given a family of cone patches \mathcal{F} , we define an order on the elements of this family: for all cone patches $P_1, P_2 \in \mathcal{F}$, we say that $P_1 \leq_F P_2$ if and only if there exists a subgraph in P_2 that is isomorphic to P_1 . ◇*

Lemma 6.3.8 *For each family of cone patches \mathcal{F} : the order \leq_F is a total order.*

Proof: This follows from the fact that the operation ρ shortens the boundary and can be inverted in a unique way. ■

Corollary 6.3.9 *Each family of cone patches contains a unique smallest non-empty member.*

Proof: The cone patch P' obtained from a given cone patch P by the removal of a layer of only hexagons contains less vertices than P . A cone patch is a finite

boundary
signature
family of
patches
canonical
family

graph and so the removal operation cannot be repeated infinitely. Since \leq_F is a total order this means that we have a unique smallest member in each family. The only thing that needs to be proven is that this smallest member is non-empty. This is however quite trivial since a cone patch contains at least one pentagon, and this pentagon cannot be removed by the removal operation.

■

Definition 6.3.10 *Given a family of patches, the **canonical representative** of the family is the unique smallest member of this family.* ◇ **canonical representative**

Corollary 6.3.11 *Given a canonical family \mathcal{F} of cone patches, the canonical representative of this family contains a pentagon that has a non-empty intersection with the boundary.*

Proof: A cone patch P is the unique smallest member of a family \mathcal{F} if and only if the operation ρ can not be applied to P . There are three reasons why this might be the case. The first is that the layer contains a pentagon. The second is that one of the components of the boundary vector is 0. It can easily be shown that for a canonical family the latter is never the case without also the former applying. The third reason is that there is no layer, but it was already shown that in this case either one of the first two also applies.

If we have a symmetric boundary, then the boundary vector will be the $6 - p$ zero-vector. This means that the patch is a $6 - p$ -gon. In case $p = 1$ this is a pentagon, and thus the patch indeed contains a pentagon with a non-empty intersection with the boundary. In the other cases ($2 \leq p \leq 5$) there exists no patch with this boundary, because the patch would correspond to, respectively, a quadrangle, a triangle, a cycle of length 2 and a single vertex of degree 2.

In case of a near-symmetric boundary, we have three possibilities. When $p = 2$ the boundary is $2(23)^0(2(23)^1)^3$. Due to Lemma 6.3.2 the face carrying the two break-edges is a hexagon (and otherwise we would have proven this case). Suppose one of the other faces is a hexagon (see Figure 6.5). This would mean that there is a bridge in the patch. Therefore the patch only contains two more

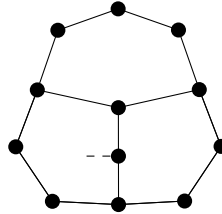


Figure 6.5: A pseudo-convex patch with boundary $2(23)^0(2(23)^1)^3$ cannot have two hexagons in the boundary.

faces, and these are pentagons. When $p = 3$, the boundary is $2(23)^0(2(23)^1)^2$. We consider the face F that contains the three vertices corresponding to the three consecutive 2's. If the vertices corresponding to the two 3's are adjacent to the same vertex inside the patch, then also in this case there is a bridge in the patch, since there are no other ingoing edges into the patch. If the vertices corresponding to the two 3's are adjacent to different vertices inside the patch, then F will have a size of at least 7, which is not possible. So in this case it is not possible that one number in the boundary vector becomes 0. Finally when $p = 3$, the boundary is $2(23)^0(2(23)^1)^1$, which is a contradiction with the 2-connectedness of a cone patch, because the edge that does not lie on the boundary and is incident to the vertex of degree 3 on the boundary is a bridge.

■

6.3.1 Classification of nanocones

We will now classify the different types of nanocones by using a theory developed for the study of disordered tilings.

We first need to define when two disordered tilings — or, more generally, two infinite plane graphs — are equivalent.

Definition 6.3.12 Given two infinite plane graphs $C = (V, E)$ and $C' = (V', E')$. C and C' are called equivalent if there are finite subsets $V_0 \subset V$ and $V'_0 \subset V'$ so

that the graphs $C[V \setminus V_0]$ and $C'[V' \setminus V'_0]$ induced by $V \setminus V_0$, resp. $V' \setminus V'_0$ are isomorphic. \diamond

The previously mentioned method to classify disordered tilings was devised in [19], extended in [28] and was completely solved in [39]. A tiling \mathcal{T}' is a disordering of a periodic tiling \mathcal{T} if we can map the chamber system of \mathcal{T}' onto the Delaney-Dress symbol of \mathcal{T} such that the actions of σ_i (with $0 \leq i \leq 2$) and the values of m_{ij} (with $0 \leq i < j \leq 2$) are respected on the complement of a finite set. We just mention the main result from [39] here. It would not be trivial to introduce the prerequisites for this theorem, but we will translate it to our specific case.

Theorem 6.3.13 (Balke [39]) *A disordered periodic tiling \mathcal{T}' of an equivariant periodic tiling \mathcal{T} is up to equivalence characterised by*

- *the corresponding equivariant periodic tiling \mathcal{T}*
- *a winding number*
- *a conjugacy class of an automorphism in the symmetry group of \mathcal{T}*

For details and proofs on how the invariants correspond to the tilings \mathcal{T} and \mathcal{T}' , see [39]. Here we will give a short sketch of the constructions for the special case of the hexagonal tiling discussed here:

In a hexagonal tiling one can have vertex or face disorders. Vertex disorders are vertices with degree different from 3 and face disorders are faces with size different from 6. A disordered hexagonal tiling is a hexagonal tiling with a finite number of vertex and face disorders.

First, choose a closed path in clockwise direction around all disorders repeating the starting edge at the end. This path is described by right and left turns. Then choose an arbitrary directed starting edge e in the hexagonal lattice and follow the same path. The last (directed) edge e' of this path will (in general) no longer be identical to e , but there is an automorphism γ of the tiling mapping e to e' . This is trivial in the case of the hexagonal tiling but in fact also true in more complicated cases. The conjugacy class of γ in the symmetry group of the hexagonal tiling is the invariant in the third point of Theorem 6.3.13. Furthermore γ can be written as the product

of automorphisms obtained as paths around the single disorders – the pentagons. These automorphisms can easily be determined as a rotation by 60 degrees in counterclockwise direction around the center of a face. If there are only p pentagons as disorders, γ is the product of p rotations by 60 degrees around centers of faces.

In our case of only one face size and only disordering faces with smaller size, the winding number is a direct consequence of the automorphism, so it does not give extra information and we can neglect it here. In the case of the hexagonal lattice it would only be needed to distinguish e.g., between a disorder by a 5-gon and a disorder by an 11-gon which would correspond to the same conjugacy class.

Two rotations belong to the same conjugacy class if and only if they have the same angle of rotation and the centers of rotation are equivalent under a symmetry of the tiling – that is in our case: they are both centers of edges, both centers of faces or both vertices.

This gives us only a limited number of possibilities for these symmetries. They are all rotations and are depicted in Table 6.1. The patches in Table 6.1 are example patches that correspond to these symmetries and therefore also prove existence of such cones.

In [39] the existence of disordered tilings for a given parameter set is proven in a very general context. Nevertheless this result can not be applied here, because we have very special requirements for the disordered tilings: we do not want an arbitrary disordered tiling for this parameter set, but one with only face disorders and these disorders are $1 \leq p \leq 5$ pentagons.

In the previous text we have only considered nanocones containing hexagons and up to five pentagons. This technique however can also be used for other cases. We will give a short example by allowing quadrangles forming cones instead of pentagons as disorders of the hexagonal tiling to illustrate that. A cone is formed if there are one or two quadrangles.

In case of one quadrangle we have, as with the one-pentagon-case, a unique structure. Two quadrangles correspond to two rotations of 120° around the center of a face, so that makes 240° . There are two candidate conjugacy classes of symmetries in the automorphism group of the hexagonal tiling. One corresponds to a rotation of

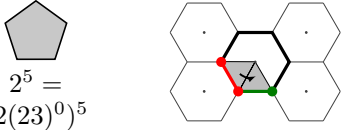
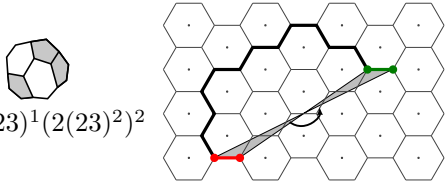
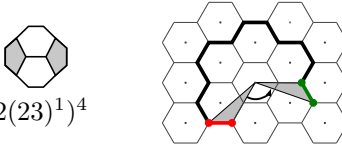
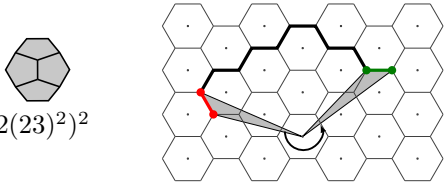
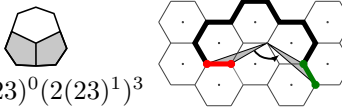
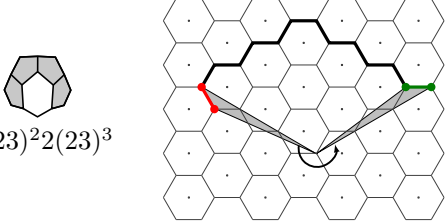
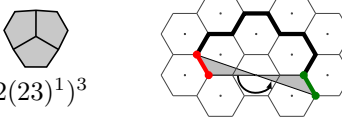
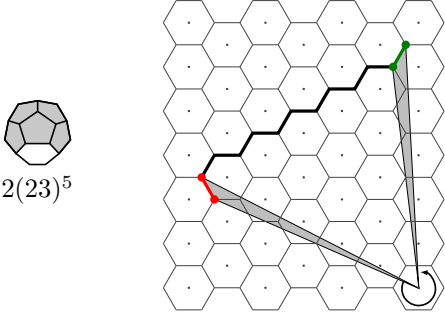
 <p>$2^5 = (2(23)^0)^5$</p>	 <p>$2(23)^1(2(23)^2)^2$</p>
 <p>$(2(23)^1)^4$</p>	 <p>$(2(23)^2)^2$</p>
 <p>$2(23)^0(2(23)^1)^3$</p>	 <p>$2(23)^2(2(23)^3)^3$</p>
 <p>$(2(23)^1)^3$</p>	 <p>$2(23)^5$</p>

Table 6.1: *The complete classification of cone patches.*

240° around the center of a face and one to a rotation of 240° around a vertex. A short investigation however shows that a rotation of 240° around a vertex can not be written as the combination of two rotations of 120° around the center of a face. Let C denote the center of the rotation that is the result of the two rotations r_1, r_2 of 120° around face centers f_1, f_2 . Then C lies on the perpendicular bisector of the segment connecting f_1 and f_2 . Since C is a fixed point of the combination of r_1 and r_2 , the line connecting C to f_1 , resp. f_2 forms an angle of 60° with the segment connecting f_1 and f_2 . Thus the three centers form an equilateral triangle and since f_1 and f_2 lie in the center of faces, so does C , since it is the image of f_1 under a rotation of 60° around f_2 . See also Figure 6.6.

This gives us only one class for the nanocones with two quadrangles. That this class is realizable, can be seen by considering two quadrangles that share an edge. This structure forms a cone patch for these quadrangle cones and corresponds to the boundary $(2(23)^1)^2$.

This example with quadrangles also illustrates the fact that it is not always so that each conjugacy class of symmetries corresponds to an equivalence class of cones under the stricter conditions we use.

Theorem 6.3.14 *For each cone graph $G(V, E)$, there exists a canonical family \mathcal{F} such that for each cone patch $P \in \mathcal{F}$, there exist sets $V_F \subset V$ and $E_F \subset E$ such that the subgraph $F(V_F, E_F)$ of $G(V, E)$ is isomorphic to P .*

In case of $p \in \{1, 5\}$ the family will have a symmetric boundary. In the other cases both symmetric and near-symmetric boundaries are possible.

Proof: We need only prove there exists a subgraph F in the cone graph G such that F is induced by all the vertices in or on a cycle in G ; F is a symmetric or a near-symmetric cone patch and F contains all the pentagonal faces in G . Since the addition or removal of a layer of only hexagons can only be done in a unique way, all the other members of the family will also be contained in G .

Table 6.1 contains an example for each possible equivalence class of automorphisms. Also each patch has either a symmetric boundary or a near-symmetric boundary. Due to the theorem of Balke (Theorem 6.3.13) two disordered tilings that correspond to the same automorphism (and of course the same winding

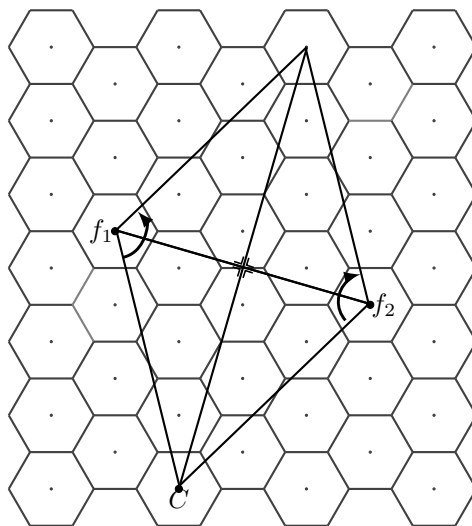


Figure 6.6: The fixed point of the combination of two counter-clockwise rotations of 120° around the centre of a face is always the centre of a face.

number) are equivalent. By Corollary 6.3.4 we can add a number of layers to the patch and the signature stays the same. If we have a cone graph, it must correspond to one of the eight automorphisms in Table 6.1. We can then enlarge the exemplary patch for that automorphism by adding layers of hexagons until the complement of that cone patch in its cone graph is isomorphic to a part of the given cone graph. Since the original patch and thus the enlarged patch was either symmetric or near-symmetric, the given cone contains a symmetric or a near-symmetric family. ■

A cone can be seen as the limit of a family of cone patches with respect to the operation ρ^{-1} .

**canonical
cone patch**

Definition 6.3.15 *Given a cone graph G , a cone patch P of G is a **canonical cone patch** if it is the canonical representative of a canonical family of patches of G .* ◇

Due to the previous theorem each cone has at least one canonical cone patch. The following theorems will prove the unicity of this canonical cone patch, but first we need some more definitions.

**marked
cone patch**

Definition 6.3.16 *A **marked cone patch** P_m consists of a cone patch P and a mark m on one of the edges in the boundary of P . For a marked patch the boundary sequence is no longer cyclic, i.e., it has a fixed starting point: the boundary is given as the sequence of degrees following the marked edge in clockwise direction.*

**canonically
marked
patch**

*A marked cone patch P_m based on a cone patch P is called **canonically marked** when the boundary sequence is lexicographically smallest among its cyclic permutations.*

**canonical
marked
edge**

*The edge carrying the mark in a (canonically) marked patch is called a (/the) **(canonical) marked edge**.* ◇

Since the graphite lattice and the cone graph are both cubic graphs we can describe paths in both these structures by right and left turns. We define the following operations on the set E^* of directed edges:

$$\tau_2 : E^* \rightarrow E^* : e \mapsto \text{right neighbour of } e$$

$$\tau_3 : E^* \rightarrow E^* : e \mapsto \text{left neighbour of } e$$

A path can then be described by a directed start edge e and a sequence of operations τ_2 and τ_3 . The subsequent edges of the path can then be obtained by applying the sequence of operations on the start edge. The names of the operation have been chosen such that the path of the boundary of a cone patch when traversed in clockwise direction has the sequence described by the degrees of the vertices on the boundary in the order they appear.

Theorem 6.3.17 *For each cone graph $G(V, E)$ that is not equivalent to the cone graph corresponding to the near-symmetric cone patch with two pentagons in Table 6.1 c, there exists exactly one canonical family \mathcal{F} of subgraphs of G . In the case that G is equivalent to the cone graph corresponding to the near-symmetric cone patch with two pentagons there can exist at most two canonical families of subgraphs of G .*

Proof: Given a cone C , select two cone patches P_1 and P_2 with a symmetric or a near-symmetric boundary. We need to prove that these two belong to the same family of patches, or in case of a near-symmetric cone with two pentagons they can belong to different families but any other patch will belong to one of both families. For this proof we will again be embedding the patch boundaries in the hexagonal lattice to get a path which identifies the boundary.

A path in the carbon network can be described by a directed start edge and a sequence of τ_2 and τ_3 operations.

We start by making some observations.

Observation 1 *The two patches will have the same boundary signature, because they contain the same number of pentagons and correspond to the same automorphism.*

In this proof we will always consider the patches P_1 and P_2 as subgraphs of the same cone graph associated with C instead of as two separate structures.

Observation 2 *W.l.o.g. we can consider only the case where the boundaries of P_1 and P_2 share at least 1 edge. If this is not the case we can replace the smallest patch P_i by $\rho^{-1}P_i$ and repeat this until this condition is met.*

We choose a directed edge e on the boundary of P_1 , such that e also lies on the boundary of P_2 and the edge following e is different for P_1 and P_2 . If such an edge does not exist we have proven the theorem so assume that such an edge exists.

Split the boundary at the start vertex of e . The first operation in the operation sequences of both paths will differ, because otherwise the edge following e would have been shared by both boundaries. W.l.o.g. we can assume that the sequence of P_1 starts with τ_2 and the sequence of P_2 starts with τ_3 .

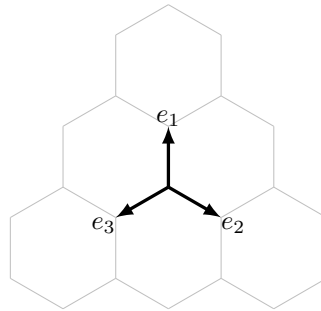


Figure 6.7: Vectors in the hexagonal grid.

We can associate a vector with a directed path in the hexagonal lattice in the following way: we take the vector that starts at the start vertex of the first edge and that ends at the end vertex of the last edge. If we choose an arbitrary point o in the hexagonal lattice, we define the vectors \mathbf{e}_1 , \mathbf{e}_2 and \mathbf{e}_3 with o as starting point as shown in Figure 6.7. By abuse of notation, we will also use \mathbf{e}_1 , \mathbf{e}_2 and \mathbf{e}_3 for the three directed edges that have o as first vertex.

We introduce the following notation for some more vectors (see also Figure 6.8 and Figure 6.9):

$$\begin{aligned} b_0 &= \mathbf{e}_1 & b_3 &= -\mathbf{e}_1 \\ b_1 &= -\mathbf{e}_3 & b_4 &= \mathbf{e}_3 \\ b_2 &= \mathbf{e}_2 & b_5 &= -\mathbf{e}_2 \end{aligned}$$

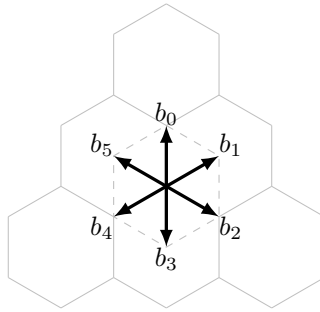


Figure 6.8: Vectors in the hexagonal grid.

and

$$B_i = b_i + b_{i+1} \quad (i \in \{0, \dots, 5\}).$$

Note that index arithmetic on the vectors b_i is done modulo 6.

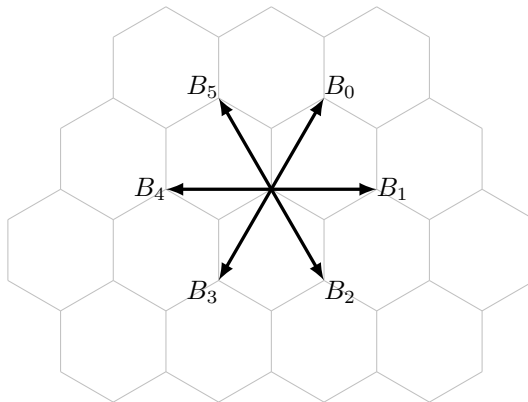


Figure 6.9: Vectors in the hexagonal grid.

A sequence that starts with τ_2 and corresponds to a patch with $6 - p$ break-

edges will result in the following vector when applied to the directed edge e_1 :

$$\sum_{i=0}^{6-p} (B_i n_i + b_i) - b_{6-p},$$

where n_i is the number of three's between the break-edges (see Figure 6.10 for an example).

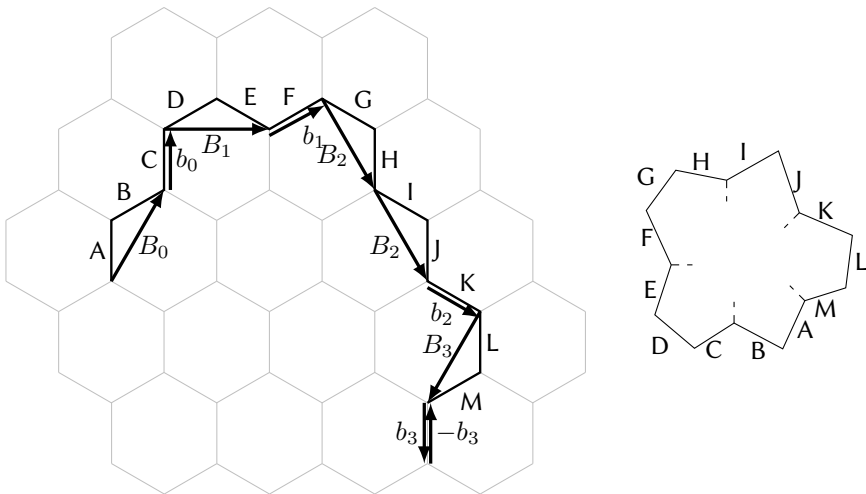


Figure 6.10: The vector corresponding to the boundary 2322322323223 . This is a boundary with 3 break-edges and the values for n_0, n_1, n_2 and n_3 are respectively 1, 1, 2 and 1.

A sequence that starts with τ_3 and corresponds to a patch with $6 - p$ break-edges will result in the following vector when applied to the directed edge e_1 :

$$\sum_{i=0}^{6-p} (B_{i-1} m_i + b_{i-1}) - b_5,$$

where m_i is the number of three's between the break-edges. Note that index arithmetic on the vectors B_i and b_i is done modulo 6. Since we have the extra condition that the sequence must start with τ_3 we have that $m_0 \geq 1$.

The vectors corresponding to boundaries with 5 break-edges are then:

$$(2n_0 + n_1 - n_2 - 2n_3 - n_4 + n_5)\mathbf{e}_1 + (n_0 + 2n_1 + n_2 - n_3 - 2n_4 - n_5 + 1)\mathbf{e}_2, \quad (6.1)$$

and

$$(m_0 + 2m_1 + m_2 - m_3 - 2m_4 - m_5)\mathbf{e}_1 + (-m_0 + m_1 + 2m_2 + m_3 - m_4 - 2m_5 + 1)\mathbf{e}_2. \quad (6.2)$$

The vectors corresponding to boundaries with 4 break-edges are:

$$(2n_0 + n_1 - n_2 - 2n_3 - n_4 + 1)\mathbf{e}_1 + (n_0 + 2n_1 + n_2 - n_3 - 2n_4 + 2)\mathbf{e}_2, \quad (6.3)$$

and

$$(m_0 + 2m_1 + m_2 - m_3 - 2m_4 + 1)\mathbf{e}_1 + (-m_0 + m_1 + 2m_2 + m_3 - m_4 + 2)\mathbf{e}_2. \quad (6.4)$$

The vectors corresponding to boundaries with 3 break-edges are:

$$(2n_0 + n_1 - n_2 - 2n_3 + 2)\mathbf{e}_1 + (n_0 + 2n_1 + n_2 - n_3 + 2)\mathbf{e}_2, \quad (6.5)$$

and

$$(m_0 + 2m_1 + m_2 - m_3 + 2)\mathbf{e}_1 + (-m_0 + m_1 + 2m_2 + m_3 + 2)\mathbf{e}_2. \quad (6.6)$$

The vectors corresponding to boundaries with 2 break-edges are:

$$(2n_0 + n_1 - n_2 + 2)\mathbf{e}_1 + (n_0 + 2n_1 + n_2 + 1)\mathbf{e}_2, \quad (6.7)$$

and

$$(m_0 + 2m_1 + m_2 + 2)\mathbf{e}_1 + (-m_0 + m_1 + 2m_2 + 1)\mathbf{e}_2. \quad (6.8)$$

The vectors corresponding to boundaries with 1 break-edge are:

$$(2n_0 + n_1 + 1)\mathbf{e}_1 + (n_0 + 2n_1)\mathbf{e}_2, \quad (6.9)$$

and

$$(m_0 + 2m_1 + 1)\mathbf{e}_1 + (-m_0 + m_1)\mathbf{e}_2. \quad (6.10)$$

We will now discuss each possible value of p separately. The calculations in this proof have been verified using Sage[66].

