UNIVERSIDAD
SERGIO ARBOLEDA

Characterizing dual graphs associated to triangulations of the Caracol flow polytope

Francisco Alfonso Mayorga Cetina

Universidad Sergio Arboleda
School of exact sciences and engineering
Mathematics

Bogotá D.C. February of 2020

UNIVERSIDAD
SERGIO ARBOLEDA

Characterizing dual graphs associated to triangulations of the Caracol flow polytope

Francisco Alfonso Mayorga Cetina

Advisor:
Rafael Santiago González D'León

This thesis is presented as partial requirement for getting the degree of
Mathematician

Universidad Sergio Arboleda
School of exact sciences and engineering
Mathematics

Bogotá D.C. February of 2020

Abstract

Associated to a directed graph G and a sequence representing net flows at the vertices of G we can define a polytope whose points correspond to flows through the directed edges of G. The family of polytopes obtained in this way are called flow polytopes and are the subject of recent study. Stanley-Postnikov and Mézsáros-Morales-Striker proposed a procedure to construct different subdivisions of a flow polytope. For net-flow sequence $(1, 0, \ldots, 0)$ any such subdivision happens to be a triangulation. We study the dual graphs of such triangulations for a particular family of graphs known as Caracol graphs, previously studied by Benedetti et al., and whose associated flow polytopes have normalized volumes given by the Catalan numbers. We show that the dual graph of one of the triangulations happens to be the 1-skeleton of the simplicial associahedron and another one is a toggle graph that is obtained by flip operations on the set of Dyck paths.

Dedicated to the woman that I love, Alicia ♥.

Acknowledgments

There are a lot of people who have directly or indirectly influenced what I have achieved throughout my career. I wanted to give personal thanks to each of those who have made part of my journey, but it is difficult with so little space to write. That's why, as I know I can't mention them all, I apologize.

First, the person who deserves the greatest of thanks is my mother, the person for whom I have become who I am. She, with all her effort, dedication and affection, has inspired, supported and motivated us (me and my brothers) to build a future. Without that woman, I would not have achieved anything I have so far. Therefore, I can't find a way to give infinite thanks to mom. The other person for whom I came to study this madness (because mathematics is for crazy people) is my brother Carlos. That guy has inspired me even before entering college. Despite its simplicity, he is a genius of mathematics that makes me not able to be left behind, that I try harder to be a little closer to his level. I thank Carlos not only for his collaboration in my studies but for being that friend who always accompanied me, who told me his things and who understood my jokes; because he is the best, thanks for being my brother. I want also to thank my grandparents who were always on the lookout for me to be in the best way in my studies, who supported me when I had difficulties and who with their stubbornness always found a way to help.

Now, I have to thank the people who also took part in what was my undergraduate studies. I want to thank my professors, classmates, and administrators for being there to lend a hand when necessary. To each of these people for their small or large intervention in my professional development, for their teachings, for their company and everything else. In particular, I would like to thank Dr. Rafael González, that man and his patience made this work come to be. Thank you for explaining, again and again, *"dumb"* things that were not clear to me, thank you for having this disposition to resolve any concerns and support my growth in various ways, for lending me your knowledge and creativity to direct my work, for proposing the topic for this document because it is something that has filled me as a mathematician and that has made me happy to study.

Finally, and not least, I want to thank the woman with whom I could understand what love meant, what true patience and understanding meant. To that woman for whom I managed to finish writing this work, for which I promised to be a mathematician who inspired and complemented her. To the woman to whom this thesis is dedicated. Thank you very much, Alicia, thank you very much for being part of my career and my life.

To all of them, thank you very much. And also to everyone who takes the time and effort to read this work that was done with the greatest of affection.

Contents

List of figures

List of codes

1 Introduction

Let G be a directed graph with vertex set $[n+1] := \{1, 2, \ldots, n+1\}$ and a set of directed edges $E \subseteq [n+1] \times [n+1]$, and let $\overrightarrow{a} = (a_1, a_2, \ldots, a_n) \in \mathbb{Z}^n$ be an integer tuple. The *flow polytope* $\mathcal{F}_G(\overrightarrow{a})$ is the set of points in $\mathbb{R}^E$ that correspond to flows through the edges of G and where $(a_1, a_2, \ldots, a_n, -\sum_{i=1}^{n} a_i)$ represents the *net flow* on the vertices of G. This family of polytopes is the subject of study in the present work and have received a lot of attention in multiple research areas as optimization (see for example [19]), geometry (see [12]), algebra (see [14]) and combinatorics (see [7, 15]) because of its interesting properties. In particular, in combinatorics there has been relevant recent work of Baldoni and Vergne [2], Mészaros and Morales [16], Benedetti et al. [3] and unpublished work of Postnikov and Stanley, that has revitalize this area of study.