- $p = 1$

In this case only a symmetric boundary is possible.

The first path P_1 has the vector given by (6.1). Since

$$n_0 + n_5 = n,$$

$$n_1 = \dots = n_4 = n,$$

this is also equal to

$$-(n_0 + 2n_5)\mathbf{e}_1 + (n_0 - n_5 + 1)\mathbf{e}_2. \quad (6.11)$$

The second path P_2 has the vector given by (6.2). Since

$$m_0 + m_5 = m,$$

$$m_1 = \dots = m_4 = m,$$

this is also equal to

$$(m_0 - m_5)\mathbf{e}_1 + (m + m_0 + 1)\mathbf{e}_2. \quad (6.12)$$

When we combine (6.11) and (6.12) we find the following system of equations:

$$\begin{cases} -n_0 - 2n_5 = m_0 - m_5 \\ n_0 - n_5 = m_0 + m \end{cases} \quad (6.13)$$

By multiplying the second equation by two and subtracting the first equation from it, and, by subtracting the new equation from the second equation, we find that this system is equivalent to

$$\begin{cases} n_0 = m \\ -n_5 = m_0 \end{cases} \quad (6.14)$$

Since n_i and m_i are all positive, we find that $m_0 = 0$, but this contradicts $m_0 \geq 1$, so there is only one path.

- $p = 2$

In this case also a near-symmetric boundary is possible. For a near-symmetric boundary one of the elements in the boundary vector will be one smaller than the others.

The first path P_1 has the vector given by (6.3). Since

$$\begin{aligned} n_0 + n_4 &= n - \delta_4, \\ n_i &= n - \delta_i (\forall 1 \leq i \leq 3), \\ \sum_{i=1}^4 \delta_i &\leq 1 \\ \forall 1 \leq i \leq 4 : \delta_i &\in \{0, 1\} \end{aligned}$$

this is also equal to

$$(2n_0 - n_4 - 2n - \delta_1 + \delta_2 + 2\delta_3 + 1)\mathbf{e}_1 + (n_0 - 2n_4 + 2n - 2\delta_1 - \delta_2 + \delta_3 + 2)\mathbf{e}_2. \quad (6.15)$$

The second path P_2 has the vector given by (6.4). Since

$$\begin{aligned} m_0 + m_4 &= m - \partial_4, \\ m_i &= m - \partial_i (\forall 1 \leq i \leq 3), \\ \sum_{i=1}^4 \partial_i &\leq 1 \\ \forall 1 \leq i \leq 4 : \partial_i &\in \{0, 1\} \end{aligned}$$

this is also equal to

$$(m_0 - 2m_4 + 2m - 2\partial_1 - \partial_2 + \partial_3 + 1)\mathbf{e}_1 + (3m - \partial_1 - 2\partial_2 - \partial_3 + \partial_4 + 2)\mathbf{e}_2. \quad (6.16)$$

When we combine (6.15) and (6.16) we find the following system of equations:

$$\begin{cases} 2n_0 - n_4 - 2n - \delta_1 + \delta_2 + 2\delta_3 = m_0 - 2m_4 + 2m - 2\partial_1 - \partial_2 + \partial_3 \\ n_0 - 2n_4 + 2n - 2\delta_1 - \delta_2 + \delta_3 = 3m - \partial_1 - 2\partial_2 - \partial_3 + \partial_4 \end{cases} \quad (6.17)$$

By subtracting the second equation from the first, we find the following equation

$$-3n_0 - 3n_4 + \delta_1 + 2\delta_2 + \delta_3 - 4\delta_4 = -3m_4 - \partial_1 + \partial_2 + 2\partial_3 - 2\partial_4 \quad (6.18)$$

By looking at this equation modulo 3, we find that

$$(\delta_1 + 2\delta_2 + \delta_3 + 2\delta_4) \bmod 3 = (2\partial_1 + \partial_2 + 2\partial_3 + \partial_4) \bmod 3 \quad (6.19)$$

Since only one δ_i and only one ∂_i can be equal to one, and all the others are equal to zero, we can derive that there are nine possible situations:

$$\delta_1 = 1 \quad \wedge \quad \partial_2 = 1, \quad (6.20)$$

$$\delta_1 = 1 \quad \wedge \quad \partial_4 = 1, \quad (6.21)$$

$$\delta_2 = 1 \quad \wedge \quad \partial_1 = 1, \quad (6.22)$$

$$\delta_2 = 1 \quad \wedge \quad \partial_3 = 1, \quad (6.23)$$

$$\delta_3 = 1 \quad \wedge \quad \partial_2 = 1, \quad (6.24)$$

$$\delta_3 = 1 \quad \wedge \quad \partial_4 = 1, \quad (6.25)$$

$$\delta_4 = 1 \quad \wedge \quad \partial_1 = 1, \quad (6.26)$$

$$\delta_4 = 1 \quad \wedge \quad \partial_3 = 1, \quad (6.27)$$

$$\delta_1 = \delta_2 = \delta_3 = \delta_4 = \partial_1 = \partial_2 = \partial_3 = \partial_4 = 0. \quad (6.28)$$

We first consider possibility (6.20). In this case, system (6.17) reduces to

$$\begin{cases} -n_4 = m_0 \\ n_0 = m \end{cases} \quad (6.29)$$

Since n_i and m_i are all positive, this contradicts $m_0 \geq 1$; so this situation is not possible.

Next we consider possibility (6.21). In this case, system (6.17) reduces to

$$\begin{cases} -n_4 = m_0 + 1 \\ n_0 = m \end{cases} \quad (6.30)$$

Since n_i and m_i are all positive, this contradicts $m_0 \geq 1$; so this situation is not possible.

Next we consider possibility (6.22). In this case, system (6.17) reduces to

$$\begin{cases} -n_4 = m_0 - 1 \\ n_0 = m \end{cases} \quad (6.31)$$

This leads to another solution with $n_0 = n = m$, $n_4 = 0$, $m_0 = 1$ and $m_4 = m - 1$.

Next we consider possibility (6.23). In this case, system (6.17) reduces to

$$\begin{cases} -n_4 = m_0 \\ n_0 = m \end{cases} \quad (6.32)$$

Since n_i and m_i are all positive, this contradicts $m_0 \geq 1$; so this situation is not possible.

Next we consider possibility (6.24). In this case, system (6.17) reduces to

$$\begin{cases} -n_4 = m_0 - 1 \\ n_0 = m - 1 \end{cases} \quad (6.33)$$

This leads to another solution with $n_0 = m_4 = n = m - 1$, $n_4 = 0$ and $m_0 = 1$.

Next we consider possibility (6.25). In this case, system (6.17) reduces to

$$\begin{cases} -n_4 = m_0 \\ n_0 = m \end{cases} \quad (6.34)$$

Since n_i and m_i are all positive, this contradicts $m_0 \geq 1$; so this situation is not possible.

Next we consider possibility (6.26). In this case, system (6.17) reduces to

$$\begin{cases} -n_4 = m_0 \\ n_0 = m - 1 \end{cases} \quad (6.35)$$

Since n_i and m_i are all positive, this contradicts $m_0 \geq 1$; so this situation is not possible.

Next we consider possibility (6.27). In this case, system (6.17) reduces to

$$\begin{cases} -n_4 = m_0 + 1 \\ n_0 = m - 1 \end{cases} \quad (6.36)$$

Since n_i and m_i are all positive, this contradicts $m_0 \geq 1$; so this situation is not possible.

Finally we consider possibility (6.28). In this case, system (6.17) reduces to

$$\begin{cases} -n_4 = m_0 \\ n_0 = m \end{cases} \quad (6.37)$$

Since n_i and m_i are all positive, this contradicts $m_0 \geq 1$; so this situation is not possible.

- $p = 3$

In this case also a near-symmetric boundary is possible. For a near-symmetric boundary one of the elements in the boundary vector will be one smaller than the others.

The first path P_1 has the vector given by (6.5). Since

$$n_0 + n_3 = n - \delta_3,$$

$$n_i = n - \delta_i (\forall 1 \leq i \leq 2),$$

$$\sum_{i=1}^3 \delta_i \leq 1$$

$$\forall 1 \leq i \leq 3 : \delta_i \in \{0, 1\}$$

this is also equal to

$$(2(n_0 - n_3) - \delta_1 + \delta_2 + 2)\mathbf{e}_1 + (n_0 - n_3 + 3n - 2\delta_1 - \delta_2 + 2)\mathbf{e}_2. \quad (6.38)$$

The second path P_2 has the vector given by (6.6). Since

$$\begin{aligned} m_0 + m_3 &= m - \partial_3, \\ m_i &= m - \partial_i (\forall 1 \leq i \leq 2), \\ \sum_{i=1}^3 \partial_i &\leq 1 \\ \forall 1 \leq i \leq 3 : \partial_i &\in \{0, 1\} \end{aligned}$$

this is also equal to

$$(m_0 - m_3 + 3m - 2\partial_1 - \partial_2 + 2)\mathbf{e}_1 + (-m_0 + m_3 + 3m - \partial_1 - 2\partial_2 + 2)\mathbf{e}_2. \quad (6.39)$$

When we combine (6.38) and (6.39) we find the following system of equations:

$$\begin{cases} 2(n_0 - n_3) - \delta_1 + \delta_2 = m_0 - m_3 + 3m - 2\partial_1 - \partial_2 \\ n_0 - n_3 + 3n - 2\delta_1 - \delta_2 = -m_0 + m_3 + 3m - \partial_1 - 2\partial_2 \end{cases} \quad (6.40)$$

If we add the second equation to the first we find that this is equivalent to

$$\begin{cases} n_0 - n_3 + n - \delta_1 = 2m - \partial_1 - \partial_2 \\ n_0 - n_3 + 3n - 2\delta_1 - \delta_2 = -m_0 + m_3 + 3m - \partial_1 - 2\partial_2 \end{cases} \quad (6.41)$$

Since $n_0 + \delta_3 = n - n_3$ and $m_3 + \partial_3 = m - m_0$ we can rewrite this to

$$\begin{cases} 2n_0 - \delta_1 + \delta_3 = 2m - \partial_1 - \partial_2 \\ 2n_0 + 2n - 2\delta_1 - \delta_2 + \delta_3 = 2m_3 + 2m - \partial_1 - 2\partial_2 + \partial_3 \end{cases} \quad (6.42)$$

The first equation in (6.42) leads to

$$(\delta_1 = 1 \vee \delta_3 = 1) \Leftrightarrow (\partial_1 = 1 \vee \partial_2 = 1). \quad (6.43)$$

The second equation in (6.42) leads to

$$(\delta_2 = 1 \vee \delta_3 = 1) \Leftrightarrow (\partial_1 = 1 \vee \partial_3 = 1). \quad (6.44)$$

Putting (6.43) and (6.44) together we find four possibilities:

$$\delta_1 = 1 \quad \wedge \quad \partial_2 = 1, \quad (6.45)$$

$$\delta_2 = 1 \quad \wedge \quad \partial_3 = 1, \quad (6.46)$$

$$\delta_3 = 1 \quad \wedge \quad \partial_1 = 1, \quad (6.47)$$

$$\delta_1 = \delta_2 = \delta_3 = \partial_1 = \partial_2 = \partial_3 = 0. \quad (6.48)$$

We first consider possibility (6.45). In this case system (6.42) reduces to

$$\begin{cases} n_0 = m \\ n = m_3 \end{cases} \quad (6.49)$$

Since $n - n_3 = n_0 = m = m_0 + m_3 = m_0 + n$ we find $n_3 = -m_0$. This contradicts $m_0 \geq 1$.

Next we consider possibility (6.46). In this case system (6.42) reduces to

$$\begin{cases} n_0 = m \\ n - 1 = m_3 \end{cases} \quad (6.50)$$

Since $n - n_3 = n_0 = m = m_0 + m_3 + 1 = m_0 + n$ we find $n_3 = -m_0$. This contradicts $m_0 \geq 1$.

Then we consider possibility (6.47). In this case system (6.42) reduces to

$$\begin{cases} n_0 + 1 = m \\ n = m_3 \end{cases} \quad (6.51)$$

Since $n - n_3 = n_0 + 1 = m = m_0 + m_3 = m_0 + n$ we find $n_3 = -m_0$. This contradicts $m_0 \geq 1$.

Finally we consider possibility (6.48). In this case system (6.42) reduces to

$$\begin{cases} n_0 = m \\ n = m_3 \end{cases} \quad (6.52)$$

Since $n - n_3 = n_0 = m = m_0 + m_3 = m_0 + n$ we find $n_3 = -m_0$. This contradicts $m_0 \geq 1$.

- $p = 4$

In this case also a near-symmetric boundary is possible. For a near-symmetric boundary one of the elements in the boundary vector will be one smaller than the others.

The first path P_1 has the vector given by (6.7). Since

$$\begin{aligned} n_0 + n_2 &= n - \delta_2, \\ n_1 &= n - \delta_1, \\ \sum_{i=1}^2 \delta_i &\leq 1 \\ \forall 1 \leq i \leq 2 : \delta_i &\in \{0, 1\} \end{aligned}$$

this is also equal to

$$(3n_0 - \delta_1 + \delta_2 + 2)\mathbf{e}_1 + (3n - 2\delta_1 - \delta_2 + 1)\mathbf{e}_2. \quad (6.53)$$

The second path P_2 has the vector given by (6.8). Since

$$\begin{aligned} m_0 + m_2 &= m - \partial_2, \\ m_1 &= m - \partial_1, \\ \sum_{i=1}^2 \partial_i &\leq 1 \\ \forall 1 \leq i \leq 2 : \partial_i &\in \{0, 1\} \end{aligned}$$

this is also equal to

$$(3m - 2\partial_1 - \partial_2 + 2)\mathbf{e}_1 + (3m_2 - \partial_1 + \partial_2 + 1)\mathbf{e}_2. \quad (6.54)$$

When we combine (6.53) and (6.54), we find the following system of equations:

$$\begin{cases} 3n_0 - \delta_1 + \delta_2 = 3m - 2\partial_1 - \partial_2 \\ 3n - (2\delta_1 + \delta_2) = 3m_2 - \partial_1 + \partial_2 \end{cases} \quad (6.55)$$

By looking at these equations modulo 3, we find that

$$(2\delta_1 + \delta_2) \bmod 3 = (\partial_1 + 2\partial_2) \bmod 3 \quad (6.56)$$

Since only one δ_i and only one ∂_i can be equal to one, and all the others are equal to zero, we can derive that there are three possible situations:

$$\delta_1 = 1 \quad \wedge \quad \partial_2 = 1, \quad (6.57)$$

$$\delta_2 = 1 \quad \wedge \quad \partial_1 = 1, \quad (6.58)$$

$$\delta_1 = \delta_2 = \partial_1 = \partial_2 = 0. \quad (6.59)$$

First we consider possibility (6.57). In this case, system (6.55) reduces to

$$\begin{cases} n_0 = m \\ n = m_2 + 1 \end{cases} \quad (6.60)$$

Since $n - n_2 = n_0 = m = m_0 + m_2 + 1 = m_0 + n$, we find that $m_0 = -n_2$. This contradicts $m_0 \geq 1$.

Next we consider possibility (6.58). In this case, system (6.55) reduces to

$$\begin{cases} n_0 = m - 1 \\ n = m_2 \end{cases} \quad (6.61)$$

Since $n - n_2 = n_0 + 1 = m = m_0 + m_2 = m_0 + n$, we find that $m_0 = -n_2$. This contradicts $m_0 \geq 1$.

Finally we consider possibility (6.59). In this case, system (6.55) reduces to

$$\begin{cases} n_0 = m \\ n = m_2 \end{cases} \quad (6.62)$$

Since n_i and m_i are all positive, we find that

$$\begin{cases} n_2 = m_0 = 0 \\ n_0 = m_2 = n = m \end{cases} \quad (6.63)$$

Again this contradicts $m_0 \geq 1$ and thus we find that there is only one possible path.

- $p = 5$

In this case only a symmetric boundary is possible.

The first path P_1 has the vector given by (6.9) and the second path P_2 has the vector given by (6.9).

When we combine (6.9) and (6.10), we find the following system of equations:

$$\begin{cases} 2n_0 + n_1 = m_0 + 2m_1 \\ 2n_1 + n_0 = m_1 - m_0 \end{cases} \quad (6.64)$$

which is equivalent to

$$\begin{cases} n_1 = -m_0 \\ n_0 = m_1 + m_0 \end{cases} \quad (6.65)$$

Since n_i and m_i are all positive, we find that

$$\begin{cases} n_1 = m_0 = 0 \\ n_0 = m_1 = n = m \end{cases} \quad (6.66)$$

Again this contradicts $m_0 \geq 1$ and thus we find that there is only one possible path.

We have proven that in all cases except a near-symmetric cone with two pentagons the two patches need to coincide. In the near-symmetric case with two pentagons we found that there are possibly two other possible patches, but it is easily verified that these two correspond to the same family. This proves the theorem. ■

At this point we have a one-to-one correspondence for all nanocones and cone patches except for the near-symmetric case with two pentagons. This case needs some further investigation. The following theorem proves that the two different canonical families in a near-symmetric cone with two pentagons are in fact isomorphic. We will be using facial paths for this, but first some definitions and lemmas are needed.

facial path **Definition 6.3.18** A **facial path** is a sequence f_1, \dots, f_n of n different faces so that $\forall 1 \leq i < n$: face f_i shares an edge with f_{i+1} . \diamond

Note 6.3.19 If a facial path f_1, \dots, f_n is the shortest facial path connecting f_1 and f_n , then for each i with $1 \leq i < n - 1$, face f_i does not share an edge with face f_{i+2} .

It follows from this lemma that if for some $1 < i < n$ the face f_i is a hexagon there are only 3 possibilities for f_{i+1} in a shortest facial path f_1, \dots, f_n . We will denote these 3 possibilities with *right*, *left* and *straight*, as is illustrated in Figure 6.11. Using this terminology we can also give the facial path as $f_1, f_2, d_1, \dots, d_k$ with $d_1, \dots, d_k \in \{\text{left, right, straight}\}$.

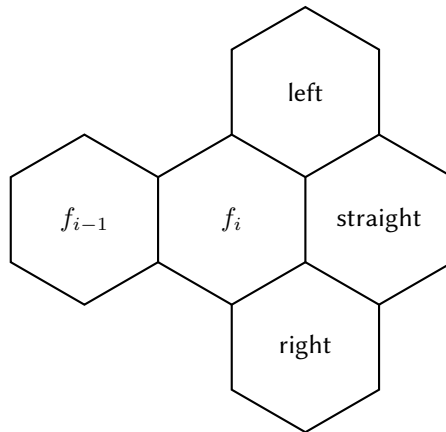


Figure 6.11: The three possible directions in a shortest facial path when arriving in a hexagon.

Another way of thinking about these facial paths is by looking at the dual situation. In the dual the hexagons correspond to vertices of degree 6 and the vertices correspond to triangles. A facial path f_1, \dots, f_n becomes a path v_1, \dots, v_n which we can also identify with a sequence of $n - 1$ directed edges. The concepts of *right*, *left* and *straight* can also be ported to this case. For a directed edge e there are 3 possible next edges which we will denote with $R(e)$, $L(e)$ and $S(e)$. When we look at the 6 directed edges which share the same starting vertex, we see that they form a

cyclic sequence. We will denote the edge before e when this sequence is considered clockwise as $B(e)$ and the edge after e as $A(e)$.

Lemma 6.3.20 *In the dual of a nanocone, for each edge e and for each $n \in \mathbb{N}$: the path corresponding to*

$$e, L(e), S(L(e)), S^2(L(e)), \dots, S^n(L(e)), L(S^n(L(e)))$$

and the path corresponding to

$$B(e), S(B(e)), \dots, S^{n+1}(B(e))$$

have the same start and end vertex when all vertices except possibly the start and end vertex of the first path have degree 6.

Proof: We will prove this by induction on n . The situation for $n = 0$ is illustrated in Figure 6.13.

The edges e and $B(e)$ have the same start vertex by definition of $B(e)$.

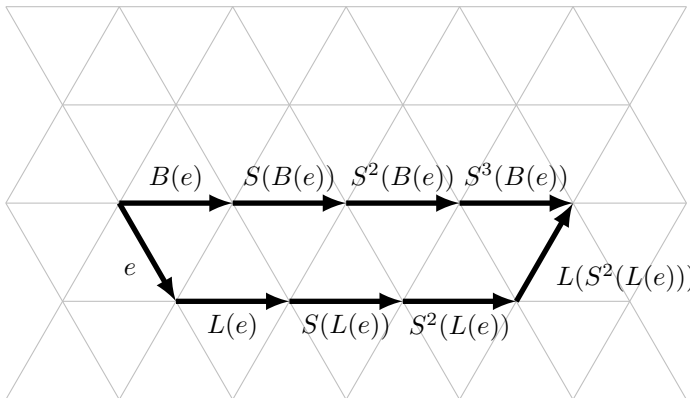


Figure 6.12: *A sequence containing only straight steps and two left steps can always be rewritten as a shorter sequence.*

Since all faces are triangles and the vertex v has degree 6, the path that corresponds to e and the path that corresponds to $B(e)$ followed by the edge

$A(R(B(e)))$ have the same end vertex. Similar argumentations can be used to show that the path that corresponds to $L(e)$ and the path that corresponds to $B(L(e))$ followed by the edge $R(B(e))$ have the same end vertex and that the path that corresponds to $L(L(e))$ and the path that corresponds to $B(L(L(e)))$ followed by the edge $S(B(e))$ have the same end vertex.

Combining these results we find that the path $e, L(e), L(L(e))$ has the same start and end vertex as the path

$$B(e), A(R(B(e))), B(L(e)), R(B(e)), B(L(L(e))), S(B(e)).$$

If we remove consecutive inverse edges in the last path, we find that that path has the same start and end vertex as $B(e), S(B(e))$.

For further n , we just need to repeat the last two detours in the case for $n = 0$.

■

Lemma 6.3.21 *In the dual of a nanocone, for each edge e and for each $n \in \mathbb{N}$: the path corresponding to*

$$e, R(e), S(R(e)), S^2(R(e)), \dots, S^n(R(e)), R(S^n(R(e)))$$

and the path corresponding to

$$A(e), S(A(e)), \dots, S^{n+1}(A(e))$$

have the same start and end vertex when all except possibly the start and end vertex have degree 6.

Proof: Completely analogous to the previous lemma. ■

Corollary 6.3.22 *A shortest path between two given vertices in the dual of a nanocone with all vertices (except possibly the start and end vertex) of degree 6, does not turn in the same direction twice without first turning in the other direction.*

Proof: This is a direct consequence of the previous two lemmas since the new sequence in these two lemmas is always one shorter than the original sequence.

■

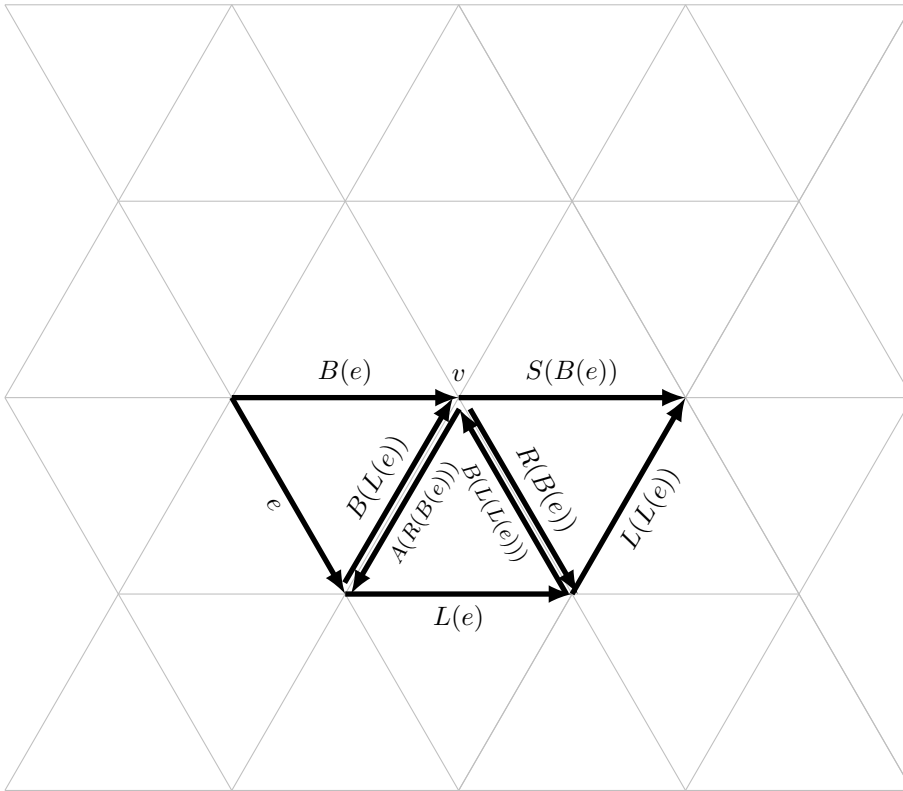


Figure 6.13: A sequence containing only straight steps and two left steps can always be rewritten as a shorter sequence.

Lemma 6.3.23 *In the dual of a nanocone, for each edge e and for each $n \in \mathbb{N}$: the path corresponding to*

$$e, L(e), S(L(e)), S^2(L(e)), \dots, S^n(L(e))$$

and the path corresponding to

$$B(e), S(B(e)), \dots, S^n(B(e)), R(S^n(B(e)))$$

have the same start and end vertex when all except possibly the start and end vertex have degree 6 and all faces are triangles.

Proof: See Figure 6.14. This proof is also analogous to the previous two lemmas. ■

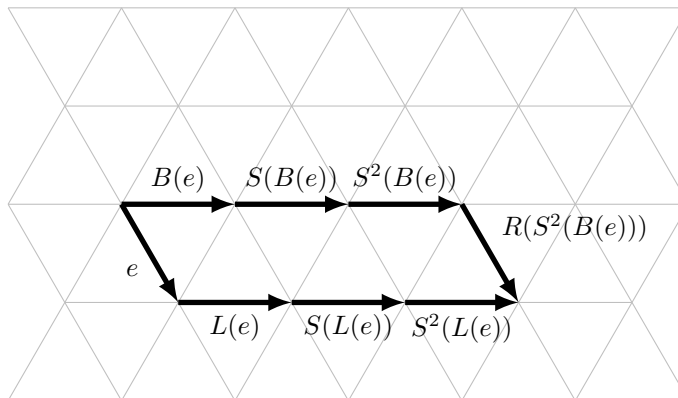


Figure 6.14: Two sequences with the same length.

Theorem 6.3.24 *A near-symmetric cone with two pentagons contains two canonical patches and these two patches are isomorphic.*

Proof: We start by constructing an initial patch based on shortest paths between the pentagons and then construct a symmetric or near-symmetric patch from this initial patch.

Using the previous lemmas one can see that in a nanocone with two pentagons there are two possible situations for shortest paths connecting the pentagon:

- 1 There is a unique shortest path that goes straight at each hexagon (see Figure 6.15).

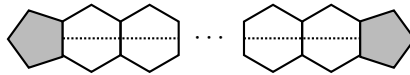


Figure 6.15: A chain of hexagons bounded by a pentagon on each side.

The patch corresponding to this path has a boundary of the form

$$(22(23)^l)^2, l \geq 1,$$

where $l - 1$ is the number of hexagons.

- 2 There is no straight path between the pentagons. In this case there are several shortest paths but due to Lemma 6.3.20, Lemma 6.3.21 and Lemma 6.3.23 there exist exactly two shortest paths with only one turn each. One of them has only one right turn and the other one only one left turn (see Figure 6.16). The initial patch is then formed by taking the faces enclosed by these two shortest paths.

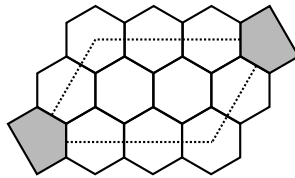


Figure 6.16: A quadrangular patch formed by two pentagons and the enclosing shortest paths with only one turn each.

This patch has a boundary of the form

$$(2(23)^k 2(23)^l)^2, 0 < k \leq l,$$

where k is the number of hexagons in the left or right side and l is the number of hexagons in the top or bottom side.

We can summarise both results in a general boundary form:

$$(2(23)^k 2(23)^l)^2, 0 \leq k \leq l \neq 0.$$

Including hexagons of the cone that lie along the two sides with length l , the corresponding sides in the new patch will have length $l - 1$ and the other two sides will have length $k + 2$. Note that the sets of hexagons for both sides are disjoint, because otherwise some bounded region would be formed that — due to the structure of the boundary and the Euler formula — would have to contain pentagons. We can distinguish three cases for the parameters k and l , which will be handled differently:

(i) $l - k \pmod 3 = 0$

We can add $\frac{l-k}{3}$ times a side of hexagons to the sides with length l . This results in a patch with a boundary of the form $(2(23)^m)^4$ with

$$m = l - \frac{l-k}{3} = k + 2 \left(\frac{l-k}{3} \right) = \frac{2l+k}{3}. \quad (6.67)$$

We only add hexagons to two opposite sides, so in the obtained patch the pentagons are (on symmetric positions) on the boundary, and thus this is a canonical patch corresponding to a symmetric cone.

(ii) $l - k \pmod 3 = 1$

In this case the four sides will never be equal. We can add $\frac{l-k-1}{3}$ times a side of hexagons to the sides with length l . This results in a patch with a boundary of the form $(2(23)^m 2(23)^{m+1})^2$ with

$$m = \frac{k+2l-2}{3}. \quad (6.68)$$

At this moment this patch still contains a rotational symmetry of 180° and the pentagons are on the sides with length m . To get a canonical cone patch we can add one side of hexagons to one side with length $m+1$. This results in a patch with boundary $2(23)^m (2(23)^{m+1})^3$. We can choose the

side in two ways, but this results in isomorphic patches, and as maximal two patches exist, these are all. The two opposite sides with length $m + 1$ will both contain one of the pentagons.

(iii) $l - k \pmod 3 = 2$

In this case the four sides will never be equal. We can add $\frac{l-k+1}{3}$ times a side of hexagons to the sides with length l . This results in a patch with a boundary of the form $(2(23)^m 2(23)^{m+1})^2$ with

$$m = \frac{k + 2l - 1}{3}. \quad (6.69)$$

At this moment this patch still contains a rotational symmetry of 180° and the pentagons are on the sides with length $m + 1$. To get a canonical cone patch we can add one side of hexagons to one side with length $m + 1$. This results in a patch with boundary $2(23)^m (2(23)^{m+1})^3$. We can choose the side in two ways, but this results in isomorphic patches, and as maximal two patches exist, these are all. The side opposite to the side with length m will contain one of the pentagons. The other pentagon does not lie on the boundary.

In Theorem 6.3.17 we showed that a near-symmetric cone with two pentagons contains at most two canonical cone patches. The construction above now proves that these two are isomorphic. ■

There is also an intuitive argument for this theorem. The initial patch in Figure 6.16 has a 2-fold rotational symmetry. A near-symmetric cone patch does not have such a 2-fold rotational symmetry, so the near-symmetric patch will be mapped on to another near-symmetric cone patch in the cone and that patch will thus be isomorphic to the first one. In the case of a symmetric cone patch, the patch is mapped on to itself.

Lemma 6.3.25 *Each pseudo-convex patch with $1 \leq p \leq 5$ pentagons can be extended to a unique nanocone by adding layers of hexagons.*

Proof: Let P be a pseudo-convex patch that is the subgraph of a cone graph G . Both P and G contain $1 \leq p \leq 5$ pentagons. We denote the vertices of degree 2, respectively degree 3, on the boundary of P by b_2 , respectively b_3 . Due to the Euler formula, we have

$$b_2 - b_3 = 6 - p. \quad (6.70)$$

We need to prove that the set of neighbouring hexagons N_H^P of P induces a cycle in the dual graph of G . There are two possible (mutually non-exclusive) situations when N_H^P does not induce a cycle:

- there is a hexagon that has a disconnected intersection with the boundary of P , or
- there are two hexagons that share an edge that has an empty intersection with P .

Assume first that there is a hexagon H that has a disconnected intersection with the boundary of P . The possible situations are shown in Figure 6.17. Only one of the regions A and B (or A , B and C in the fifth case) is unbounded. The other regions are bounded and correspond to a (not necessarily pseudo-convex) patch in G with only hexagons.

Consider the subgraph G_A of G that corresponds to A . We denote the number of vertices with degree 2, respectively degree 3, in the boundary of G_A by b_2^A , respectively b_3^A . If A is a bounded face, then we know from the Euler formula that

$$b_2^A - b_3^A = 6. \quad (6.71)$$

If A is an unbounded face, then its complement is a patch with p pentagons, and so we know from the Euler formula that

$$b_2^A - b_3^A = p - 6. \quad (6.72)$$

We will now examine case by case. Assume first that we have the situation in Figure 6.17.a. Since either A or B is bounded, and the other is unbounded, we

know from 6.71 and 6.72, that

$$(b_2^A - b_3^A) + (b_2^B - b_3^B) = p. \quad (6.73)$$

The vertices of degree 2 in the boundary of G_A and G_B correspond to the vertices of degree 3 in the boundary of P plus the four vertices of the hexagon that lie on the boundary of either G_A or G_B . This gives us

$$b_2^A + b_2^B = b_3 + 4. \quad (6.74)$$

The vertices of degree 3 in the boundary of G_A and G_B correspond to the vertices of degree 2 in the boundary of P minus the four vertices of the hexagon that lie on the boundary of either G_A or G_B . This gives us

$$b_3^A + b_3^B = b_2 - 4. \quad (6.75)$$

If we substitute 6.74 and 6.75 into 6.73, then we get

$$b_3 - b_2 = p - 8.$$

This is in contradiction with 6.70, so the situation in Figure 6.17.a cannot occur.

Assume that we have the situation in Figure 6.17.b. Since again either A or B is bounded, and the other is unbounded, we know from 6.71 and 6.72, that

$$(b_2^A - b_3^A) + (b_2^B - b_3^B) = p. \quad (6.76)$$

The vertices of degree 2 in the boundary of G_A and G_B correspond to the vertices of degree 3 in the boundary of P plus the four vertices of the hexagon that lie on the boundary of either G_A or G_B and on the boundary of P . This gives us

$$b_2^A + b_2^B = b_3 + 4. \quad (6.77)$$

The vertices of degree 3 in the boundary of G_A and G_B correspond to the vertices of degree 2 in the boundary of P minus the four vertices of the hexagon

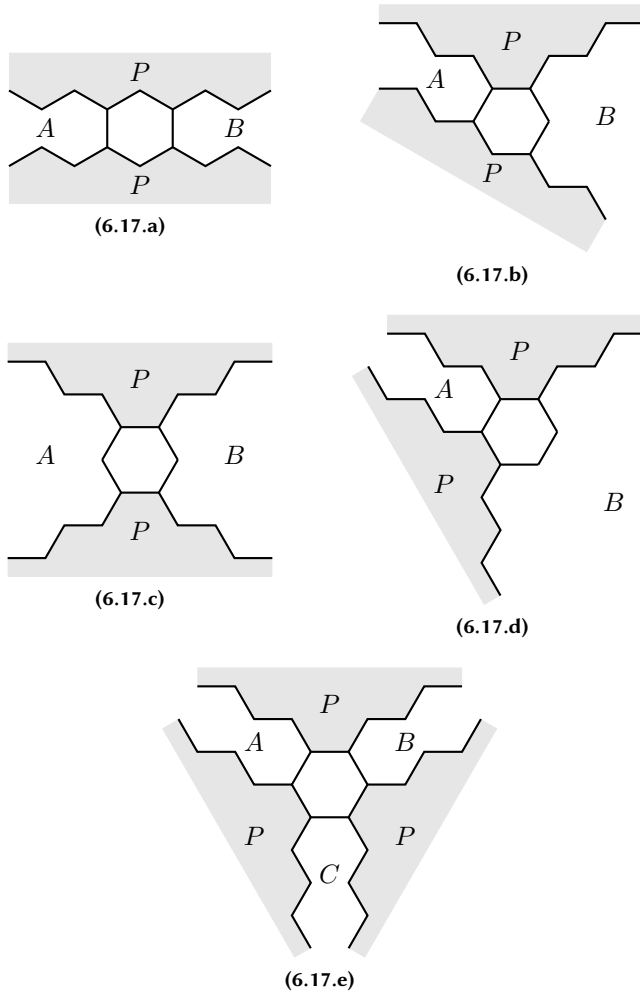


Figure 6.17: Possible situations when a neighbouring hexagon has a disconnected boundary with a pseudo-convex patch P .

that lie on the boundary of either G_A or G_B and on the boundary of P and plus the single vertex of the hexagon that does not lie on the boundary of P . This gives us

$$b_3^A + b_3^B = b_2 - 3. \quad (6.78)$$

If we substitute 6.77 and 6.78 into 6.76, then we get

$$b_3 - b_2 = p - 7.$$

This is in contradiction with 6.70, so the situation in Figure 6.17.b cannot occur. Assume that we have the situation in Figure 6.17.c. Since again either A or B is bounded, and the other is unbounded, we know from 6.71 and 6.72, that

$$(b_2^A - b_3^A) + (b_2^B - b_3^B) = p. \quad (6.79)$$

The vertices of degree 2 in the boundary of G_A and G_B correspond to the vertices of degree 3 in the boundary of P plus the four vertices of the hexagon that lie on the boundary of P . This gives us

$$b_2^A + b_2^B = b_3 + 4. \quad (6.80)$$

The vertices of degree 3 in the boundary of G_A and G_B correspond to the vertices of degree 2 in the boundary of P minus the four vertices of the hexagon that lie on the boundary of P and plus the two vertices of the hexagon that do not lie on the boundary of P . This gives us

$$b_3^A + b_3^B = b_2 - 2. \quad (6.81)$$

If we substitute 6.80 and 6.81 into 6.79, then we get

$$b_3 - b_2 = p - 6.$$

So here we find no contradiction with 6.70.

Since P is a pseudo-convex patch, the boundary of G_A has exactly two edges that are incident to two vertices of degree 2, i.e., the two edges that are incident

to exactly one vertex of the hexagon. The boundary of G_A has at most $(6 - p) - 2$ edges that are incident to two vertices of degree 3, since such an edge corresponds to a break-edge of P that is not incident to the hexagon. All other edges on the boundary of G_A are incident to exactly one vertex of degree 2 and exactly one vertex of degree 3. This gives us

$$b_3^A - b_2^A \leq (6 - p) - 2 - 2 = 2 - p.$$

The same argumentation can be used for G_B . The obtained inequality, however, is in contradiction with 6.72, so the situation in Figure 6.17.c cannot occur. Also for the situations in Figure 6.17.d and Figure 6.17.e, the initial approach to calculate the sum of the number of degree two and degree three vertices in the boundary of the different regions will not work. Therefore we immediately use the second approach for these cases.

Assume that we have the situation in Figure 6.17.d. Since P is a pseudo-convex patch, the boundary of G_A has exactly three edges that are incident to two vertices of degree 2, i.e., the three edges that are incident to a vertex of the hexagon. The boundary of G_A has at most $(6 - p) - 2$ edges that are incident to two vertices of degree 3, since such an edge corresponds to a break-edge of P that is not incident to the hexagon. All other edges on the boundary of G_A are incident to exactly one vertex of degree 2 and exactly one vertex of degree 3. This gives us

$$b_3^A - b_2^A \leq (6 - p) - 2 - 3 = 1 - p. \quad (6.82)$$

Since this is in contradiction to 6.72, we find that A is bounded.

Since P is a pseudo-convex patch, the boundary of G_B has exactly two edges that are incident to two vertices of degree 2, i.e., the two edges that are incident to exactly one vertex of the hexagon. The boundary of G_B has at most $(6 - p) - 2 + 1$ edges that are incident to two vertices of degree 3, since such an edge corresponds to a break-edge of P that is not incident to the hexagon or to the edge of the hexagon that has an empty intersection with P . All other edges on the boundary of G_A are incident to exactly one vertex of degree 2 and

exactly one vertex of degree 3. This gives us

$$b_3^B - b_2^B \leq (6 - p) - 2 + 1 - 2 = 3 - p.$$

Since this is in contradiction to 6.72, we find that B is bounded. So we find that the situation in Figure 6.17.d cannot occur, because either A or B is unbounded.

Assume that we have the situation in Figure 6.17.e. The same argumentation as for the region A , can be used for all regions in this case. This means that all regions satisfy 6.82, and thus all regions are bounded. So we find that the situation in Figure 6.17.e cannot occur, because either A , B or C is unbounded.

This means that it is not possible that a hexagon of N_H^P has a disconnected intersection with the boundary of P . Next we turn our attention to the possible situations for two hexagons of N_H^P to share an edge that has an empty intersection with P . The possibilities are shown in Figure 6.18.

Assume first that we have the situation in Figure 6.18.a or in Figure 6.18.b. Since either A or B is bounded, and the other is unbounded, we know from 6.71 and 6.72, that

$$(b_2^A - b_3^A) + (b_2^B - b_3^B) = p. \quad (6.83)$$

The vertices of degree 2 in the boundary of G_A and G_B correspond to the vertices of degree 3 in the boundary of P plus the four vertices of the hexagons that lie on the boundary of either G_A or G_B and on the boundary of P plus the two vertices that lie on the edge that is shared by the two hexagons. This gives us

$$b_2^A + b_2^B = b_3 + 4 + 2 = b_3 + 6. \quad (6.84)$$

The vertices of degree 3 in the boundary of G_A and G_B correspond to the vertices of degree 2 in the boundary of P minus the four vertices of the hexagon that lie on the boundary of either G_A or G_B and on the boundary of P plus the two vertices of the hexagons that do not lie on the boundary of P or on the edge that is shared by the two hexagons. This gives us

$$b_3^A + b_3^B = b_2 - 4 + 2 = b_2 - 2. \quad (6.85)$$

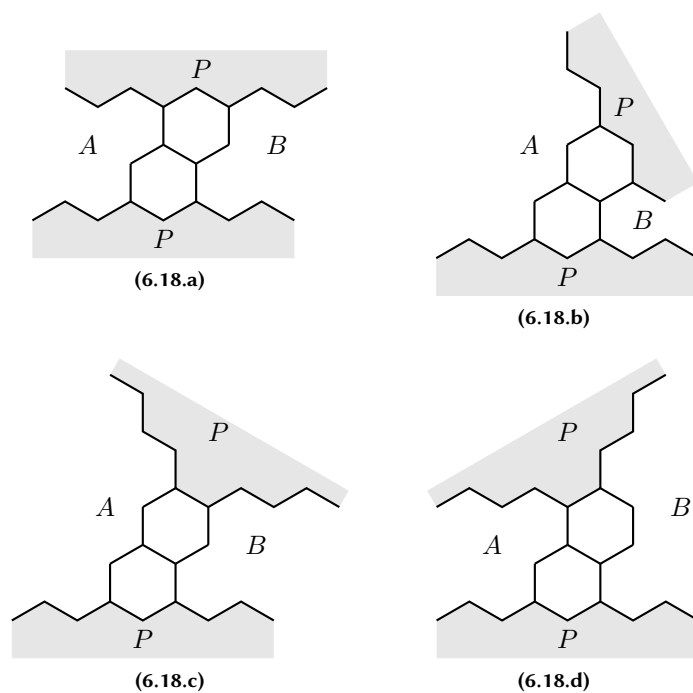


Figure 6.18: Possible situations when two neighbouring hexagons share an edge that has an empty intersection with the pseudo-convex patch P .

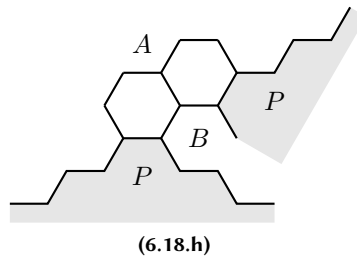
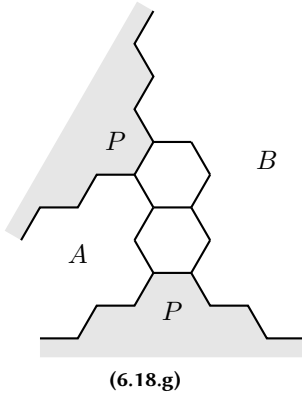
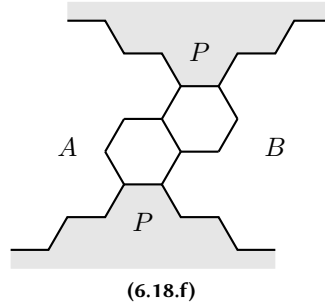
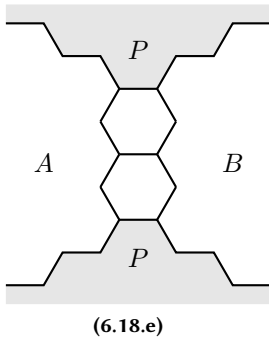


Figure 6.18: Possible situations when two neighbouring hexagons share an edge that has an empty intersection with the pseudo-convex patch P . (Continued)

If we substitute 6.84 and 6.85 into 6.83, then we get

$$b_3 - b_2 = p - 8.$$

This is in contradiction with 6.70, so the situations in Figure 6.18.a and in Figure 6.18.b cannot occur.

Assume that we have the situation in Figure 6.18.c or in Figure 6.18.d. Since either A or B is bounded, and the other is unbounded, we know from 6.71 and 6.72, that

$$(b_2^A - b_3^A) + (b_2^B - b_3^B) = p. \quad (6.86)$$

The vertices of degree 2 in the boundary of G_A and G_B correspond to the vertices of degree 3 in the boundary of P plus the four vertices of the hexagons that lie on the boundary of either G_A or G_B and on the boundary of P plus the two vertices that lie on the edge that is shared by the two hexagons. This gives us

$$b_2^A + b_2^B = b_3 + 4 + 2 = b_3 + 6. \quad (6.87)$$

The vertices of degree 3 in the boundary of G_A and G_B correspond to the vertices of degree 2 in the boundary of P minus the four vertices of the hexagon that lie on the boundary of either G_A or G_B and on the boundary of P plus the three vertices of the hexagons that do not lie on the boundary of P or on the edge that is shared by the two hexagons. This gives us

$$b_3^A + b_3^B = b_2 - 4 + 3 = b_2 - 1. \quad (6.88)$$

If we substitute 6.87 and 6.88 into 6.86, then we get

$$b_3 - b_2 = p - 7.$$

This is in contradiction with 6.70, so the situations in Figure 6.18.c and in Figure 6.18.d cannot occur.

For the remaining four cases, it is again so that the initial approach to calculate the sum of the number of degree two and degree three vertices in the boundary

of the different regions will not work. Therefore, for these cases, we will again use the second approach and find an upper bound on the difference of the number of degree three and degree two vertices in the boundary of the different regions.