From the combinatorial and geometric point of view, one interesting problem is to compute the volume of $\mathcal{F}_G(\overrightarrow{a})$ for a suitable graph G and a net flow vector $\overrightarrow{a}$. An interesting case is when $G = K_{n+1}$ (the complete graph with edges (i, j) whenever $i < j$) and $\overrightarrow{a} = (1, 0, \ldots, 0)$. It was conjectured by Chan, Robbins and Yuen in [6] and proved by Zeilberger in[24] that the normalized volume

$$\mathrm{Vol}(\mathcal{F}_{K_{n+1}}(1, 0, \ldots, 0)) = C_1 \cdot C_2 \cdot \cdots \cdot C_{n-2}, \tag{1.1}$$

where C_n denotes the n-th Catalan number. Zeilberger proof is algebraic and is based on the constant term of a power series identity. The polytope $\mathcal{F}_{K_{n+1}}(1, 0, \ldots, 0)$ is known in the literature as the $\mathcal{CRY}_n$ polytope after the authors of the original conjecture. It is of current interest in the community to find a combinatorial explanation of the appearance of the product of Catalan numbers, since Catalan numbers have plenty of interesting combinatorial interpretations.

One possible approach to find a combinatorial interpretation of the volume of the $\mathcal{CRY}_n$ polytope is to consider families of simpler graphs that can shed some light on the underlying combinatorial mechanisms of the problem. One such graph that has received enough attention has been coined with the name of the Caracol graph Car_n by Benedetti et al. in [3]. Stanley (unpublished), Benedetti et al.[3] and Mészaros et al.[17] proved that

$$\mathrm{Vol}(\mathcal{F}_{Car_n}(1, 0, \ldots, 0)) = C_{n-2}, \tag{1.2}$$

the final term of the product formula (1.1) for the $\mathcal{CRY}_n$ polytope.
One technique to prove formulas like (1.1) and (1.2) is to use what is known as a *triangulation*, where the polytope is subdivided in smaller pieces, known as simplices, each of normalized volume equal to 1. Then we are just left with the problem of counting the number of simplices of the triangulation to be able to determine the volume of the whole polytope. Postnikov and Stanley introduced (unpublished) a family of triangulations for flow polytopes that were developed further and formalized in an article by Mészáros and Morales [16].
Since the family of Catalan objects is rich in combinatorial properties, formula (1.2) suggest that additional structure can be revealed on Catalan objects by studying the triangulations that give rise to such formula. In particular, given a triangulation of a polytope one can define an (undirected) graph whose vertices are the maximal simplices or *facets* in the triangulation and where there is an edge between any two facets that intersect maximally. This graph is called the *dual graph* of the triangulation.
In this work we study the subdivision mechanism described in [17] to build triangulations of the Caracol polytope $\mathcal{F}_{Car_n}(1,0,\ldots,0)$. We refer to these triangulations as totally reductive Postnikov-Stanley (*TRPS*) triangulations. They are obtained by making *reductions* at every vertex of a *framed* graph. The concept of a framed graph was defined by Danilov, Karzanov, and Koshevoy, in [8] as a graph together with a pair of linear orderings on the sets of incoming and outgoing edges at each vertex, which is called a *framing*. Based on the fact that different framings potentially give different triangulations and with the help of some computations in the free open-source mathematics software system *SageMath* (which provided us with examples on how these triangulations behave) we discovered relations between the dual graphs of two particular triangulations and known graphs related to Catalan objects.
An interesting family of combinatorial objects that encode geometric information are the *tubings* on a connected graph $G = (V, E)$ that were defined by Carr and Devadoss in [4]. These are collections of subsets $T \subset V$ such that the induced graph G_T is connected (also known as *tubes*) and satisfying a compatibility condition. It is known that the set of maximal tubings on a line graph L_n is a family of Catalan objects and that they are also the set of facets of a simplicial complex (known as the *nested set complex*) constructed out of the information enconded on tubings. The dual graph of this particular, nested set complex for the line graph is known to coincide with the 1-skeleton of a very famous polytope known as the *associahedron* or the *Stasheff polytope* (see [9, 10, 11, 18]). We prove then the following theorem.

Theorem (4.9). *The dual graph of the backward TRPS triangulation of the Caracol flow polytope $\mathcal{F}_{Car_n}(1,0,\ldots,0)$ given by the lex-revlex ordering as framing on the edges is the 1-skeleton of the $(n-3)$-associahedron.*

We prove Theorem 4.9 by finding a bijection between a set of paths on the Caracol graph Car_n (which correspond to vertices of $\mathcal{F}_{Car_n}(1,0,\ldots,0)$) and tubes of the line graph L_{n-2}. This bijection maps collections of paths satisfying a compatibility condition to tubings.
Another family of very popular Catalan objects is the set of Dyck paths D_n. *Dyck paths* are lattice paths from $(0,0)$ to (n,n) using $(0,1)$ (N or North) and $(1,0)$ (E or East) steps such that the path does not go below the line $y = x$. One can construct a graph that relates pairs of these paths whenever one is obtained from the other by making a valid switch of consecutive steps *NE* to *EN*. This graph is called the *Toggle graph* on the set of Dyck paths. We prove the following theorem.