Assume that we have the situation in Figure 6.18.e. Since P is a pseudo-convex patch, the boundary of G_A has exactly two edges that are incident to two vertices of degree 2, i.e., the two edges that are incident to exactly one vertex of the hexagons. The boundary of G_A has at most $(6 - p) - 2$ edges that are incident to two vertices of degree 3, since such an edge corresponds to a break-edge of P that is not incident to one of the hexagon. All other edges on the boundary of G_A are incident to exactly one vertex of degree 2 and exactly one vertex of degree 3. This gives us

$$b_3^A - b_2^A \leq (6 - p) - 2 - 2 = 2 - p.$$

The same argumentation can be used for G_B . The obtained inequality, however, is in contradiction with 6.72, so the situation in Figure 6.18.e cannot occur.

Assume that we have the situation in Figure 6.18.f. Since P is a pseudo-convex patch, the boundary of G_A has exactly three edges that are incident to two vertices of degree 2, i.e., the two edges that are incident to exactly one vertex of the hexagons and the edge that is incident to a vertex of P and to a vertex of the edge that is shared by the two hexagons. The boundary of G_A has at most $(6 - p) - 2 + 1$ edges that are incident to two vertices of degree 3, since such an edge corresponds to a break-edge of P that is not incident to one of the hexagon or to the edge on the hexagons that is not shared by the two hexagons and that has an empty intersection with P . All other edges on the boundary of G_A are incident to exactly one vertex of degree 2 and exactly one vertex of degree 3. This gives us

$$b_3^A - b_2^A \leq (6 - p) - 2 + 1 - 3 = 2 - p.$$

The same argumentation can be used for G_B . The obtained inequality, however, is in contradiction with 6.72, so the situation in Figure 6.18.f cannot occur.

Assume that we have the situation in Figure 6.18.g. Since P is a pseudo-convex patch, the boundary of G_A has exactly three edges that are incident to two vertices of degree 2, i.e., the two edges that are incident to exactly one vertex of the hexagons and the edge that is incident to a vertex of P and to a vertex of the edge that is shared by the two hexagons. The boundary of G_A has at most $(6 - p) - 2$ edges that are incident to two vertices of degree 3, since such an edge corresponds to a break-edge of P that is not incident to one of the hexagon. All other edges on the boundary of G_A are incident to exactly one vertex of degree 2 and exactly one vertex of degree 3. This gives us

$$b_3^A - b_2^A \leq (6 - p) - 2 - 3 = 1 - p.$$

Since this is in contradiction to 6.72, we find that A is bounded.

Since P is a pseudo-convex patch, the boundary of G_B has exactly two edges that are incident to two vertices of degree 2, i.e., the two edges that are incident to exactly one vertex of the hexagons. The boundary of G_B has at most $(6 - p) - 2 + 1$ edges that are incident to two vertices of degree 3, since such an edge corresponds to a break-edge of P that is not incident to one of the hexagon or to the edge on the hexagons that has an empty intersection with the patch P and an empty intersection with the edge that is shared by the two hexagons. All other edges on the boundary of G_B are incident to exactly one vertex of degree 2 and exactly one vertex of degree 3. This gives us

$$b_3^A - b_2^A \leq (6 - p) - 2 + 1 - 2 = 3 - p.$$

Since this is in contradiction to 6.72, we find that B is bounded. So we find that the situation in Figure 6.18.g cannot occur, because either A or B is unbounded.

Finally assume that we have the situation in Figure 6.18.h. Since P is a pseudo-convex patch, the boundary of G_A has exactly two edges that are incident to two vertices of degree 2, i.e., the two edges that are incident to exactly one vertex of the hexagons. The boundary of G_A has at most $(6 - p) - 2 + 2$ edges that are incident to two vertices of degree 3, since such an edge corresponds

to a break-edge of P that is not incident to one of the hexagon or to one of the edge on the hexagons that has an empty intersection with the patch P and an empty intersection with the edge that is shared by the two hexagons. All other edges on the boundary of G_A are incident to exactly one vertex of degree 2 and exactly one vertex of degree 3. This gives us

$$b_3^A - b_2^A \leq (6 - p) - 2 + 2 - 2 = 4 - p.$$

Since this is in contradiction to 6.72, we find that A is bounded.

Since P is a pseudo-convex patch, the boundary of G_B has exactly four edges that are incident to two vertices of degree 2, i.e., the two edges that are incident to exactly one vertex of the hexagons the edges that are incident to a vertex of P and to a vertex of the edge that is shared by the two hexagons. The boundary of G_B has at most $(6 - p) - 2$ edges that are incident to two vertices of degree 3, since such an edge corresponds to a break-edge of P that is not incident to one of the hexagon. All other edges on the boundary of G_B are incident to exactly one vertex of degree 2 and exactly one vertex of degree 3. This gives us

$$b_3^A - b_2^A \leq (6 - p) - 2 - 4 = -p.$$

Since this is in contradiction to 6.72, we find that B is bounded. So we find that the situation in Figure 6.18.h cannot occur, because either A or B is unbounded.

This means that it is not possible that two hexagons of N_H^P share an edge that has an empty intersection with P . ■

Corollary 6.3.26 *There is a 1-1 correspondence between the set of canonical patches and the set of nanocones.*

Proof: Due to the previous lemma each (canonical) patch can be extended to a unique nanocone by adding layers of hexagons. Due to Theorem 6.3.14 each nanocone contains a canonical patch. Owing to Theorem 6.3.17 and Theorem

6.3.24, each nanocone has a unique canonical patch that contains all the pentagons, as subgraph. ■

The proof of Theorem 6.3.24 gives us even more information about the structure of the canonical patches with two pentagons.

In Chapter 5 it is discussed how a marked cone patch — which is a pseudo-convex patch — can be reconstructed by its boundary and its outer spiral code.

Corollary 6.3.27 *If we distinguish between mirror images, there are exactly m canonical patches with boundary $(2(23)^m)^4$ and they have outer spiral code $(i, 2m + i)$ with $i = 0, \dots, m - 1$. When we do not distinguish between mirror images, there are exactly $\lceil \frac{m+1}{2} \rceil$ canonical patches with boundary $(2(23)^m)^4$ and they have outer spiral code $(i, 2m + i)$ with $i = 0, \dots, \lceil \frac{m+1}{2} \rceil - 1$.*

Proof: This result follows from the proof of Theorem 6.3.24 together with the fact that there are m possible positions for the pentagon on a side. In this proof it was shown that if the parameters l and k satisfy the condition $l - k \bmod 3 = 0$ then the initial patch leads to a symmetric patch. The value of i in Figure 6.19 is equal to the number of sides of hexagons that are added to one side of the initial patch. This means that $i = \frac{l-k}{3}$. When we substitute this in formula 6.67, we find:

$$m = \frac{2l + k}{3} = l - i.$$

This equation has a solution for $l \in \mathbb{N}$ for each value of $i \in \{0, \dots, m - 1\}$ and so an initial patch for each of the m positions of the pentagon can be constructed and thus a symmetric patch for each position exists. The outer spiral code can be deduced from Figure 6.19.

When we take all symmetries into account we can use the fact that the patches with $i < \lceil \frac{m+1}{2} \rceil$ are mirror images with as mirror axis the middle row of hexagons of the patches with $i > \lceil \frac{m+1}{2} \rceil$. Therefore each patch was considered twice in this case except the patch with $i = 0$ and, in case $m \in 2\mathbb{N}$, the patch with $i = \frac{m}{2}$. ■

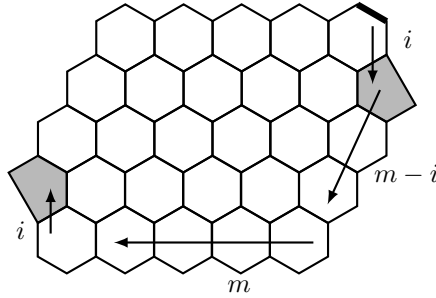


Figure 6.19: A symmetric cone patch with two pentagons. In this example the values for the parameters are $k = 2, l = 5, m = 4$ and $i = 1$.

Corollary 6.3.28 *If we distinguish between mirror images, there are exactly $2m + 1$ canonical patches with boundary $2(23)^m(2(23)^{m+1})^3$ and they have outer spiral code $(i, 2m + 1 + i)$ with $i = m + 1, \dots, 3m + 1$. When we do not distinguish between mirror images, there are exactly $m + 1$ canonical patches with boundary $2(23)^m(2(23)^{m+1})^3$ and they have outer spiral code $(i, 2m + 1 + i)$ with $i = m + 1, \dots, m + 2 + \lfloor \frac{m+1}{2} \rfloor, 2m + 2, \dots, 3m + 1 - \lfloor \frac{m}{2} \rfloor$.*

Proof: This result follows from the proof of Theorem 6.3.24 together with the fact that there are $m + 1$ possible positions for the pentagon on the first side with length m and m positions for the pentagon on the second side with length m .

Assume first that there is a pentagon on the second side with length m . The only case where this occurs in the proof of Theorem 6.3.24 is when $l - k \pmod 3 = 2$. The value of i in Figure 6.20.a is equal to $\frac{l-k+1}{3} + (2m + 1)$, i.e., the number of hexagon sides added to one side plus $2m + 1$. When we substitute this equality in formula 6.69, we find:

$$m = \frac{k + 2l - 1}{3} = l + (2m + 1) - i,$$

which means that

$$m = i - 1 - l.$$

This last equation has a solution for $l \in \mathbb{N}$ for each value of $i \in \{2m + 2, \dots, 3m + 1\}$ and so an initial patch for each of the m positions of the first pentagon in Figure 6.20.a can be constructed and thus a near-symmetric patch for each position exists. The outer spiral code can be deduced from Figure 6.20.a. This code is $(i, 2m + 1 + i)$ with $i = 2m + 2, \dots, 3m + 1$.

The other case in the proof of Theorem 6.3.24 is when there is a pentagon on the first side with length m , and this patch corresponds to an initial patch for which $l - k \bmod 3 = 1$. The value of i in Figure 6.20.b is equal to $\frac{l-k-1}{3} + m + 1$, i.e., the number of hexagon sides added to one side plus $m + 1$. When we substitute this equality in formula 6.68, we find:

$$m = \frac{k + 2l - 2}{3} = l - i + m,$$

which means that

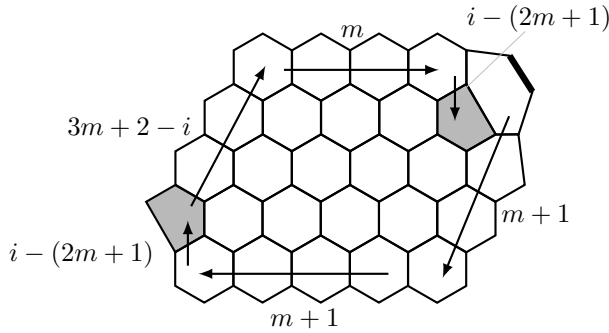
$$l = i.$$

This last equation has a solution for $l \in \mathbb{N}$ for each value of $i \in \{m + 1, \dots, 2m + 1\}$ and so an initial patch for each of the $m + 1$ positions of the first pentagon in Figure 6.20.b can be constructed and thus a near-symmetric patch for each position exists. The outer spiral code can be deduced from Figure 6.20.b. This code is $(i, 2m + 1 + i)$ with $i = m + 1, \dots, 2m + 1$.

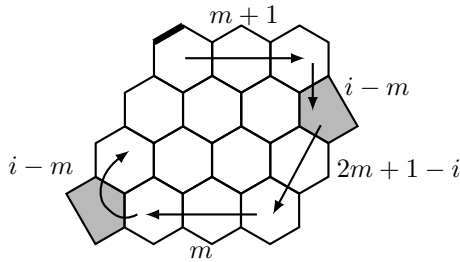
These two cases can be summarised in the given code.

When we take all symmetries into account we can use the fact that, for the case of Figure 6.20.b, the patches with $i < 3m + 1 - \lfloor \frac{m}{2} \rfloor$ are mirror images with as mirror axis the middle row of hexagons of the patches with $i > 3m + 1 - \lfloor \frac{m}{2} \rfloor$, and, for the case of Figure 6.20.a, the patches with $i < m + 1 - \lfloor \frac{m+1}{2} \rfloor$ are mirror images with as mirror axis the middle row of hexagons of the patches with $i > m + 1 - \lfloor \frac{m+1}{2} \rfloor$. This gives us $\lceil \frac{m}{2} \rceil + \lceil \frac{m+1}{2} \rceil = m + 1$ distinct patches. ■

Remark 6.3.29 *To generate all canonical cone patches with two pentagons, a given side length and a near-symmetric boundary we can just generate the spiral code for*



(6.20.a) In this example the values for the parameters are $k = 2, l = 4, m = 3$ and $i = 8$.



(6.20.b) In this example the values for the parameters are $k = 2, l = 3, m = 2$ and $i = 3$.

Figure 6.20: Two near-symmetric patches.

near-symmetric patches with $i = m + 1 + \lfloor \frac{m+1}{2} \rfloor, \dots, 3m + 1 - \lfloor \frac{m}{2} \rfloor$. This will give the same patches, but for the patches of the type in Figure 6.20.b the mirror image will be generated. This is the method that is used in cone.

Theorem 6.3.30 *There is up to isomorphism exactly one cone with one pentagon and the canonical cone patch for this cone is the pentagon.*

The reason that we defined a near-symmetric boundary only for $1 < p < 5$ and not for $1 \leq p < 5$ is because this would not allow for the 1-1 correspondence between canonical cone patches and cones. If we allowed the near-symmetric variant for $p = 1$, then the unique nanocone with one pentagon would contain two non-

isomorphic canonical cone patches (see Figure 6.21).

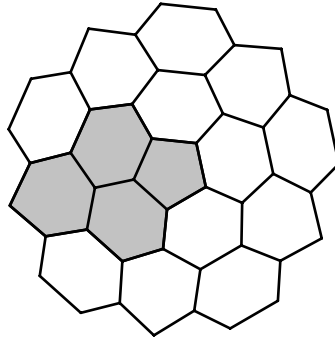


Figure 6.21: *The unique nanocone with one pentagon with a near-symmetric-like patch in grey. This patch has the canonical boundary sequence $2(23)^0(2(23)^1)^4$.*

6.4 Algorithm

Since these nanocone patches are also pseudo-convex patches we will use the algorithm developed in Chapter 5. For this special case the pseudo-convex boundaries have some specific properties.

A symmetric patch has $2(6 - p)$ possible ways of defining a canonical marked edge: each break-edge once for the patch and once for the mirror image of the patch. A near-symmetric patch has 2 possible ways of defining a canonical marked edge: the two break-edges next to the shortest side: one for the patch and the other for the mirror image of the patch.

A canonical cone patch also has the restriction that there must be a pentagon with a non-empty intersection with the boundary. This means that a pentagon must be placed in the first shell. In the case of a symmetric patch, due to the symmetry of the boundary, a pentagon must be placed in the first half of the first side after the break-edge that is used as mark. In most cases this will break the symmetry very early in the generation process and in those cases the generation can be com-

pleted without performing any isomorphism checks as was explained in the previous chapter.

In chemistry, *IPR-structures*, i.e., structures in which no two pentagons share an edge, are especially interesting because they tend to be energetically better than structures with fused pentagons. The generation in *cone* can also be restricted to this subclass.

6.5 Testing

This algorithm was implemented in *C* as the program *cone*. It was tested against the completely independent program *vul_in* described in [54], which was itself already extensively tested.

For our test we compared the output structures with boundary $(2(23)^s)^{6-p}$ and $2(23)^s(2(23)^{s+1})^{6-p-1}$ for $2 \leq p \leq 5$ and s up to 35 and compared the numbers of structures for s up to 50. This was done once with and once without the restriction to IPR-structures. We had agreement in all cases.

6.6 Results

Table 6.2: *The number of canonical cone patches for given minimal side length, given number of pentagons and given type of boundary.*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
0	1	0	1	0	0	0	0	0
1	0	1	2	1	2	0	0	0
2	0	2	3	3	8	1	2	0
3	0	2	4	5	18	4	9	0
4	0	3	5	12	37	16	32	0

Continued on next page

Table 6.2: *The number of canonical cone patches for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
5	0	3	6	18	63	37	89	1
6	0	4	7	31	104	89	204	5
7	0	4	8	44	153	175	420	14
8	0	5	9	67	224	336	786	41
9	0	5	10	88	305	579	1376	93
10	0	6	11	124	413	975	2272	212
11	0	6	12	157	534	1533	3589	413
12	0	7	13	207	688	2363	5450	797
13	0	7	14	255	856	3477	8021	1399
14	0	8	15	323	1064	5030	11474	2424
15	0	8	16	387	1288	7040	16032	3941
16	0	9	17	476	1557	9716	21928	6317
17	0	9	18	560	1845	13078	29452	9686
18	0	10	19	671	2184	17400	38902	14654
19	0	10	20	778	2543	22707	50645	21451
20	0	11	21	915	2960	29342	65056	31025
21	0	11	22	1046	3399	37336	82585	43754
22	0	12	23	1212	3901	47112	103692	61042
23	0	12	24	1371	4428	58705	128920	83484
24	0	13	25	1567	5024	72627	158822	113084
25	0	13	26	1757	5646	88918	194044	150732
26	0	14	27	1987	6344	108185	235240	199211
27	0	14	28	2209	7070	130476	283165	259804
28	0	15	29	2476	7877	156502	338586	336263

Continued on next page

Table 6.2: *The number of canonical cone patches for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
29	0	15	30	2734	8715	186315	402377	430401
30	0	16	31	3039	9640	220741	475422	547166
31	0	16	32	3336	10597	259839	558724	689034
32	0	17	33	3683	11648	304554	653296	862426
33	0	17	34	4020	12733	354953	760276	1070653
34	0	18	35	4412	13917	412111	880814	1321891
35	0	18	36	4793	15138	476101	1016193	1620501
36	0	19	37	5231	16464	548137	1167708	1976744
37	0	19	38	5659	17828	628299	1336797	2396247
38	0	20	39	6147	19304	717946	1524908	2891749
39	0	20	40	6623	20820	817168	1733640	3470397
40	0	21	41	7164	22453	927478	1964606	4147802
41	0	21	42	7692	24129	1048972	2219576	4932954
42	0	22	43	8287	25928	1183328	2500332	5844785
43	0	22	44	8870	27771	1330651	2808825	6894435
44	0	23	45	9523	29744	1492788	3147018	8104688
45	0	23	46	10162	31763	1669854	3517049	9489198
46	0	24	47	10876	33917	1863876	3921070	11075123
47	0	24	48	11575	36120	2074977	4361416	12879072
48	0	25	49	12351	38464	2305373	4840436	14933165
49	0	25	50	13113	40858	2555196	5360672	17257392
50	0	26	51	13955	43400	2826857	5924678	19889522
51	0	26	52	14781	45994	3120500	6535209	22853408
52	0	27	53	15692	48741	3438740	7195036	26193138

Continued on next page

Table 6.2: *The number of canonical cone patches for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
53	0	27	54	16586	51543	3781729	7907137	29937017
54	0	28	55	17567	54504	4152297	8674504	34136195
55	0	28	56	18532	57521	4550607	9500348	38823982
56	0	29	57	19587	60704	4979708	10387894	44059455
57	0	29	58	20624	63945	5439775	11340592	49881557
58	0	30	59	21756	67357	5934087	12361908	56358126
59	0	30	60	22869	70830	6462829	13455541	63534485
60	0	31	61	24079	74480	7029519	14625206	71488151
61	0	31	62	25271	78192	7634353	15874861	80271535
62	0	32	63	26563	82088	8281094	17208478	89972894
63	0	32	64	27835	86048	8969952	18630280	100652509
64	0	33	65	29212	90197	9704944	20144508	112410395
65	0	33	66	30568	94413	10486290	21755660	125315638
66	0	34	67	32031	98824	11318272	23468250	139481120
67	0	34	68	33474	103303	12201123	25287061	154985603
68	0	35	69	35027	107984	13139394	27216892	171956115
69	0	35	70	36558	112735	14133332	29262817	190482058
70	0	36	71	38204	117693	15187768	31429928	210705824
71	0	36	72	39827	122724	16302961	33723600	232728588
72	0	37	73	41567	127968	17484031	36149226	256709426
73	0	37	74	43285	133286	18731250	38712492	282762348
74	0	38	75	45123	138824	20050033	41419100	311064629
75	0	38	76	46937	144438	21440668	44275053	341744268

Table 6.3: *The number of canonical IPR cone patches for given minimal side length, given number of pentagons and given type of boundary.*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
0	1	0	0	0	0	0	0	0
1	0	1	2	0	0	0	0	0
2	0	2	3	2	5	0	0	0
3	0	2	4	4	14	0	0	0
4	0	3	5	10	32	4	6	0
5	0	3	6	16	57	15	36	0
6	0	4	7	29	97	48	111	0
7	0	4	8	41	145	111	272	0
8	0	5	9	64	215	238	565	0
9	0	5	10	85	295	441	1063	1
10	0	6	11	120	402	782	1844	13
11	0	6	12	153	522	1279	3022	46
12	0	7	13	203	675	2029	4717	153
13	0	7	14	250	842	3055	7093	358
14	0	8	15	318	1049	4499	10319	797
15	0	8	16	382	1272	6390	14617	1527
16	0	9	17	470	1540	8922	20216	2817
17	0	9	18	554	1827	12130	27405	4787
18	0	10	19	665	2165	16269	36479	7916
19	0	10	20	771	2523	21381	47803	12411
20	0	11	21	908	2939	27790	61749	19067
21	0	11	22	1039	3377	35544	78766	28230
22	0	12	23	1204	3878	45045	99310	41108
23	0	12	24	1363	4404	56349	123923	58269

Continued on next page

Table 6.3: *The number of canonical IPR cone patches for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
24	0	13	25	1559	4999	69943	153155	81478
25	0	13	26	1748	5620	85890	187650	111580
26	0	14	27	1978	6317	104772	228059	151090
27	0	14	28	2200	7042	126660	275136	201233
28	0	15	29	2466	7848	152238	329644	265436
29	0	15	30	2724	8685	181585	392456	345462
30	0	16	31	3029	9609	215496	464453	445875
31	0	16	32	3325	10565	254059	546636	569080
32	0	17	33	3672	11615	298188	640015	721084
33	0	17	34	4009	12699	347979	745727	905102
34	0	18	35	4400	13882	404474	864918	1128825
35	0	18	36	4781	15102	467779	998870	1396537
36	0	19	37	5219	16427	539071	1148875	1717937
37	0	19	38	5646	17790	618465	1316369	2098545
38	0	20	39	6134	19265	707283	1502797	2550509
39	0	20	40	6610	20780	805650	1709757	3080836
40	0	21	41	7150	22412	915040	1938858	3704461
41	0	21	42	7678	24087	1035588	2191869	4430260
42	0	22	43	8273	25885	1168929	2470569	5276392
43	0	22	44	8855	27727	1315209	2776907	6253850
44	0	23	45	9508	29699	1476232	3112843	7384619
45	0	23	46	10147	31717	1652154	3480514	8682157
46	0	24	47	10860	33870	1844957	3882068	10172724
47	0	24	48	11559	36072	2054809	4319839	11872774

Continued on next page

Table 6.3: *The number of canonical IPR cone patches for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
48	0	25	49	12335	38415	2283877	4796173	13813410
49	0	25	50	13096	40808	2532340	5313610	16014425
50	0	26	51	13938	43349	2802560	5874701	18512550
51	0	26	52	14764	45942	3094728	6482200	21331380
52	0	27	53	15674	48688	3411408	7138874	24513852
53	0	27	54	16568	51489	3752803	7847700	28088065
54	0	28	55	17549	54449	4121688	8611667	32103875
55	0	28	56	18513	57465	4518279	9433984	36594341
56	0	29	57	19568	60647	4945570	10317873	41617232
57	0	29	58	20605	63887	5403789	11266783	47211159
58	0	30	59	21736	67298	5896158	12284176	53442525
59	0	30	60	22849	70770	6422919	13373750	60356392
60	0	31	61	24059	74419	6987529	14539217	68028673
61	0	31	62	25250	78130	7590243	15784533	76511464
62	0	32	63	26542	82025	8234763	17113667	85891413
63	0	32	64	27814	85984	8921358	18530841	96228389
64	0	33	65	29190	90132	9653982	20040292	107620657
65	0	33	66	30546	94347	10432918	21646517	120136977
66	0	34	67	32009	98757	11262381	23354027	133888284
67	0	34	68	33451	103235	12142669	25167603	148952957
68	0	35	69	35004	107915	13078266	27092041	165456083
69	0	35	70	36535	112665	14069484	29132414	183486560
70	0	36	71	38180	117622	15121085	31293810	203184684
71	0	36	72	39803	122652	16233397	33581603	224651235

Continued on next page

Table 6.3: *The number of canonical IPR cone patches for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
72	0	37	73	41543	127895	17411467	36001183	248042966
73	0	37	74	43260	133212	18655638	38558234	273473426
74	0	38	75	45098	138749	19971252	41258455	301117586
75	0	38	76	46912	144362	21358668	44107848	331102843

Table 6.4: *The number of canonical non-IPR cone patches for given minimal side length, given number of pentagons and given type of boundary.*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
0	0	0	1	0	0	0	0	0
1	0	0	0	1	2	0	0	0
2	0	0	0	1	3	1	2	0
3	0	0	0	1	4	4	9	0
4	0	0	0	2	5	12	26	0
5	0	0	0	2	6	22	53	1
6	0	0	0	2	7	41	93	5
7	0	0	0	3	8	64	148	14
8	0	0	0	3	9	98	221	41
9	0	0	0	3	10	138	313	92
10	0	0	0	4	11	193	428	199
11	0	0	0	4	12	254	567	367
12	0	0	0	4	13	334	733	644

Continued on next page

Table 6.4: *The number of canonical non-IPR cone patches for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
13	0	0	0	5	14	422	928	1041
14	0	0	0	5	15	531	1155	1627
15	0	0	0	5	16	650	1415	2414
16	0	0	0	6	17	794	1712	3500
17	0	0	0	6	18	948	2047	4899
18	0	0	0	6	19	1131	2423	6738
19	0	0	0	7	20	1326	2842	9040
20	0	0	0	7	21	1552	3307	11958
21	0	0	0	7	22	1792	3819	15524
22	0	0	0	8	23	2067	4382	19934
23	0	0	0	8	24	2356	4997	25215
24	0	0	0	8	25	2684	5667	31606
25	0	0	0	9	26	3028	6394	39152
26	0	0	0	9	27	3413	7181	48121
27	0	0	0	9	28	3816	8029	58571
28	0	0	0	10	29	4264	8942	70827
29	0	0	0	10	30	4730	9921	84939
30	0	0	0	10	31	5245	10969	101291
31	0	0	0	11	32	5780	12088	119954
32	0	0	0	11	33	6366	13281	141342
33	0	0	0	11	34	6974	14549	165551
34	0	0	0	12	35	7637	15896	193066
35	0	0	0	12	36	8322	17323	223964
36	0	0	0	12	37	9066	18833	258807

Continued on next page

Table 6.4: *The number of canonical non-IPR cone patches for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
37	0	0	0	13	38	9834	20428	297702
38	0	0	0	13	39	10663	22111	341240
39	0	0	0	13	40	11518	23883	389561
40	0	0	0	14	41	12438	25748	443341
41	0	0	0	14	42	13384	27707	502694
42	0	0	0	14	43	14399	29763	568393
43	0	0	0	15	44	15442	31918	640585
44	0	0	0	15	45	16556	34175	720069
45	0	0	0	15	46	17700	36535	807041
46	0	0	0	16	47	18919	39002	902399
47	0	0	0	16	48	20168	41577	1006298
48	0	0	0	16	49	21496	44263	1119755
49	0	0	0	17	50	22856	47062	1242967
50	0	0	0	17	51	24297	49977	1376972
51	0	0	0	17	52	25772	53009	1522028
52	0	0	0	18	53	27332	56162	1679286
53	0	0	0	18	54	28926	59437	1848952
54	0	0	0	18	55	30609	62837	2032320
55	0	0	0	19	56	32328	66364	2229641
56	0	0	0	19	57	34138	70021	2442223
57	0	0	0	19	58	35986	73809	2670398
58	0	0	0	20	59	37929	77732	2915601
59	0	0	0	20	60	39910	81791	3178093
60	0	0	0	20	61	41990	85989	3459478

Continued on next page

Table 6.4: *The number of canonical non-IPR cone patches for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
61	0	0	0	21	62	44110	90328	3760071
62	0	0	0	21	63	46331	94811	4081481
63	0	0	0	21	64	48594	99439	4424120
64	0	0	0	22	65	50962	104216	4789738
65	0	0	0	22	66	53372	109143	5178661
66	0	0	0	22	67	55891	114223	5592836
67	0	0	0	23	68	58454	119458	6032646
68	0	0	0	23	69	61128	124851	6500032
69	0	0	0	23	70	63848	130403	6995498
70	0	0	0	24	71	66683	136118	7521140
71	0	0	0	24	72	69564	141997	8077353
72	0	0	0	24	73	72564	148043	8666460
73	0	0	0	25	74	75612	154258	9288922
74	0	0	0	25	75	78781	160645	9947043
75	0	0	0	25	76	82000	167205	10641425

Table 6.5: *The maximum number of hexagons in a cone patch for given minimal side length, given number of pentagons and given type of boundary.*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
0	0	N/A	1	N/A	N/A	N/A	N/A	N/A
1		2	7	0	3	N/A	N/A	N/A

Continued on next page

Table 6.5: *The maximum number of hexagons in a cone patch for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
2		8	15	4	9	0	2	N/A
3		16	26	11	17	4	7	N/A
4		27	39	19	27	10	13	N/A
5		40	55	30	39	16	21	1
6		56	73	42	53	24	29	5
7		74	94	57	69	33	39	8
8		95	117	73	87	44	50	13
9		118	143	92	107	55	63	17
10		144	171	112	129	68	76	23
11		172	202	135	153	82	91	28
12		203	235	159	179	98	107	35
13		236	271	186	207	114	125	41
14		272	309	214	237	132	143	49
15		310	350	245	269	151	163	56
16		351	393	277	303	172	184	65
17		394	439	312	339	193	207	73
18		440	487	348	377	216	230	83
19		488	538	387	417	240	255	92
20		539	591	427	459	266	281	103
21		592	647	470	503	292	309	113
22		648	705	514	549	320	337	125
23		706	766	561	597	349	367	136
24		767	829	609	647	380	398	149
25		830	895	660	699	411	431	161

Continued on next page

Table 6.5: *The maximum number of hexagons in a cone patch for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
26		896	963	712	753	444	464	175
27		964	1034	767	809	478	499	188
28		1035	1107	823	867	514	535	203
29		1108	1183	882	927	550	573	217
30		1184	1261	942	989	588	611	233
31		1262	1342	1005	1053	627	651	248
32		1343	1425	1069	1119	668	692	265
33		1426	1511	1136	1187	709	735	281
34		1512	1599	1204	1257	752	778	299
35		1600	1690	1275	1329	796	823	316
36		1691	1783	1347	1403	842	869	335
37		1784	1879	1422	1479	888	917	353
38		1880	1977	1498	1557	936	965	373
39		1978	2078	1577	1637	985	1015	392
40		2079	2181	1657	1719	1036	1066	413
41		2182	2287	1740	1803	1087	1119	433
42		2288	2395	1824	1889	1140	1172	455
43		2396	2506	1911	1977	1194	1227	476
44		2507	2619	1999	2067	1250	1283	499
45		2620	2735	2090	2159	1306	1341	521
46		2736	2853	2182	2253	1364	1399	545
47		2854	2974	2277	2349	1423	1459	568
48		2975	3097	2373	2447	1484	1520	593
49		3098	3223	2472	2547	1545	1583	617

Continued on next page

Table 6.5: *The maximum number of hexagons in a cone patch for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
50		3224	3351	2572	2649	1608	1646	643

Table 6.6: *The minimum number of hexagons in a cone patch for given minimal side length, given number of pentagons and given type of boundary.*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
0	0	N/A	1	N/A	N/A	N/A	N/A	N/A
1		2	6	0	2	N/A	N/A	N/A
2		7	13	3	6	0	1	N/A
3		14	22	7	11	2	3	N/A
4		23	33	12	17	4	5	N/A
5		34	46	18	24	6	7	1
6		47	61	25	32	8	9	2
7		62	78	33	41	10	11	3
8		79	97	42	51	12	13	5
9		98	118	52	62	14	15	6
10		119	141	63	74	16	17	8
11		142	166	75	87	18	19	9
12		167	193	88	101	20	21	11
13		194	222	102	116	22	23	12
14		223	253	117	132	24	25	14
15		254	286	133	149	26	27	15

Continued on next page

Table 6.6: *The minimum number of hexagons in a cone patch for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

1	2	2	3	3	4	4	5
	s	n	s	n	s	n	
16	287	321	150	167	28	29	17
17	322	358	168	186	30	31	18
18	359	397	187	206	32	33	20
19	398	438	207	227	34	35	21
20	439	481	228	249	36	37	23
21	482	526	250	272	38	39	24
22	527	573	273	296	40	41	26
23	574	622	297	321	42	43	27
24	623	673	322	347	44	45	29
25	674	726	348	374	46	47	30
26	727	781	375	402	48	49	32
27	782	838	403	431	50	51	33
28	839	897	432	461	52	53	35
29	898	958	462	492	54	55	36
30	959	1021	493	524	56	57	38
31	1022	1086	525	557	58	59	39
32	1087	1153	558	591	60	61	41
33	1154	1222	592	626	62	63	42
34	1223	1293	627	662	64	65	44
35	1294	1366	663	699	66	67	45
36	1367	1441	700	737	68	69	47
37	1442	1518	738	776	70	71	48
38	1519	1597	777	816	72	73	50
39	1598	1678	817	857	74	75	51

Continued on next page

Table 6.6: *The minimum number of hexagons in a cone patch for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
40		1679	1761	858	899	76	77	53
41		1762	1846	900	942	78	79	54
42		1847	1933	943	986	80	81	56
43		1934	2022	987	1031	82	83	57
44		2023	2113	1032	1077	84	85	59
45		2114	2206	1078	1124	86	87	60
46		2207	2301	1125	1172	88	89	62
47		2302	2398	1173	1221	90	91	63
48		2399	2497	1222	1271	92	93	65
49		2498	2598	1272	1322	94	95	66
50		2599	2701	1323	1374	96	97	68

Table 6.7: *The maximum number of vertices in a cone patch for given minimal side length, given number of pentagons and given type of boundary.*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
0	5	N/A	11	N/A	N/A	N/A	N/A	N/A
1		14	27	10	18	N/A	N/A	N/A
2		30	47	21	33	12	17	N/A
3		50	73	38	52	22	29	N/A
4		76	103	57	75	36	43	N/A
5		106	139	82	102	50	61	16

Continued on next page

Table 6.7: *The maximum number of vertices in a cone patch for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
6		142	179	109	133	68	79	25
7		182	225	142	168	88	101	32
8		228	275	177	207	112	125	43
9		278	331	218	250	136	153	52
10		334	391	261	297	164	181	65
11		394	457	310	348	194	213	76
12		460	527	361	403	228	247	91
13		530	603	418	462	262	285	104
14		606	683	477	525	300	323	121
15		686	769	542	592	340	365	136
16		772	859	609	663	384	409	155
17		862	955	682	738	428	457	172
18		958	1055	757	817	476	505	193
19		1058	1161	838	900	526	557	212
20		1164	1271	921	987	580	611	235
21		1274	1387	1010	1078	634	669	256
22		1390	1507	1101	1173	692	727	281
23		1510	1633	1198	1272	752	789	304
24		1636	1763	1297	1375	816	853	331
25		1766	1899	1402	1482	880	921	356
26		1902	2039	1509	1593	948	989	385
27		2042	2185	1622	1708	1018	1061	412
28		2188	2335	1737	1827	1092	1135	443
29		2338	2491	1858	1950	1166	1213	472

Continued on next page

Table 6.7: *The maximum number of vertices in a cone patch for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
	s	n	s	n	s	n		
30	2494	2651	1981	2077	1244	1291	505	
31	2654	2817	2110	2208	1324	1373	536	
32	2820	2987	2241	2343	1408	1457	571	
33	2990	3163	2378	2482	1492	1545	604	
34	3166	3343	2517	2625	1580	1633	641	
35	3346	3529	2662	2772	1670	1725	676	
36	3532	3719	2809	2923	1764	1819	715	
37	3722	3915	2962	3078	1858	1917	752	
38	3918	4115	3117	3237	1956	2015	793	
39	4118	4321	3278	3400	2056	2117	832	
40	4324	4531	3441	3567	2160	2221	875	
41	4534	4747	3610	3738	2264	2329	916	
42	4750	4967	3781	3913	2372	2437	961	
43	4970	5193	3958	4092	2482	2549	1004	
44	5196	5423	4137	4275	2596	2663	1051	
45	5426	5659	4322	4462	2710	2781	1096	
46	5662	5899	4509	4653	2828	2899	1145	
47	5902	6145	4702	4848	2948	3021	1192	
48	6148	6395	4897	5047	3072	3145	1243	
49	6398	6651	5098	5250	3196	3273	1292	
50	6654	6911	5301	5457	3324	3401	1345	

Table 6.8: *The minimum number of vertices in a cone patch for given minimal side length, given number of pentagons and given type of boundary.*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
0	5	N/A	11	N/A	N/A	N/A	N/A	N/A
1		14	25	10	16	N/A	N/A	N/A
2		28	43	19	27	12	15	N/A
3		46	65	30	40	18	21	N/A
4		68	91	43	55	24	27	N/A
5		94	121	58	72	30	33	16
6		124	155	75	91	36	39	19
7		158	193	94	112	42	45	22
8		196	235	115	135	48	51	27
9		238	281	138	160	54	57	30
10		284	331	163	187	60	63	35
11		334	385	190	216	66	69	38
12		388	443	219	247	72	75	43
13		446	505	250	280	78	81	46
14		508	571	283	315	84	87	51
15		574	641	318	352	90	93	54
16		644	715	355	391	96	99	59
17		718	793	394	432	102	105	62
18		796	875	435	475	108	111	67
19		878	961	478	520	114	117	70
20		964	1051	523	567	120	123	75
21		1054	1145	570	616	126	129	78
22		1148	1243	619	667	132	135	83
23		1246	1345	670	720	138	141	86

Continued on next page

Table 6.8: *The minimum number of vertices in a cone patch for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
24		1348	1451	723	775	144	147	91
25		1454	1561	778	832	150	153	94
26		1564	1675	835	891	156	159	99
27		1678	1793	894	952	162	165	102
28		1796	1915	955	1015	168	171	107
29		1918	2041	1018	1080	174	177	110
30		2044	2171	1083	1147	180	183	115
31		2174	2305	1150	1216	186	189	118
32		2308	2443	1219	1287	192	195	123
33		2446	2585	1290	1360	198	201	126
34		2588	2731	1363	1435	204	207	131
35		2734	2881	1438	1512	210	213	134
36		2884	3035	1515	1591	216	219	139
37		3038	3193	1594	1672	222	225	142
38		3196	3355	1675	1755	228	231	147
39		3358	3521	1758	1840	234	237	150
40		3524	3691	1843	1927	240	243	155
41		3694	3865	1930	2016	246	249	158
42		3868	4043	2019	2107	252	255	163
43		4046	4225	2110	2200	258	261	166
44		4228	4411	2203	2295	264	267	171
45		4414	4601	2298	2392	270	273	174
46		4604	4795	2395	2491	276	279	179
47		4798	4993	2494	2592	282	285	182

Continued on next page

Table 6.8: *The minimum number of vertices in a cone patch for given minimal side length, given number of pentagons and given type of boundary. (Continued)*

	1	2	2	3	3	4	4	5
		s	n	s	n	s	n	
48		4996	5195	2595	2695	288	291	187
49		5198	5401	2698	2800	294	297	190
50		5404	5611	2803	2907	300	303	195

6.7 Number of hexagons in cone patches

In [47] Bornhöft et al. prove that for a given upper bound on the boundary length (i.e., the number of vertices in the boundary) and for a given number of pentagons, the patch with the maximum number of hexagons is given by an inner spiral that starts with all the pentagons. Note that this is not the same as the outer spiral in the previous chapter. An inner spiral is a patch P with f faces in which the faces can be numbered from 1 to f in a way that for $1 < n \leq f$ face n has a connected intersection with the subgraph induced by the faces $1, \dots, n-1$ which includes an edge of face $n-1$ and for $n > 2$ an edge of the smallest of the faces sharing a vertex with face $n-1$ that is in the boundary of the patch induced by the faces $1, \dots, n-1$. See [47] for more details. Figure 6.22 shows some examples of inner spirals.

The upper bounds proved in [47] for the number of hexagons per number of pentagons are given in Table 6.9. In Table 6.10, we express these upper bounds in function of the shortest side for symmetric and near-symmetric boundaries.

The importance of having a good upper bound for the number of hexagons, is that it also provides an upper bound on the number of vertices for a cone patch, which allows us to do efficient memory management when implementing the algorithm. Using the Euler formula we find

$$v - e + h + p + 1 = 2,$$

p	Upper bound
0	$h \leq \frac{b^2+12}{48}$
1	$h \leq \frac{b^2-25}{40}$
2	$h \leq \frac{b^2-64}{32}$
3	$h \leq \frac{b^2-81}{24}$
4	$h \leq \frac{b^2-100}{16}$
5	$h \leq \frac{b^2-113}{8}$

Table 6.9: The upper bounds on the number of hexagons in a pseudo-convex patch with p pentagons given in [47] expressed as functions of the length b of the boundary.

p	Upper bound	
	symmetric boundary	near-symmetric boundary
1	$h \leq \frac{5m^2-5m}{2}$	
2	$h \leq \frac{4m^2+4m-3}{2}$	$h \leq \frac{16m^2+40m+9}{8}$
3	$h \leq \frac{3m^2+3m-6}{2}$	$h \leq \frac{6m^2+14m-7}{4}$
4	$h \leq m^2 + m - 6$	$h \leq \frac{4m^2+8m-21}{4}$
5	$h \leq \frac{m^2+m-28}{2}$	

Table 6.10: The upper bounds from Table 6.9 expressed as function of the length of the shortest side.

with v the number of vertices, e the number of edges, h the number of hexagons and p the number of pentagons. The number of edges can be counted as follows

$$e = \frac{6h + 5p + b}{2}$$

in which b stands for the length of the boundary. Substituting this second formula in the first formula, we find

$$h = \frac{2v - 3p - b - 2}{4}.$$

We can express the length of the boundary for symmetric and near-symmetric patches in function of the length m of the shortest side of the patch. For symmetric boundaries this is

$$b = (6 - p)(2m + 1),$$

and for near-symmetric boundaries

$$b = (6 - p - 1)(2m + 3) + 2m + 1.$$

Again substituting these formulas into the formula above we find the following equality for symmetric boundaries

$$h = \frac{v - p - (6 - p)m - 4}{2},$$

and for near-symmetric boundaries

$$h = \frac{v - (6 - p)m - 9}{2}.$$

Bornhöft et al. were looking at general patches and actually worked out a lower bound for the boundary given the number of hexagons and pentagons. A cone patch has some extra restrictions. First, although it would indeed give the patch with the most hexagons for that boundary, it is not possible – except for a few small cases – to place all pentagons in the centre, because the boundary needs to share an edge with a pentagon. Using this extra restriction we can find a new upper bound on the number of hexagons in a cone patch that performs better or in some very few cases at least as good as the old bound. Furthermore even these new bounds will in most

cases not be sharp, because the patch that is obtained by first placing all pentagons and then adding hexagons is not guaranteed to have a symmetric or near-symmetric boundary.

In a cone patch the boundary shares at least one edge with at least one pentagon. Choose such an edge e and insert a vertex into it. The resulting new patch will have one pentagon less and one hexagon more than the original patch. Furthermore, its boundary length will be one larger than that of the original patch. Using the old bounds, we can now calculate new bounds for these modified patches and deduce from them better bounds for the original patches.

If we denote the upper bound from [47] for the number of hexagons in a patch with b boundary vertices and p pentagons by $f(b, p)$, then we can write down to new upper bounds in case there are pentagons that lie at the boundary as follows. If there are k pentagons in the boundary and we use the construction above k times, we find the following upper bound on the number of hexagons:

$$f(b + k, p - k) - k. \quad (6.89)$$

There is only one patch in case of 1 pentagon and it does not contain any hexagons, therefore we leave this case out for the rest of the discussion.

In a patch with two pentagons and a symmetric boundary, we have already shown that the two pentagons always lie at the boundary. Using the formula above, with $b = 8m + 4$, $k = 2$ and $p = 2$, we find

$$h \leq \frac{4m^2 + 6m + 3}{3}. \quad (6.90)$$

For the other types, we can only assume that there is at least one pentagon that lies at the boundary. The new upper bounds for the different types of patches are given in Table 6.11

As we noted before these upper bounds are not necessarily sharp, because placing all but one pentagon at the center of the patch might not be possible for that specific boundary. In one case however this bound is sharp. This case is that of 5 pentagons. The patch shown in Figure 6.22.a has a spiral code that starts with 4 pentagons and then 2 hexagons and the boundary code is $2(23)^4 2(23)^1$. If we remove a vertex of

p	Upper bound	
	symmetric boundary	near-symmetric boundary
2	$h \leq \frac{4m^2+6m+3}{3}$	$h \leq \frac{8m^2+22m+7}{5}$
3	$h \leq \frac{9m^2+12m-20}{8}$	$h \leq \frac{9m^2+24m-8}{8}$
4	$h \leq \frac{2m^2+3m-9}{3}$	$h \leq \frac{2m^2+5m-10}{3}$
5	$h \leq \frac{m^2+2m-28}{4}$	

Table 6.11: *The new upper bounds for the number of hexagons in a cone patch with p pentagons.*

degree 2 in the last hexagon added, the new boundary code would be $2(23)^5$. As can be seen in Lemma 6.3.3 adding a layer of hexagons corresponds to adding one to each side, so if we add n layers of hexagons to this patch the boundary code would be $2(23)^{4+n}2(23)^{1+n}$, and after removal of a vertex of degree 2 in the last hexagon we get a patch with boundary $2(23)^{5+2n}$. Analogously we find that the other patch leads to patches with boundary $2(23)^{6+2n}$. Since both these patches have a valid symmetric boundary for the case with 5 pentagons, and all have by construction a pentagon in the boundary, these are valid cone patches. The first patch is the prototype for patches with an odd number of 3's in the boundary, the second is the prototype for the patches with an even number of 3's. So for each boundary corresponding to a patch with 5 pentagons, there exists a patch from this family, and so the upper bound is in this case sharp.

In Corollary 6.3.27 and Corollary 6.3.28, we saw that we can give very concrete descriptions of the patches in case of 2 pentagons. This immediately suggests that we can use these descriptions to find a sharp upper bound for this case.

First consider a symmetric boundary $(2(23)^m)^4$. This corresponds to an initial patch with parameters l and k , such that $l - k \pmod 3 = 0$. Without loss of generality we can assume that $l > k$. The number of hexagons in the initial patch is equal to $(l+1)(k+1) - 2$. To both sides of length l we add $i = \frac{l-k}{3}$ times a side of hexagons.

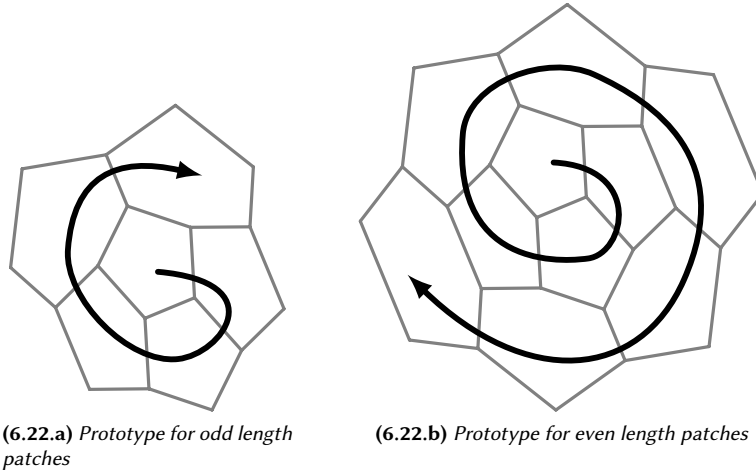


Figure 6.22: The prototypes for cone patches with 5 pentagons and the maximum number of hexagons

This corresponds to $2 \sum_{j=1}^i (l - j + 1)$ hexagons. Putting this together, we find that the cone patch contains h hexagons with

$$\begin{aligned}
 h &= (l + 1)(k + 1) - 2 + 2 \sum_{j=1}^i (l - j + 1) \\
 &= \frac{5kl + 12l + 6k - 9 + 5l^2 - k^2}{9} \\
 &= \frac{5m^2 + 8m - 4 - k^2}{4}.
 \end{aligned}$$

The last equality is due to $m = \frac{2l+k}{3}$. It is clear that this last formula for the number of hexagons is maximal when k^2 is minimal. For m even this means $k = 0$ which gives us

$$h = \frac{5m^2 + 8m - 4}{4}, \quad (6.91)$$

and for m odd this means $k = 1$ which gives us

$$h = \frac{5m^2 + 8m - 5}{4}. \quad (6.92)$$

These are sharp upper bounds for the number of hexagons in symmetric patches with 2 pentagons.

By subtracting the actual maxima for this case in 6.91 and 6.92 from the upper bound 6.90, we find that that upper bound is too large by $\frac{m^2+24}{12}$ for even m , and by $\frac{m^2+27}{12}$ for odd m .

Next consider a near-symmetric boundary $(2(23)^{m+1})^3 2(23)^m$ and add the restriction that the two pentagons share an edge with the boundary. This corresponds to an initial patch with parameters l and k ($l > k$), such that $l - k \pmod 3 = 1$ and $m = \frac{2l+k-2}{3}$. Again the number of hexagons in the initial patch is equal to $(l+1)(k+1) - 2$. To both sides of length l we add $i = \frac{l-k-1}{3}$ times a side of hexagons and we finish by adding an extra side of hexagons to one of these sides. This corresponds to a total of $2 \sum_{j=1}^i (l-j+1) + m + 1$ hexagons. Putting this together, we find that the cone patch contains h hexagons with

$$\begin{aligned} h &= (l+1)(k+1) - 2 + 2 \sum_{j=1}^i (l-j+1) + m + 1 \\ &= kl + l + k - 1 + i(l+l-i+1) + m + 1 \\ &= \frac{5m^2 + 16m + 4 - k^2}{4}. \end{aligned}$$

The last equality is due to $l = \frac{3m-k+2}{2}$ and $i = \frac{m-k}{2}$. Again this is maximal when k^2 is minimal. For m even this means $k = 0$ which gives us

$$h = \frac{5m^2 + 16m + 4}{4}, \tag{6.93}$$

and for m odd this means $k = 1$ which gives us

$$h = \frac{5m^2 + 16m + 3}{4}. \tag{6.94}$$

When we apply formula 6.89 for this case, we have that $b = 8s + 12$, $k = 2$ and $p = 2$, and we find

$$h \leq \frac{16m^2 + 48m + 15}{12}.$$

By subtracting the actual maxima for this case in 6.93 and 6.94 from this upper bound we find that this upper bound is too large by $\frac{m^2+3}{12}$ for even m , and by $\frac{m^2+4}{12}$ for odd m .

For a near-symmetric boundary we also have the possibility that only one pentagon shares an edge with the boundary. Assume this is the case and the boundary is $(2(23)^{m+1})^3 2(23)^m$. This corresponds to an initial patch with parameters l and k ($l > k$), such that $l - k \bmod 3 = 2$ and $m = \frac{2l+k-1}{3}$. The number of hexagons in the initial patch is equal to $(l+1)(k+1) - 2$. To both sides of length l we add $i = \frac{l-k+1}{3}$ times a side of hexagons and we finish by adding one side of hexagons to one of the other sides. This corresponds to a total of $2 \sum_{j=1}^i (l-j+1) + m + 1$ hexagons. Putting this together, we find that the cone patch contains h hexagons with

$$\begin{aligned} h &= (l+1)(k+1) - 2 + 2 \sum_{j=1}^i (l-j+1) + m + 1 \\ &= kl + l + k - 1 + i(l+l-i+1) + m + 1 \\ &= \frac{5m^2 + 18m + 5 - k^2}{4}. \end{aligned}$$

The last equality is due to $l = \frac{3m-k+1}{2}$ and $i = \frac{m-k+1}{2}$. This also is maximal when k^2 is minimal. For m even this means $k = 0$ which gives us

$$h = \frac{5m^2 + 18m + 5}{4}, \quad (6.95)$$

and for m odd this means $k = 1$ which gives us

$$h = \frac{5m^2 + 18m + 4}{4}. \quad (6.96)$$

By subtracting the actual maxima for this case in 6.95 and 6.96 from the upper bound in Table 6.11 we find that that upper bound is too large by $\frac{7m^2-2m+3}{20}$ for even m , and by $\frac{7m^2-2m+8}{20}$ for odd m .

It is easy to verify that for a near-symmetric boundary the maximum number of hexagons for the case with two pentagons that share an edge with the boundary is always smaller than the maximum for the case with only one pentagon that shares an

edge with the boundary. This means that we find the following sharp upper bounds for the number of hexagons in a symmetric patch with two pentagons

$$h \leq \left\lfloor \frac{5m^2 + 8m - 4}{4} \right\rfloor$$

and for the number of hexagons in a near-symmetric patch with two pentagons

$$h \leq \left\lfloor \frac{5m^2 + 18m + 5}{4} \right\rfloor.$$

A Software

Option	Function
-o	Output the tilings of O_T
-a	Output the tilings of A_T
-t	Output the tilings of A_T with only translational symmetry
-c	Output the translation-only covers of the tilings
-m	Output minimal symbols
-p	Output periodic graphs instead of Delaney-Dress symbols
-h	Print a detailed help message

Table A.1: *The options available when running the program azul.*

The programs here are listed in alphabetic order.

A.1 azul

The program `azul` is the implementation of the algorithm described in Chapter 4 and is written in C. This algorithm was developed for a very specific goal, and so this program has few configuration options. We list the main options in Table A.1.

This program can be downloaded from [63].

A.2 cone

The program `cone` is the implementation of the algorithm described in Chapter 5, but can only be used to generate cone patches as described in Chapter 6. This program has been written in C.

This program is available as part of CaGe[57, 61]. CaGe provides a simple graphical interface to specify which cone patches need to be generated. CaGe is written in Java and C, and is available for Linux and Mac OS X.

Option	Function
-h, -help	Print this help and return.
-l, -listfile <i>file</i>	Only start the generation based on the block lists in <i>file</i> .
-o, -output <i>type</i>	Specifies the export format where <i>type</i> is one of <i>c, code</i> code (depends on the generated type) <i>h, human</i> human-readable output <i>n, none</i> no output: only count (default)
-m, -modulo <i>r:n</i>	Split the generation in multiple parts. The generation is split into <i>n</i> parts and only part <i>r</i> is generated. The number <i>n</i> needs to be an integer larger than 0, and <i>r</i> should be a positive integer smaller than <i>n</i> .
-restrictiononly	calculate the restrictions for the Delaney-Dress symbol, but do not generate any structures.

Table A.2: *The general options available when running the program ddgraphs.*

A.3 ddgraphs

The program `ddgraphs` is the implementation of the algorithm described in Chapter 3 and is written in C. This program is currently not yet available online, but can be obtained from the author.

This program can be used in different ways depending on the desired result. These are the three ways to call the program from the command line:

- `./ddgraphs [options] n`
Generate graphs with *n* vertices.
- `./ddgraphs -l file [options]`
Generate graphs based on the block lists in *file*.
- `./ddgraphs -s [options]`
Generate Delaney-Dress symbols.

We list the main options in Table A.2, Table A.3 and Table A.4.

Option	Function
-L, -lists	Generate block lists.
-t, -marked	Generate cubic pregraphs with a marked 2-factor where each component is a quotient of a 4-cycle.
-c, -coloured	Generate Delaney-Dress graphs, i.e., graphs that are the underlying graphs of Delaney-Dress symbols and in which the edges are coloured with the colours 0, 1 or 2.
-s, -symbols	Generate Delaney-Dress symbols.

Table A.3: *The options available to specify the type of the generated structures when running the program `ddgraphs`. By default, `ddgraphs` will generate C_4^q -markable pregraphs.*

A.3.1 Examples

To see how many C_4^q -markable cubic pregraphs on 8 vertices exist, the following command can be used.

```
./ddgraphs 8
```

The output will then be

```
Generating cubic pregraphs that have a C4q 2-factor and with 8 vertices.
Found 99 block lists.
Found 105 cubic pregraphs that have a C4q 2-factor.
CPU time: 0.0 seconds.
```

To know to how many C_4^q -marked cubic pregraphs on 8 vertices this leads, the theoretical results from Theorem 3.1.2 can be used, but it can also be done using the next command.

```
./ddgraphs -t 8
```

The output will then be

```
Generating cubic pregraphs with marked 2-factors and 8 vertices.
Found 103 block lists.
Found 109 cubic pregraphs with marked 2-factors.
CPU time: 0.0 seconds.
```

Option	Function
-b, -bipartite	Only generate Delaney-Dress graphs that are bipartite.
-O, -orientable	Only generate Delaney-Dress graphs that (may) correspond to orientable tilings.
-R, -requiredface	Add a face size to the list of required faces.
-F, -forbiddenface	Add a face size to the list of forbidden faces.
-r, -requiredvertex	Add a vertex degree to the list of required vertices.
-f, -forbiddenvertex	Add a vertex degree to the list of forbidden vertices.
-maxfacecount	Specify the maximum number of face orbits in the tiling.
-minfacecount	Specify the minimum number of face orbits in the tiling.
-maxvertexcount	Specify the maximum number of vertex orbits in the tiling.
-minvertexcount	Specify the minimum number of vertex orbits in the tiling.
-minfacesize	Specify the minimum size of a face in the tiling.
-maxfacesize	Specify the maximum size of a face in the tiling.
-minvertexdegree	Specify the minimum degree of a vertex in the tiling.
-maxvertexdegree	Specify the maximum degree of a vertex in the tiling.
-n, -minvertices	Specify the minimum number of vertices in the Delaney-Dress graph.
-N, -maxvertices	Specify the maximum number of vertices in the Delaney-Dress graph.

Table A.4: *The options available to constraint the generated structures when running the program `ddgraphs`.*

The number of Delaney-Dress graph that have 8 vertices, can be enumerated using the following command.

```
./ddgraphs -c 8
```

The output will then be

```
Found 103 block lists.
Found 109 cubic pregraphs with marked 2-factors.
Found 315 Delaney-Dress graphs.
CPU time: 0.0 seconds.
```

To generate all Delaney-Dress symbols for tile-transitive tilings, the usage is a bit different. One can do this with the following command

```
./ddgraphs -s --maxfacecount 1
```

The output will then be

Generating Delaney-Dress symbols with 1 to 12 vertices.

Improved parameter bounds

```
-----
Number of face orbits lies in [1,1].
Number of vertex orbits lies in [1,12].
Number of edge orbits lies in [1,12].
Face sizes lie in [3,144].
Vertex degrees lie in [3,144].
Required face sizes are [].
Forbidden face sizes are [].
Required vertex degrees are [].
Forbidden vertex degrees are [].
```

```
Found 93 Delaney-Dress symbols.
CPU time: 1.8 seconds.
```

A.4 PGVisualizer

PGVisualizer is closely related with azul in that it is the program used to visualise the periodic graphs that can be output by azul. The main aim of PGVisualizer is

a straight-forward viewer for files in the PG format. It is written in Java and therefore should be portable across several operating systems.

This program and its manual can be downloaded from [63].

A.5 pregraphs

The program `pregraphs` is the implementation of the algorithm described in Chapter 2 and is written in C. This program can be downloaded from [64].

The program `pregraphs` calculates pregraphs of a given order n . The value for n needs to be specified on the command line after specifying the options. We list the main options in Table A.5.

A.5.1 Examples

To count all bipartite multigraphs with 4 vertices, the following command can be used:

```
./pregraphs -MB 4
```

The output will then be:

```
Generating bipartite multigraphs with 4 vertices.
Found 1 pregraph with 4 vertices.
CPU time: 0.0 seconds.
```

To view this graph as a simple table, the output option can be used to specify a human-readable format:

```
./pregraphs -MB -oh 4
```

The output will then be:

```
Generating bipartite multigraphs with 4 vertices.
=====
| Graph number:          1 |
| Number of vertices:   4 |
=====
|  1 ||    4 |    2 |    2 ||
```

Option	Function
-h	Print help and return.
-i	Causes pregraphs to print extra info about the generated structures.
-L	Allow loops.
-S	Allow semi-edges.
-M	Allow multi-edges.
-C	Only generate 3-edge-colourable pregraphs.
-B	Only generate bipartite pregraphs.
-q	Only generate pregraphs that have a 2-factor where each component is the quotient of a 4-cycle.
-4	Only generate pregraphs that have a 2-factor where each component is a 4-cycle.
-P	Only generate the corresponding pregraph primitives.
-I	Only start the generation from the files provided by the input file (see -F).
-f <i>file</i>	Specifies the output file. If absent, the output is written to standard out.
-F <i>file</i>	Specifies the input file. If absent, the input is taken from standard in. This option is only used if -I is used.
-o <i>c</i>	Specifies the export format where <i>c</i> is one of <i>c</i> pregraph code (or multicode if -P is used) <i>h</i> human-readable output in tabular format <i>n</i> no output : only count (default)
-m <i>r:m[:d]</i>	Split the generation into <i>m</i> parts at depth <i>d</i> and only execute part <i>r</i> . The default for <i>d</i> is 0.

Table A.5: *The options available when running the program pregraphs.*

```

|  2 ||   1 |   3 |   1 ||
|  3 ||   2 |   4 |   4 ||
|  4 ||   3 |   1 |   3 ||
=====

```

```

Found 1 pregraph with 4 vertices.
CPU time: 0.0 seconds.

```

To store this graph in a file named `bipartite_multigraph.code`, the following command can be used:

```
./pregraphs -MB -oc 4 > bipartite_multigraph.code
```

To check whether this graph has a 2-factor where each component is a quotient of a 4-cycle, the following command can be used:

```
./pregraphs -MBq -oc 4
```

The output will then be:

```

Generating bipartite multigraphs with 4 vertices and filtering
graphs that have a 2-factor where each component is a
quotient of a 4-cycle.
Found 1 pregraph with 4 vertices.
CPU time: 0.0 seconds.

```

A.6 PregraphViewer

PregraphViewer is a program that can be used to visualise several formats that can be output by `pregraphs` and `ddgraphs`. It is written in Java and therefore should be portable across several operating systems.

This program and its manual can be downloaded from [64].

B

File formats

The generators that were implemented in this thesis use a wide range of file formats. We give an overview of the formats used. Some are well established formats, others are new formats.

B.1 (Pre)graphs

B.1.1 Multi_code

This is a well established format for coding simple graphs. The generator `pregraph` is capable of reading and writing `multi_code`.

The file starts with a header. This header is one of `>>multi_code<<`, `>>multi_code le<<` or `>>multi_code be<<`. The first two headers mean that little endian is used, the third that big endian is being used (see later). This code is binary. Vertices are numbered 1 to n . To each vertex x there is a list of neighbours with higher numbers than x , followed by a zero. The last list is always empty (no neighbours of n with a higher number than n), so the last "list" is not followed by a zero. After the last byte the next graph follows.

Normally the entries in the code are of type "unsigned char". But if the number of vertices for one graph is higher than 252, then the code for this graph begins with a zero (unsigned char) and then each following entry is of type "unsigned short" (2 bytes). In this case it makes a difference whether the entry is stored in little endian or big endian style. If the file contains no graphs with more than 252 vertices, then it is not important if the style is little endian or big endian.

B.1.2 Pregraph_code

The standard output format for the generator `pregraph` and the related filters is `pregraph_code`. This code is based upon `multi_code`. It is a binary code structured as follows.

The convention is to give the filename the extension `.code`. The file starts with a header. This header is one of `>>pregraph_code<<`, `>>pregraph_code le<<` or `>>pregraph_code be<<`. The first two headers mean that little endian is used, the third that big endian is being used (see later). If the graph contains less than 255

vertices, then the first entry is the order of the graph. The vertices are numbered from 1 to n (where n is the order of the graph) and for each vertex x there is a list of neighbours of x with higher numbers than x . If a vertex is incident to any semi-edges, then $n + 1$ is added to the list for each semi-edge that is incident to x . Each list is closed by a zero. It is possible that some lists are empty. If the graph contains 255 or more vertices, then the first entry is a zero. After that the same format as with the smaller graphs is used, but now each entry consists of two bytes instead of one byte. These two-byte-numbers are in little endian or big endian depending on the header of the file. After the last entry the following graph follows immediately.

B.1.3 `Pregraphcolor_code`

The generator `ddgraphs` outputs coloured cubic pregraphs in the `pregraphcolor_code` format. This code is based upon `pregraph_code`. It is a binary code structured as follows.

The convention is to give the filename the extension `.code`. The file starts with a header which is one of `>>pregraphcolor_code<<`, `>>pregraphcolor_code le<<` or `>>pregraphcolor_code be<<`. The first two headers mean that little endian is used, the third that big endian is being used (see later). If the graph contains less than 255 vertices, then the first entry is the order of the graph. The vertices are numbered from 1 to n (where n is the order of the graph) and for each vertex x there is a list of neighbours of x with higher numbers than x . If a vertex is incident to any semi-edges, then $n + 1$ is added to the list for each semi-edge that is incident to x . This list is closed by a zero. It is possible that some lists are empty. After the list of neighbours follows a list with colours that has the same length as the list of neighbours. For each neighbour a colour is given for the corresponding edge. This list is also closed by a zero. Then the lists for the next vertex follows. If the graph contains 255 or more vertices, then the first entry is a zero. After that the same format as with the smaller graphs is used, but now each entry consists of two bytes instead of one byte. These two-byte-numbers are in little endian or big endian depending on the header of the file. After the last entry the following graph follows immediately.

B.1.4 Embedded pregraphs

Pregraph Viewer introduces a new file format: embedded pregraphs. This is an XML-based format to store the pregraphs together with an embedding. The default extension for these files is *.epxml*.

The root element is *embeddedpregraphs*. This element contains one or more *embeddedpregraph* elements. Each of these contains exactly one *vertices* element and one *edges* element. For each vertex there should be a *vertex* element in the *vertices* element describing the position of the vertex. For each semi-edge there should be a *semiedgevertex* element in the *vertices* element describing the end point of the semi-edge. For each loop there should be a *loopvertex* element in the *vertices* element describing the tip of the loop. Each of these elements has three attributes: an *X* attribute giving the horizontal position, an *Y* attribute giving the vertical position, and an *id* attribute giving an unique non-zero number for the vertex. All numbers from 1 to the order of the graph + the number of semi-edges + the number of loops need to be used. For each edge there should be an *edge* element in the *edges* element. Each of these elements has three attributes: a *vertex1* attribute and *vertex2* attribute giving the two vertices incident to this edge (use the id's to identify the vertices) and a *multiplicity* attribute giving the multiplicity of the edge. For loops use the vertex that is incident to the loop and the loop vertex describing the tip of the loop. For semi-edges use the vertex incident to the semi-edge and the semi-edge vertex describing the end point of the semi-edge.

B.2 Planar graphs

Cone patches are planar graphs. The program cone outputs the cone patches in the file format *planar_code*.

B.2.1 Planar_code

This is a binary code which is a bit similar to *multi_code*. The file starts with a header. This header is one of `>>planar_code<<`, `>>planar_code le<<` or `>>planar_code be<<`. The first two headers mean that little endian is used, the

third that big endian is being used (see later). First the number of vertices of the graph appears. Vertices are numbered 1 to n . First the neighbours of vertex 1 are listed in clockwise direction, followed by a zero. Then the same continues with vertex 2,3,..., n . After the last zero, the next graph follows.

Normally the entries in the code are of type “unsigned char”. But if the number of vertices for one graph is higher than 252, then the code for this graph begins with a zero (unsigned char) and then each following entry is of type “unsigned short” (2 bytes). In this case it makes a difference whether the entry is stored in little endian or big endian style. If the file contains no graphs with more than 252 vertices, then it is not important if the style is little endian or big endian.

B.3 Delaney-Dress symbols

Both the program `azul` and the program `ddgraphs` output Delaney-Dress symbols in a common file format [62].

Delaney-Dress symbol files use the suffix `.ds`. Each Delaney-Dress symbol is represented by a sequence of lines, the first one of which must start with an opening angular bracket '`<`', whereas the final one must end with a closing bracket '`>`'. Here is a simple example:

```
<1.1:3 2:1 2 3,3 2,2 3:8 4,3>
```

Each entry consists of four sections separated by colons. Thus, ignoring the delimiting brackets, the example entry consists of the sections

```
'1.1', '3 2', '1 2 3,3 2,2 3' and '8 4,3'.
```

The first section is just for bookkeeping. It must consist of two arbitrary positive integers, separated by a dot. The first number in the second section is the size of the Delaney-Dress symbol and the second one its dimension d . If the dimension's missing, it's assumed to be 2.

By convention, vertices (chambers) are numbered from 1 up to the size n of the symbol, while indexes go from 0 to d .

The third section consist of $d + 1$ comma-separated parts, each defining one of the functions σ_0 up to σ_d (in this case, σ_2). Here we have the parts

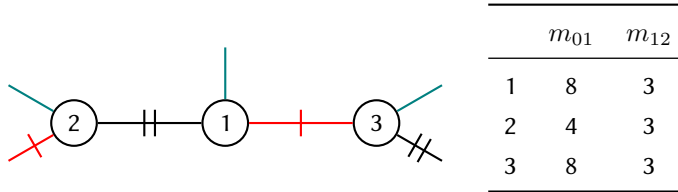
'1 2 3', '3 2' and '2 3'.

Within the i -th part, the values for the function σ_i are given in order, starting from $\sigma_i(1)$ and going up to $\sigma_i(d)$. If $\sigma_i(a) = b$ for $b > a$, we omit the specification of $\sigma_i(b)$, because it is already clear that it must be a . Thus, the first part '1 2 3' defines $\sigma_0(1) = 1, \sigma_0(2) = 2$ and $\sigma_0(3) = 3$. The second part means that $\sigma_1(1) = 3$ and, consequently $\sigma_1(3) = 1$, and $\sigma_1(2) = 2$. Finally the third part means that $\sigma_2(1) = 2$ and, consequently $\sigma_2(2) = 1$, and the second number gives the value for $\sigma_2(3)$, and not for $\sigma_2(2)$, so we have that $\sigma_2(3) = 3$.

The fourth section consists of d parts, in this case

'8 4' and '3'.

These define the values of the functions m_{01} and m_{12} . The number m_{ij} only appears once for each i, j -component. The vertices 1 and 3 are in the same 01-component, so $m_{01}(1) = m_{01}(3) = 8$ and $m_{01}(2) = 4$. There is only one 12-component, so $m_{12} \equiv 3$. The complete Delaney-Dress symbol thus looks like this:



B.4 Periodic graphs

The embedded azulenoid tilings are saved in a PG-file. A PG-file has the extension *.pg* and contains at most one periodic graph per line. A line can also consist entirely of a comment when it starts with a #. A single periodic graph looks like this:

$s_1 | s_2 | s_3 | s_4 [| s_5] [\# \text{ info }] [\# \text{ info }] \dots$

It consists of four or five strings (the fifth is optional) separated by `|`'s (pipes). At the end there is the possibility to add additional information such as comments or face highlighting.

- s_1

This string only contains one integer number. This is the order of the repetitive part of the graph.

- $s_2 = s_2^a s_2^b [s_2^c]$

This string consists of two or three real numbers separated by spaces. The first is length of the horizontal side of the domain, the second is the length of the vertical side of the domain. If there is a third number present then this will be used as the upper left angle of the domain.

- $s_3 = x_1 y_1; x_2 y_2; x_3 y_3; \dots$

This strings contains the coordinates of the vertices. It contains several parts separated by semicolons. The number of parts has to be equal to the order given in s_1 . Each part contains two real numbers separated by a space.

- $s_4 = \text{start}_1 \text{end}_1 X_1 Y_1; \text{start}_2 \text{end}_2 X_2 Y_2; \dots$

This string contains an entry for each edge. The entries are separated by semicolons and each entry consists of four integer numbers separated by spaces. The first number gives the start vertex, the second number the end vertex and the third and fourth number are the X and Y coordinate of the domain to which the end vertex belongs.

- $s_5 = f_1; f_2; f_3; \dots$

This fifth string is optional and it contains information about the faces. It contains an entry for each face. The entries are separated by semicolons and each entry consists of a row of integer numbers separated by spaces. Each number gives the index of a vertex.

$$f_i = v_1^i; v_2^i; v_3^i; \dots$$

- When the first word in one of the additional info string is **facehighlight** then the rest of the string is interpreted as face highlighting information and the string should be formatted as follows

$$\text{facehighlight } F_1 \ c_{F_1} \ F_2 \ c_{F_2} \ \dots$$

In this F_i stands for the sequence number of a face in the string s_5 and c_{F_i} stands for the colour of that face given as an integer in which bits 24-31 are alpha, 16-23 are red, 8-15 are green and 0-7 are blue, i.e., as defined by the function `getRGB()` of the class `Color` in Java 5.

- When the first word in one of the additional info string is **symbol** then the rest of the string is interpreted as a Delaney-Dress symbol in .ds-format.
- When the first word in one of the additional info string is **group** then the rest of the string is interpreted as the name of a wallpaper group. The possibilities are P6MM, P6, P4MM, P4GM, P4, P31M, P3M1, P3, C2MM, P2MM, P2MG, P2GG, P2, CM, PM, PG, P1, UNKNOWN. Any other value will be interpreted as UNKNOWN.

B.5 Block lists

The program `ddgraphs` can read and write block lists in the block list file format. Files in this format are human-readable to a certain extent: the format is text-based and not binary.

A block list file consists of a series of numbers representing the block lists. These numbers are separated using white spaces. Extra white spaces are ignored, so they can be used to group logical units of the list for a human reader.

Each list starts with a number that corresponds to the number of vertices in the different blocks in that list. After that follows different block definitions until the sum of the vertices in the blocks is equal to the specified order. After the last block definition the block list is closed by a 0. The next block list can start immediately after this zero.

B.5.1 Block definitions

Each block definition consists of multiple numbers. The first number is always the type of the block definition. There are six types and we will discuss them type by type. We refer to Chapter 3 for more information on the abbreviations which are used.

B.5.1.1 Type 1 block definitions

Block definitions of this type consist of 4 numbers: the type, the family, the parameter and the count.

	Block definition			Explanation
1	0	n	m	m copies of $H(n)$
1	1	n	m	m copies of $LH(n)$
1	2	n	m	m copies of $DLH(n)$
1	3	n	m	m copies of $DC(n)$
1	4	n	m	m copies of $DHB(n)$
1	5	n	m	m copies of $OHB(n)$
1	6	n	m	m copies of $DLB(n)$
1	7	n	m	m copies of $OLB(n)$
1	8	n	m	m copies of $LDC(n)$
1	9	n	m	m copies of $LDHB(n)$
1	10	n	m	m copies of $LOHB(n)$
1	11	n	m	m copies of $LDLB(n)$

B.5.1.2 Type 2 block definitions

Block definitions of this type consist of 4 numbers: the type, the family, the parameter and the count.

Block definition				Explanation
2	0	n	m	m copies of PC(n)
2	1	n	m	m copies of LPC(n)

B.5.1.3 Type 3 block definitions

Block definitions of this type consist of 4 numbers: the type, the family, the parameter and the count.

Block definition				Explanation
3	0	n	m	m copies of BW(n)
3	1	n	m	m copies of LBW(n)

B.5.1.4 Type 4 block definitions

Block definitions of this type consist of 2 numbers: the type and the count.

Block definition		Explanation
4	m	m copies of Q4

B.5.1.5 Type 5 block definitions

Block definitions of this type consist of 3 numbers: the type, the family and the parameter. If a block list contains this block definition, it may only contain one block definition.

Block definition			Explanation
5	0	0	T
5	1	n	DLPC(n)
5	2	n	PN(n)

B.5.1.6 Type 6 block definitions

Block definitions of this type consist of 3 numbers: the type, the family and the parameter. If a block list contains this block definition, it may only contain one block definition.

Block definition			Explanation
6	0	n	DLBW(n)
6	1	n	BWN(n)
6	2	n	DLDC(n)
6	3	n	ML(n)
6	4	n	P(n)
6	5	n	DLDLB(n)
6	6	n	CLH(n)
6	7	n	DDHB(n)
6	8	n	DLDHB(n)

B.5.2 Example

The program `ddgraphs` prints one block list per line and inserts a double space between the different block definitions. We will also use this convention here, since it makes the file more human-readable.

```
4 3 0 1 1 4 2 0
8 2 0 1 2 3 0 1 2 0
```

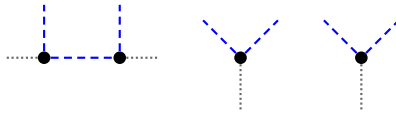


Figure B.1: *The block list that corresponds to the block list file containing 4 3 0 1 1 4 2 0.*



Figure B.2: *The block list that corresponds to the block list file containing 8 2 0 1 2 3 0 1 2 0.*

The file above contains two block lists. The first list has a total of four vertices and the second list has eight vertices. The block lists are shown in Figure B.1 and Figure B.2.

Bibliography

- [1] L. Euler. Demonstratio nonnullarum insignium proprietatum, quibus solida hedris planis inclusa sunt praedita. *Novi Commentarii Academiae Scientiarum Petropolitanae*, 4:140–160, (1752/3) 1758.
- [2] C. Jordan. Cours d'analyse, 1887.
- [3] R. Vaidyanathaswamy. Integer-roots of the unit matrix. *J. London Math. Soc.*, 3:p.121 – 124, 1928.
- [4] R. Vaidyanathaswamy. On the possible periods of integer matrices. *J. London Math. Soc.*, 3:p. 268 – 272, 1928.
- [5] D. Blanuša. Problem ceteriju boja (The problem of four colours). *Hrvatsko Prirodoslovno Društvo Glasnik Math.-Fiz. Astr., Ser. II*, 1:31–42, 1946.
- [6] C. E. Shannon. A theorem on coloring the lines of a network. *J. Math. Physics*, 28:148–151, 1949.
- [7] P. Erdős and T. Gallai. Graphs with prescribed degrees of vertices (Hungarian). *Mat. Lapok*, 11:264–274, 1960.
- [8] D. Suprunenko. On the order of an element of a group of integral matrices. *Dokl. Akad. Nauk. BSSR*, 7:p. 221 – 223, 1963.
- [9] W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, s3-13(1):743–767, 1963.

- [10] V. Vizing. On an estimate of the chromatic class of a p -graph. *Disk. Anal.*, 3:25–30, 1964.
- [11] A. Balaban. Valence-isomerism of cyclopolyenes. *Revue Roumaine de chimie* 11, pages 1097–1116, 1966. No. 12 (1967) p.103 (erratum).
- [12] B. Owens and H. Trent. On determining minimal singularities for the realization of an incidence sequence. *SIAM Journal on Applied Mathematics*, 15:406 – 418, 1967.
- [13] D. Kirby. Integer matrices of finite order. *Rend. Mat.*, 2:p. 403 – 408, 1969.
- [14] B. G. Buchanan and J. Lederberg. The heuristic DENDRAL program for explaining empirical data. Technical report, Stanford, CA, USA, 1971.
- [15] B. Grünbaum and G. Shephard. Isotoxal tilings. *Pacific Journal of Mathematics*, 76(2):407 – 430, 1978.
- [16] R. K. Lindsay, B. G. Buchanan, E. A. Feigenbaum, and J. Lederberg. *Artificial Intelligence for Organic Chemistry: The Dendral Project*. McGraw-Hill, New York, 1980.
- [17] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.
- [18] A. Dress. Regular polytopes and equivariant tessellations from a combinatorial point of view. In *Algebraic Topology*, pages 56–72. SLN 1172, Göttingen, 1984.
- [19] A. Dress. On the classification of local disorder in globally regular spatial patterns. In L. Rensing and N. Jaeger, editors, *Temporal Order, Synergetics* 29, pages 61–66. Springer Verlag, 1985.
- [20] H. Hiller. The crystallographic restriction in higher dimensions. *Acta Cryst. Sect. A*, 41:p. 541 – 544, 1985.
- [21] H. Kroto, J. Heath, S. O’Brien, R. Curl, and R. Smalley. C_{60} : Buckminsterfullerene. *Nature*, 318:162–163, 1985.

- [22] P. Pleasants. Quasicrystallography: some interesting new patterns. In H. Iwaniec, editor, *Elementary and Analytic Theory of Numbers*, volume 17, pages p. 439 – 461. Banach Center Publ., Warsaw, 1985.
- [23] A. Dress. The 37 combinatorial types of regular “heaven and hell” patterns in the euclidean plane. In H. C. et al., editor, *M.C. Escher: Art and Science*, pages 35–43. Elsevier Science Publishers B.V., North-Holland, 1986.
- [24] A. Dress and R. Scharlau. The 37 combinatorial types of minimal non-transitive, equivariant tilings of the euclidean plane. *Discrete Mathematics*, 60:121–138, 1986.
- [25] A. Dress and D. Huson. On tilings of the plane. *Geometriae Dedicata*, 24:295–310, 1987.
- [26] D. Huson. Patches, stripes and net-like tilings, 1989. Bielefeld.
- [27] L. Balke. Diskontinuierliche gruppen als automorphismengruppen von pflasterungen. Master’s thesis, Universität Bonn, 1990. Bonner Mahematische Schriften Nr. 211.
- [28] G. Brinkmann. *Zur mathematischen Behandlung gestörter periodischer Pflasterungen*. PhD thesis, Bielefeld, 1990.
- [29] B. D. McKay. nauty User’s Guide (version 1.5). Technical Report TR-CS-90-02, Australian National University, Department of Computer Science, 1990. <http://cs.anu.edu.au/~bdm/nauty>.
- [30] R. L. Roth. Some 2-isohedral tilings of the plane. *Geometriae Dedicata*, 39(1):43 – 54, 1991.
- [31] O. Delgado-Friedrichs, D. Huson, and E. Zamorzaeva. The classification of 2-isohedral tilings of the plane. *Geometriae Dedicata*, 42:43–117, 1992.
- [32] R. Grund, A. Kerber, and R. Laue. MOLGEN – ein Computeralgebrasystem für die Konstruktion molekularer Graphen. *MATCH Commun. Math. Comput. Chem.*, 27:87–131, 1992.

- [33] D. Huson. A four-color theorem for periodic tilings. to appear in: *Geometriae Dedicata*, 1993.
- [34] D. Huson. Tile-transitive partial tilings of the plane. *Contributions to Geometry and Algebra*, 34(1):87–118, 1993.
- [35] G. Brinkmann. Fast generation of cubic graphs. *Journal of Graph Theory*, 23(2):139–149, 1996.
- [36] A. Dress and G. Brinkmann. Phantasmagorical fulleroids. *MATCH Commun. Math. Comput. Chem.*, (33):87–100, 1996. P.W. Fowler ed.: *Mathematical aspects of the fullerenes*.
- [37] M. Dresselhaus, G. Dresselhaus, and P. Eklund. *Science of Fullerenes and Carbon Nanotubes*. Academic Press, 1996. ISBN 0-12-221820-5.
- [38] S. Friedland. Discrete groups of unitary isometries and balls in hyperbolic manifolds. *Linear Algebra Appl.*, 241/243:p. 305 – 341, 1996.
- [39] L. Balke. Classification of disordered tilings. *Annals of Combinatorics*, 1:297–311, 1997. 10.1007/BF02558482.
- [40] G. Brinkmann and A. Dress. A constructive enumeration of fullerenes. *Journal of Algorithms*, 23:345–358, 1997.
- [41] G. Brinkmann and A. Dress. PentHex Puzzles – a reliable and efficient top-down approach to fullerene-structure enumeration. *Advances in Applied Mathematics*, 21:473–480, 1998.
- [42] G. Brinkmann and E. Steffen. Chromatic index critical graphs of orders 11 and 12. *Europ. J. Comb.*, 19:889–900, 1998.
- [43] B. D. McKay. Isomorph-free exhaustive generation. *Journal of Algorithms*, 26:306–324, 1998.

- [44] G. Brinkmann. Isomorphism rejection in structure generation programs. In P. Hansen, P. Fowler, and M. Zheng, editors, *Discrete Mathematical Chemistry*, volume 51 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 25–38. American Mathematical Society, 2000.
- [45] G. Brinkmann, U. von Nathusius, and A. Palser. A constructive enumeration of nanotube caps. *Discrete Applied Mathematics*, 116:55–71, 2002.
- [46] D. Klein. Topo-combinatoric categorization of quasi-local graphitic defects. *Phys. Chem. Chem. Phys.*, 4:2099–2110, 2002.
- [47] J. Bornhöft, G. Brinkmann, and J. Greinus. Pentagon-hexagon-patches with short boundaries. *European Journal of Combinatorics*, 24:517–529, 2003.
- [48] O. Delgado-Friedrichs. Data structures and algorithms for tilings i. *Theor. Comput. Sci.*, 303:431–445, July 2003.
- [49] A. Tripathi and S. Vijay. A note on a theorem of Erdős & Gallai. *Discrete Mathematics*, 265:417 – 420, 2003.
- [50] O. Delgado-Friedrichs and M. O’Keeffe. Isohedral simple tilings: binodal and by tiles with ≤ 16 faces. *Acta Crystallographica A*, 61:358–362, 2005.
- [51] R. Diestel. *Graph Theory 3rd edition*. Springer, 2005.
- [52] D. Klein and A. Balaban. The eight classes of positive-curvature graphitic nanocones. *Journal of Chemical Information and Modeling*, 46(1):307–320, 2006.
- [53] C. Justus. *Boundaries of triangle-patches and the expander constant of fullerenes*. PhD thesis, Universität Bielefeld, 2007.
- [54] G. Brinkmann and B. Coppens. An efficient algorithm for the generation of planar polycyclic hydrocarbons with a given boundary. *MATCH Commun. Math. Comput. Chem.*, 62(1):209–220, 2009.
- [55] G. Brinkmann. Generating water clusters and other directed graphs. *J. Math. Chem.*, 46:1112–1121, 2009.

- [56] G. Brinkmann, O. Delgado-Friedrichs, E. C. Kirby, and N. Van Cleemput. A Catalogue of Periodic Fully-Resonant Azulene-Transitive Azulenoid Tilings Analogous to Clar Structures. *CROATICA CHEMICA ACTA*, 82(4):781–789, DEC 2009.
- [57] G. Brinkmann, O. Delgado-Friedrichs, S. Liskens, A. Peeters, and N. Van Cleemput. CaGe – a virtual environment for studying some special classes of plane graphs – an update. *MATCH Commun. Math. Comput. Chem.*, 63:533–552, 2010.
- [58] G. Brinkmann, J. Goedgebeur, and B. D. McKay. Generation of cubic graphs. *Discrete Mathematics and Theoretical Computer Science*, 13(2):69–80, 2011.
- [59] G. Brinkmann, N. Van Cleemput, and T. Pisanski. Generation of various classes of trivalent graphs. *Theoretical Computer Science*, 2012.
- [60] G. Brinkmann. Generating regular directed graphs. Submitted.
- [61] Homepages of CaGe. <http://www.mathematik.uni-bielefeld.de/CaGe/>, <http://caagt.ugent.be/CaGe/>
- [62] O. Delgado-Friedrichs. Gavrog. <http://www.gavrog.org>
- [63] Azul website. <http://caagt.ugent.be/azul>
- [64] Pregraphs website. <http://caagt.ugent.be/pregraphs>
- [65] O. Delgado-Friedrichs. Personal communications.
- [66] W. Stein et al. *Sage Mathematics Software (Version 4.7.1)*. The Sage Development Team, 2011. <http://www.sagemath.org>.

Nederlandse samenvatting

Met structuurgeneratie bedoelen we het opstellen, implementeren (en uitvoeren) van een algoritme om objecten uit een bepaalde klasse te construeren. Structuurgeneratie heeft verschillende toepassingen binnen de theoretische chemie en de wiskunde. In de theoretische chemie vormt het een belangrijk gereedschap om structuren te voorspellen en hypothesen te testen. In de wiskunde wordt het gebruikt bij de classificatie van structuren en bij het zoeken naar tegenvoorbeelden voor vermoedens. We maken een onderscheid tussen random generatiealgoritmen en exhaustieve generatiealgoritmen. De eerste groep heeft als doel een willekeurige, gelijk verdeelde deelverzameling van alle structuren binnen een bepaalde klasse te genereren. De tweede groep heeft als doel een bepaalde klasse volledig te genereren. In deze thesis hebben we het enkel over exhaustieve generatiealgoritmen. Meer specifiek gaat het hier steeds om isomorfvrije, exhaustieve generatiealgoritmen. Dit wil zeggen dat we niet alleen alle structuren uit een bepaalde klasse genereren, maar dat we ook garanderen dat slechts één structuur per isomorfieklasse wordt gegenereerd.

In het **eerste hoofdstuk** introduceren we de concepten die in de rest van de thesis gebruikt worden. De twee voornaamste concepten zijn graaf en betegeling.

Een graaf $G(V, E)$ is een structuur die bestaat uit twee verzamelingen. De eerste verzameling V noemen we de toppen van de graaf. De tweede verzameling E bestaat uit deelverzamelingen van grootte 2 van V en noemen we de bogen van de graaf. De klassieke manier om een graaf voor te stellen is om de toppen als punten te tekenen en de bogen als lijnen die deze punten verbinden. Een graaf is kubisch als elke top incident is met exact 3 bogen.

Een betegeling is een opdeling van het vlak in tegels. Wij beschouwen enkel betegelingen waarbij alle tegels eindig zijn en waarbij elke eindige omgeving een eindig aantal tegels bevat. De plaatsen waar meer dan twee tegels elkaar ontmoeten, noemen we de toppen van de betegeling en de plaatsen waar exact twee tegels elkaar ontmoeten, noemen we de bogen van de betegeling.

We spreken van een periodieke betegeling als de symmetriegroep van de betegeling twee onafhankelijke verschuivingen bevat.

Een equivariante betegeling is een paar (T, G) waarbij T een betegeling is en G de symmetriegroep van de betegeling.

We introduceren in dit hoofdstuk ook Delaney-Dress symbolen [25]. Deze structuren bestaan uit een kubische multigraaf met eventueel semi-bogen (i.e. bogen met slechts één top) waarvan de bogen met drie kleuren zijn gekleurd, en twee functies die de toppen afbeelden op natuurlijke getallen. Deze Delaney-Dress symbolen encoderen equivariante, periodieke betegelingen en een Delaney-Dress symbool vormt dus een combinatorische, eindige voorstelling van een equivariante, periodieke betegeling. De graaf in een Delaney-Dress symbool noemen we een Delaney-Dress graaf.

De rest van deze thesis is opgedeeld in twee delen. Het eerste deel handelt over structuurgeneratie in de wiskunde.

In het **tweede hoofdstuk** stellen we een generatiealgoritme voor voor veralgemeende, kubische grafen. Kubische grafen vormen een goedbestudeerde klasse van grafen binnen de grafentheorie en er bestaan verschillende efficiënte programma's voor de generatie van kubische, simpele grafen. We veralgemenen het begrip graaf door ook multibogen, semi-bogen en lussen toe te laten en ook elke combinatie van deze. De verzamelnaam voor al deze klassen van grafen is pregrafen.

We splitsen het probleem op door het eerst te herleiden naar de generatie van kubische pregraafprimitieven. Dit zijn multigrafen met graden 1 en 3. We beschrijven een generatiealgoritme, gebaseerd op McKay's 'canonical construction path'-methode [43], dat op basis van kubische, simpele grafen deze kubische pregraafprimitieven genereert. Daarna gebruiken we het homomorfismeprincipe [32] om de kubische pregrafen te genereren op basis van de kubische pregraafprimitieven.

We besluiten dit hoofdstuk met enkele wijzigingen aan het generatiealgoritme die toelaten om op een efficiënte wijze de gegenereerde structuren te beperken tot enkel degene die 3-boog-kleurbaar zijn of tot enkel degene die bipartiet zijn. Ten slotte vermelden we ook een algoritme waarmee in lineaire tijd bepaald kan worden of een gegeven kubische pregraaf een 2-factor heeft waarbij elke component een quotiënt van C_4 is. Deze klasse is bijzonder interessant omdat ze de onderliggende, ongekleurde grafen vormen voor de Delaney-Dress grafen.

In het **derde hoofdstuk** grijpen we terug naar het filterprogramma op het einde van het vorige hoofdstuk. De analyse van de structuur van de kubische pregrafen met een 2-factor waarbij elke component een quotiënt van C_4 is, wordt gebruikt om deze

grafen in blokken op te delen. Vervolgens beschrijven we een generatiealgoritme dat via deze blokken rechtstreeks deze klasse van grafen kan genereren. De snelheidswinst ten opzichte van de filtertechniek uit het vorige hoofdstuk is dusdanig dat het in dit geval ook haalbaar wordt om Delaney-Dress grafen te genereren. Ten slotte nemen we ook de laatste stap en beschrijven hoe we op basis van deze resultaten Delaney-Dress symbolen en dus equivariante betegelingen kunnen genereren.

Na het derde hoofdstuk begint het tweede deel van deze thesis dat handelt over toepassingen in de chemie.

In het **vierde hoofdstuk** beschrijven we een generatiealgoritme voor een specifieke klasse van ‘grafietachtige’ netwerken. Azuleen ($C_{10}H_8$) is een isomeer van naftaleen. Het bestaat uit een zeven-ring en een vijf-ring die twee atomen en een binding delen. Dit onderzoek werd gestart naar aanleiding van een vraag van Edward Kirby. Hij was geïnteresseerd in welke koolstofnetwerken er gevormd konden worden waarbij er een partitie van de atomen in azulenen mogelijk is. Deze structuren worden azulenoiden genoemd. We modelleren dit aan de hand van betegelingen en focussen ons op deze netwerken waar er maar één baan van azulenen is onder de symmetriegroep van de betegeling. We beschrijven in dit hoofdstuk uitvoerig het algoritme dat ontwikkeld werd om alle Delaney-Dress symbolen te genereren die betegelingen encoderen die corresponderen met zo’n azulenoiden. Daarnaast beschrijven we ook hoe we achteraf deze Delaney-Dress symbolen visualiseren op een voor chemici bruikbare manier.

Een 1,5-patch is een eindige, bruggeloze, vlakke graaf met drie soorten vlakken: één “buitenvlak” met een onbeperkte grootte, 1 tot 5 vijfhoeken en een onbeperkt aantal zeshoeken. Daarnaast heeft elke top graad drie, behalve enkele toppen van het buitenvlak die graad twee hebben. Als we het aantal toppen van het buitenvlak met graad twee, respectievelijk graad drie, aanduiden met b_2 , respectievelijk b_3 , dan weten we door de Eulerformule dat $b_2 - b_3 = 6 - p$ met p het aantal vijfhoeken in de patch. Als het buitenvlak geen opeenvolgende toppen van graad drie bevat, dan noemen we de patch pseudoconvex. In het **vijfde hoofdstuk** beschrijven we een algoritme om alle pseudoconvexe patches met een gegeven randsequentie (i.e. een opeenvolging van de graden van de toppen van het buitenvlak) te genereren. Dit al-

goritme is gebaseerd op de encoding van pseudoconvexe patches in [40]. In plaats van de pseudoconvexe patch rechtstreeks op te bouwen, werken we met operaties die werken op de randsequentie en een zogehete buitenspiraal (een opeenvolging van de vlakken als ze spiraalsgewijs overlopen worden vanaf een speciale boog in het buitenvlak). Deze hebben als voordeel dat het relatief korte sequenties van getallen blijven, terwijl de pseudoconvexe patch een zeer grote graaf kan zijn. Hierdoor verkleinen we de overhead tijdens het genereren enorm.

In het **laatste hoofdstuk** richten we onze aandacht op nanocones ('nanokegels'). Een nanocone is een koolstofnetwerk dat conceptueel te situeren valt tussen enerzijds grafiet en anderzijds de halfopen, oneindige nanotubes ('nanobuisjes'): naast zeshoeken bevat deze structuur tussen één en vijf vijfhoeken, zodat noch de 'platte' vorm van grafiet, noch de buis met constante diameter van een nanotube gevormd kan worden.

Wij stellen een nanocone voor als een verstoring van een betegeling, meer bepaald een verstoring van een zeshoekig rooster waarbij alle verstoringen vlakken zijn die vijfhoeken zijn. We gebruiken een resultaat van Balke [39] dat zegt dat een verstoring van een equivariante betegeling volledig bepaald is door de oorspronkelijke, equivariante betegeling, een windingsgetal en een nevenklasse van een automorfisme in de symmetriegroep van de betegeling. Op basis hiervan tonen we aan dat er 8 (oneindige) klassen van nanocones zijn. Daarna stellen we een verdere classificatie op waarbij we elke klasse opdelen in een oneindig aantal eindige klassen die ook de localisatie van de vijfhoeken in rekening brengen. Ten slotte beschrijven we hoe het algoritme voor de generatie van pseudoconvexe patches kan gebruikt worden om een voorstelling van elke nanocone in zo'n klasse te genereren.

List of Figures

1.1	Three tilings with the same Delaney-Dress graph.	15
1.2	The square tiling (left), the barycentric subdivision of the square tiling (center) and the Delaney-Dress symbol of the square tiling (right). . .	20
1.3	The barycentric subdivision of the rectangular tiling (left) and the Delaney-Dress symbol of the rectangular tiling (right).	20
1.4	The bathroom tiling (left) and its barycentric subdivision (right). . . .	21
1.5	Representatives of the three orbits of chambers for the bathroom tiling.	22
1.6	The Delaney-Dress symbol of the bathroom tiling from Figure 1.4.a .	22
1.7	Two tilings that are topologically equivalent, but not equivariantly equivalent.	23
1.8	The two possible $\sigma_0\sigma_1$ orbits corresponding to a quadrangle(a) and a hexagon (b).	31
1.9	The Delaney-Dress symbols of the only two tile-transitive tilings containing only translation symmetry.	32
2.1	The relations between the different graph classes generated by pregraphs.	44
2.2	The relations between the different pregraph primitive classes.	46
2.3	The construction operation O_1 and the corresponding reduction. . . .	47
2.4	The construction operation O_2 and the corresponding reduction. . . .	47
2.5	The construction operation O_3 and the corresponding reduction. . . .	48
2.6	The construction operation O_4 and the corresponding reduction. . . .	49
2.7	The theta graph and the buoy graph.	50

2.8	The graphs G_1 and G_2 are obtained by applying operation O_3 to G .	54
2.9	The graphs G_1 and G_2 are obtained by applying operation O_4 to G .	55
2.10	The graphs G_1 and G_2 are obtained by applying operation O_1 to G .	57
2.11	The graphs G_1 and G_2 are obtained by applying operation O_2 to G .	58
2.12	The quotients of C_4 .	65
2.13	A prism and a Möbius ladder.	65
2.14	Possible end points of ladders.	66
2.15	Possible types of paths with semi-edges.	66
2.16	Each occurrence of a loop can be transformed into a double edge and vice versa.	71
2.17	The balloon graph.	71
3.1	The three C_4^q -markable pregraphs on two vertices.	88
3.2	The two C_4^q -markable pregraphs on two vertices that have non-isomorphic C_4^q -factors.	88
3.3	All C_4^q -marked pregraphs on one and three vertices.	89
3.4	The three isomorphic C_4^q -factors in a Möbius ladder on four vertices.	89
3.5	The two isomorphic C_4^q -factors in a C_4^q -markable Möbius ladder on more than four vertices.	90
3.6	The two isomorphic C_4^q -factors in a C_4^q -markable prism on more than eight vertices.	91
3.7	A ladder bounded by two digons has two different C_4^q -factors when its order is divisible by 4 and two isomorphic C_4^q -factors otherwise.	92
3.8	A ladder bounded by four semi-edges has two different C_4^q -factors when its order is divisible by 4 and two isomorphic C_4^q -factors otherwise.	93
3.9	A ladder bounded by a digon and two semi-edges has two different C_4^q -factors.	94

3.10	A path where each vertex is incident to at least one semi-edge and the two end vertices are incident to two semi-edges has two different C_4^q -factors when its order is even and two isomorphic C_4^q -factors otherwise.	95
3.11	The possible blocks containing quotients of type q_1 that can occur in a block partition when the partition contains more than one block.	97
3.12	The possible blocks containing quotients of type q_1 that can occur in a block partition when the partition contains one block.	98
3.13	The possible blocks containing quotients of type q_2 that can occur in a block partition when the partition contains more than one block.	99
3.14	The possible blocks containing quotients of type q_2 that can occur in a block partition when the partition contains one block.	99
3.15	The possible blocks containing quotients of type q_3 that can occur in a block partition when the partition contains more than one block.	100
3.16	The possible blocks containing quotients of type q_3 that can occur in a block partition when the partition contains one block.	100
3.17	The only possible block containing a quotient of type q_4 that can occur in a block partition when the partition contains more than one block.	101
3.18	The only possible block containing a quotient of type q_4 that can occur in a block partition when the partition contains one block.	101
3.19	A partial C_4^q -marked pregraph that is not closed.	105
3.20	Two extensions of the partial C_4^q -marked pregraph in Figure 3.19 which are isomorphic but for which the isomorphism does not induce an automorphism of the original partial C_4^q -marked pregraph.	106
4.1	The structures of naphthalene (a) and azulene (b).	140
4.2	The azulene graph.	141
4.3	An azulene has 8 outgoing bonds.	142
4.4	The five possible $\sigma_0\sigma_2$ -orbit s.	144
4.5	The four possible $\sigma_1\sigma_2$ -orbit s.	145

4.6	The first partial Delaney-Dress graph \mathcal{D} after connecting all the vertex components to the octagon component.	146
4.7	The partial Delaney-Dress graph \mathcal{D} at the beginning of the algorithm.	147
4.8	The bathroom tiling belongs to O_T and contains two different sets of octagons that satisfy the property that each vertex is contained in exactly one of the octagons in the set.	152
4.9	Replacing an octagon with an azulene in the partial Delaney-Dress symbol.	153
4.10	Some tilings with two isomorphic azulenic sets.	157
4.11	Some tilings with two nonisomorphic azulenic sets.	158
4.12	Marked and unmarked tilings do not necessarily have the same minimal symbol.	159
4.13	The Delaney-Dress graphs for the marked tilings in Figure 4.12.	160
4.14	The Delaney-Dress graphs for the minimal unmarked tiling in Figure 4.12.	161
4.15	An unmarked azulene-transitive azulenoid graph (4.15.a) and the 4 marked azulene-transitive azulenoid graph (4.15.b - 4.15.e) that have this tiling as underlying unmarked tiling.	162
4.16	The local chamber system of an octagon (a) and a vertex of degree three (b). For each chamber d the neighbouring chamber at the solid edge is $\sigma_0(d)$, at the dotted edge is $\sigma_1(d)$ and at the dashed edge is $\sigma_2(d)$	163
4.17	The chamber system of our start point. For each chamber d the neighbouring chamber at the solid edge is $\sigma_0(d)$, at the dotted edge is $\sigma_1(d)$ and at the dashed edge is $\sigma_2(d)$	163
4.18	Replacing an octagon with an azulene in the Delaney-Dress symbol.	164
4.19	Fundamental domain	165
4.20	The grid of fundamental domains	166
4.21	Some examples of edges in a periodic graph.	167
4.22	An edge and its inverse in a periodic graph.	167
4.23	Rescaling of the fundamental domain.	168

4.24	Shearing of the fundamental domain.	168
4.25	The two types of boundaries for fundamental patches.	171
4.26	A schematic view of how hexagonal fundamental patches are positioned in a quadrangular grid.	172
4.27	An example of a tiling with a hexagonal fundamental patch that has been translated to a periodic graph.	173
4.28	A torus with two topologically different fundamental cycles.	176
4.29	A torus cut open and shown as a rectangle.	176
4.30	A small tile and the 4×4 large tile corresponding to this tile.	177
4.31	The six structures where all the azulenes are equivalent under a group of translations.	178
5.1	A patch with boundary $2(23)^22(23)^32(23)^4$	181
5.2	The three possible outer spirals in the pseudo-convex patch from Figure 5.1.	182
5.3	The non-degenerate case of the operation \mathcal{H}	187
5.4	The non-degenerate case of the operation $\mathcal{P}(i)$ with $0 \leq i < s_1$	188
5.5	The non-degenerate case of the operation $\mathcal{P}(s_1)$	188
5.6	The operation \mathcal{H} in case the length of the boundary sequence is 1	189
5.7	The operation $\mathcal{P}(i)$ in case the length of the boundary sequence is 1	190
5.8	The operation $\mathcal{P}(s_1 - 1)$ in case the length of the boundary sequence is 1	191
5.9	The operation \mathcal{H} in case the length of the boundary sequence is 2	192
5.10	The operation $\mathcal{P}(i)$ in case the length of the boundary sequence is 2	193
5.11	The operation $\mathcal{P}(s_1)$ in case the length of the boundary sequence is 2	193
5.12	The operation \mathcal{H} in the general case of the second class of degenerate cases with $s_2 = 0$	197
5.13	The operation \mathcal{H} in the general case of the second class of degenerate cases with $s_l = 0$	198
5.14	The operation $\mathcal{P}(i)$ in the general case of the second class of degenerate cases with $s_l = 0$	198

5.15	The operation $\mathcal{P}(s_1)$ in the general case of the second class of degenerate cases with $s_l = 0$	199
5.16	The operation \mathcal{H} in the general case of the second class of degenerate cases with $s_2 = s_l = 0$	199
5.17	The operation \mathcal{H} in the case length = 2 of the second class of degenerate cases with $s_2 = 0$	201
5.18	The operation $\mathcal{P}(i)$ in the case length = 2 of the second class of degenerate cases with $s_2 = 0$	202
5.19	The operation \mathcal{H} in the case length = 2 of the second class of degenerate cases with $s_2 = 1$	202
5.20	The operation \mathcal{H} in the case length = 3 of the second class of degenerate cases with $s_2 = 0$	203
5.21	The operation \mathcal{H} in the case length = 3 of the second class of degenerate cases with $s_3 = 0$	204
5.22	The operation $\mathcal{P}(s_1)$ in the case length = 3 of the second class of degenerate cases with $s_3 = 0$	205
5.23	The operation \mathcal{H} in the case length = 4 of the second class of degenerate cases with $s_2 = s_4 = 0$	206
6.1	Three types of carbon networks	212
6.2	Two views of a patch with two pentagons.	213
6.3	There are no consecutive break-edges in a layer with more than three faces.	216
6.4	The boundary of a cone patch with 3 pentagons and a layer of hexagons. 217	
6.5	A pseudo-convex patch with boundary $2(23)^0(2(23)^1)^3$ cannot have two hexagons in the boundary.	220
6.6	The fixed point of the combination of two counterclockwise rotations of 120° around the centre of a face is always the centre of a face. . .	225
6.7	Vectors in the hexagonal grid.	228
6.8	Vectors in the hexagonal grid.	229
6.9	Vectors in the hexagonal grid.	229

6.10	The vector corresponding to the boundary 2322322323223.	230
6.11	The three possible directions in a shortest facial path when arriving in a hexagon.	242
6.12	A sequence containing only straight steps and two left steps can al- ways be rewritten as a shorter sequence.	243
6.13	A sequence containing only straight steps and two left steps can al- ways be rewritten as a shorter sequence.	245
6.14	Two sequences with the same length.	246
6.15	A chain of hexagons bounded by a pentagon on each side.	247
6.16	A quadrangular patch formed by two pentagons and the enclosing shortest paths with only one turn each.	247
6.17	Possible situations when a neighbouring hexagon has a disconnected boundary with a pseudo-convex patch P	252
6.18	Possible situations when two neighbouring hexagons share an edge that has an empty intersection with the pseudo-convex patch P	256
6.19	A symmetric cone patch with two pentagons.	263
6.20	Two near-symmetric patches.	265
6.21	The unique nanocone with one pentagon with a near-symmetric- like patch in grey. This patch has the canonical boundary sequence $2(23)^0(2(23)^1)^4$	266
6.22	The prototypes for cone patches with 5 pentagons and the maximum number of hexagons	292
B.1	The block list that corresponds to the block list file containing 4 3 0 1 1 4 2 0.	318
B.2	The block list that corresponds to the block list file containing 8 2 0 1 2 3 0 1 2 0.	318

List of Tables

- 2.1 The number of structures for each class for a given number of vertices n 72
- 2.2 The timings for the numbers in Table 2.1 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. 73
- 2.3 The average number of structures per second for the numbers in Table 2.1 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. 73
- 2.4 The number of 3-edge-colourable structures for each class for a given number of vertices n 74
- 2.5 The timings for the numbers in Table 2.4 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. 75
- 2.6 The average number of structures per second for the numbers in Table 2.4 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. 75
- 2.7 The number of non-3-edge-colourable structures for each class for a given number of vertices n 76
- 2.8 The number of structures for each class that have a C_4^q -factor and those that have a C_4 -factor for a given number of vertices n 77
- 2.9 The timings for the numbers in Table 2.8 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. 78
- 2.10 The average number of structures per second for the numbers in Table 2.8 using the program `pregraphs` and run on a 2.40 GHz Intel Xeon. 78

2.11	The number of bipartite graphs for a given number of vertices n and the ratio of bipartite graphs versus the total number of graphs.	79
2.12	The timings for the numbers in Table 2.11 using the program <code>pregraphs</code> and run on a 2.40 GHz Intel Xeon.	80
2.13	The average number of structures per second for the numbers in Table 2.11 using the program <code>pregraphs</code> and run on a 2.40 GHz Intel Xeon.	80
2.14	The number of bipartite graphs that allow a 2-factor where each component is a quotient of C_4 for a given number of vertices n and the ratio of bipartite pregraphs with such a 2-factor versus the total number of bipartite pregraphs.. . . .	81
2.15	The timings for the numbers in Table 2.14 using the program <code>pregraphs</code> and run on a 2.40 GHz Intel Xeon.	82
2.16	The average number of structures per second for the numbers in Table 2.14 using the program <code>pregraphs</code> and run on a 2.40 GHz Intel Xeon.	82
2.17	The number of bipartite graphs that allow a 2-factor where each component is a C_4 for a given number of vertices n and the ratio of bipartite pregraphs with such a 2-factor versus the total number of bipartite pregraphs.	83
2.18	The timings for the numbers in Table 2.17 using the program <code>pregraphs</code> and run on a 2.40 GHz Intel Xeon.	83
2.19	The average number of structures per second for the numbers in Table 2.17 using the program <code>pregraphs</code> and run on a 2.40 GHz Intel Xeon.	83
3.1	Comparison of the number of 3-edge-colourable pregraphs on n vertices and the number of pregraphs with a C_4^q -factor on n vertices. . .	87
3.2	The possible blocks containing quotients of type q_1 that can occur in a block partition.	98

3.3	The possible blocks containing quotients of type q_2 that can occur in a block partition.	99
3.4	The possible blocks containing quotients of type q_3 that can occur in a block partition.	100
3.5	The possible blocks containing a quotient of type q_4 that can occur in a block partition.	101
3.6	The properties of tilings which we want to be able to specify and the corresponding default values.	117
3.7	An overview of the inequality 3.4 calculated for several different combinations of minima.	119
3.8	An overview of the inequality 3.5 calculated for several different combinations of maxima.	119
3.9	Some properties of lists of blocks which are useful to reject such a list.	124
3.10	The number of Delaney-Dress graphs that appear in Delaney-Dress symbols of a tiling of the Euclidean plane and those that do not appear in such symbols.	126
3.11	The percentage of Delaney-Dress graphs that satisfy the conditions 3.9, 3.10 and/or 3.11 and thus can be rejected. On up to 8 vertices, no Delaney-Dress graph satisfies any of these conditions.	128
3.12	An overview of the number of block lists, the number of C_4^q -marked pregraphs and the number of C_4^q -markable pregraphs with n vertices.	129
3.13	An overview of the number of Delaney-Dress graphs and the time needed by <code>ddgraphs</code> to generate these graphs when run on a 2.40 GHz Intel Xeon.	132
3.14	An overview of the number of Delaney-Dress symbols of the Euclidean plane and the time needed by <code>ddgraphs</code> to generate these symbols when run on a 2.40 GHz Intel Xeon.	133
3.15	The refinement on the parameters as calculated by <code>ddgraphs</code> in order to generate the tile-transitive tilings of the plane	134
3.16	The refinement on the parameters as calculated by <code>ddgraphs</code> in order to generate the tile-2-transitive tilings of the plane	134

4.1	The twelve sets of eight integers that are valid candidates for the values of m_{01} in the Delaney-Dress graph.	150
4.2	Changes to σ_i for the partial Delaney-Dress symbol in Figure 4.9 when replacing an octagon by an azulene.	154
4.3	Adjusting m_{01} for the azulene	155
4.4	The parameters for the edges in figure 4.21	166
5.1	An overview of the different cases for the operations to fill a pseudo-convex boundary.	186
6.1	The complete classification of cone patches.	223
6.2	The number of canonical cone patches for given minimal side length, given number of pentagons and given type of boundary.	267
6.3	The number of canonical IPR cone patches for given minimal side length, given number of pentagons and given type of boundary.	271
6.4	The number of canonical non-IPR cone patches for given minimal side length, given number of pentagons and given type of boundary.	274
6.5	The maximum number of hexagons in a cone patch for given minimal side length, given number of pentagons and given type of boundary.	277
6.6	The minimum number of hexagons in a cone patch for given minimal side length, given number of pentagons and given type of boundary.	280
6.7	The maximum number of vertices in a cone patch for given minimal side length, given number of pentagons and given type of boundary.	282
6.8	The minimum number of vertices in a cone patch for given minimal side length, given number of pentagons and given type of boundary.	285
6.9	The upper bounds on the number of hexagons in a pseudo-convex patch with p pentagons given in [47] expressed as functions of the length b of the boundary.	288
6.10	The upper bounds from Table 6.9 expressed as function of the length of the shortest side.	288
6.11	The new upper bounds for the number of hexagons in a cone patch with p pentagons.	291

A.1	The options available when running the program azul.	298
A.2	The general options available when running the program ddgraphs.	299
A.3	The options available to specify the type of the generated structures when running the program ddgraphs.	300
A.4	The options available to constraint the generated structures when running the program ddgraphs.	301
A.5	The options available when running the program pregraphs.	304

Index

The definition of entries are written in a bold type face.

- 1,5-patch, **180**
- A_T , **141**
- A_T° , **142**
- acceptable list of blocks, **104**
- adjacent, **2**
- $Aut(G)$, **3**
- automorphism, **3**
- azulene, xiv, **140**
- azulene graph, **140**
- azulene-transitive azulenoid tiling, **141**
- azulenic set, **140**
- azulenoid, **140**
- azulenoid tiling, **140**
- \mathcal{B}_L , **104**
- $B(P)$, **104**
- barycentric subdivision, **18**
- bipartite, **5**, 63–64
- block partition, **96**
- blocks, **96**
- boundary, **180**, **214**
- boundary sequence, **183**
- boundary signature, **218**
- boundary vector, **181**, **215**
- branched component, **25**
- branching size, **25**
- break-edge, **180**
- buoy graph, **48**
- canonical cone patch, **226**
- canonical family, **218**
- canonical marked edge, *see* marked edge
- canonical representative, **34**, **219**
- canonical representative function, **33**
- canonical vertex labelling, **33**
- canonically marked patch, **226**
- canonicity, **33**
- chamber, **18**
- chamber system, **19**
- chromatic index, **6**
- chromatic number, **5**
- closed, **105**
- combinatorial fullerene, **8**
- cone graph, **213**
- cone patch, *see* patch

- connected, **4**
 construction operation, **32**
 cover
 of a component, **25**
 of a symbol, **25**
 C_4^q -marked pregraph, **86**
 crown, **66**
 cubic pregraph primitive, **45**
 curvature, **16**
 cycle, **4**

 deficient vertex, **104**
 degree, **2**
 pregraph, **42**
 degree of a face
 tiling, **10**
 degree sequence, **102**
 Delaney-Dress graph, **14**, 110–111, 131
 Delaney-Dress symbol, **15**, 13–31, 111–
 127, 131
 $\mathcal{D}(T, G)$, **14**

 edge
 graph, **2**
 tiling, **10**
 edge colouring, **6**
 edge-colouring, 60–63
 equivalent
 equivariantly, **14**
 topologically, **13**
 equivariant tiling, **13**, 220–224
 equivariantly equivalent, *see* equivalent

 Euler's Formula, **7**
 extension, **105**
 extracted, **26**

 face
 plane graph, **7**
 tiling, **10**
 facial cycle, **11**
 facial path, **242**
 family of patches, **218**
 flag graph, **13**
 flagspace, **11**
 fullerene, **8**
 fundamental patch, **26**

 $\Gamma(\mathcal{D})_{ij}$, **18**
 graph
 simple, **2**
 with loops, **3, 43**
 with semi-edges, **43**
 graphic sequence, **102**

 heaven-and-hell tiling, **122**

 incident, **2**
 induced subgraph, **4**
 IPR, *see* Isolated Pentagon Rule
 Isolated Pentagon Rule, **9**
 isomorphism, **2**

 k -colourable, **5**
 k -partite, **5**

- ladder, **65**
- layer, **215**
- length
 - cycle, **4**
 - path, **4**
- loop, **3**
- loopless pregraph, **43**

- mark, **182**
- marked boundary, **182**
- marked break-edge, **182**
- marked cone patch, **226**
- marked edge
 - canonical, **226**
- marked patch, **182**
- minimal, **24**
- morphism, **24**
- multi-edge, **3**
- multigraph, **3, 43**
- multigraphic sequence, **102**
- multiset, **3**
 - \in , **3**
 - multiplicity, **3**

- nanocone, **212–213**
- near-symmetric boundary, **214**
- n th shell, **209**

- O_T , **142**
- orientable, **25**
- outer shell, **209**
- outer spiral, **182**

- P^* , **45**
- partial C_4^q -marked pregraph, **104**
- partition, **5**
 - discrete, **5**
 - size, **5**
- patch
 - cone, **214**
- path, **4**
- periodic graph, **165, 165–169**
- periodic tiling, **10, 175–177**
- planar, **6**
- plane graph, **7**
- pregraph, **42**
- primitivisation, **45**
- prism, **66**
- pseudo-convex, **180**

- q_1 , **65**
- q_2 , **65**
- q_3 , **65**
- q_4 , **65**
- quotient of C_4 , **65**

- reduction operation, **33**
- regular heaven-and-hell tiling, **122**

- search space, **32**
- semi-edge-free pregraph, **43**
- semi-regular heaven-and-hell tiling, **122**
- σ_i, σ_j -components, **13**
- simple graph, **43**
- simple pregraph, **43**

- single edge, **3**
- $\sigma_i\sigma_j$ -component, **18**
- skeleton graph, **10**
- strongly closed, **108**
- subgraph, **4**
 - marked, **104**
- successor, **183**
- symmetric boundary, **214**
- symmetry breaking, **23**

- theta graph, **48**
- tile-transitive tiling, **29**
- tiling
 - degree of a vertex, **10**
 - edge, *see* edge
 - face, *see* face
 - marked faces, **24**
 - vertex, *see* vertex
- topologically equivalent, *see* equivalent

- underlying graph, **3**

- vertex
 - graph, **2**
 - tiling, **10**