Theorem (4.10). *The dual graph of the backward TRPS triangulation of the Caracol flow polytope $\mathcal{F}_{Car_n}(1,0,\ldots,0)$ given by the planar ordering as framing on the edges is the toggle graph $Togg_{n-2}$ on the set of Dyck paths D_{n-2}.*

The results in [17] imply that there is an integral equivalence between a flow polytope of a planar graph G (together with its planar embedding) and an order polytope on a poset P_G on vertices of the dual graph of G. Order polytopes are a family of polytopes associated to posets that were studied by Stanley in [21]. Stanley defined a triangulation called the *canonical* triangulation whose simplices correspond to linear extensions of the associated poset P. We prove Theorem 4.10 using a bijection between the set of linear extensions of the poset P_{Car_n} and the set of Dyck paths D_{n-2}.

This thesis is organized as follows: On Chapter 2 we provide some background and notions on graphs, polytopes, triangulations and Catalan combinatorics. On Chapter 3 we give the definition of a flow polytope and given some examples. We also describe the subdivision technique in [17] and describe an algorithm to obtain such a subdivision. We call this the *MMS* algorithm. On Chapter 4 we prove Theorems 4.9 and 4.10. On appendix A we provide the SageMath code used to visualize and conjecture the results presented in this work.

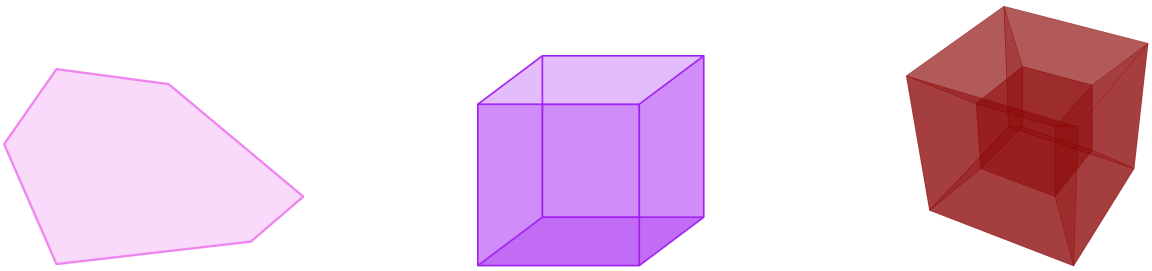
(...) and thus we have the anomaly of the most rigidly exact in science applied to the shadow and spirituality of the most intangible in speculation.

Edgar Allan Poe

2 Preliminaries

2.1 Polytopes and graphs

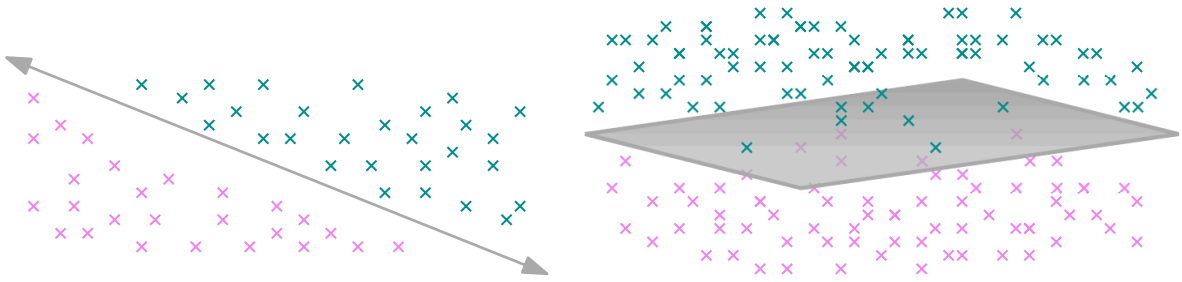
Most of us are familiar with the idea of a *polygon*. We can think of a polygon as a flat figure bounded by edges as Figure 2.1a. As we know from school, polygons live in the plane and that implies that their dimension is two. However, we can think about how these kind of bounded figures will look-like in more general spaces. In the three dimensional space we can take as an example the Platonic solids, that we know of from our geometry course at school. One of them is the cube that is illustrated in Figure 2.1b. Concepts and definitions in this section will follow closely the book of Ziegler [25] and we invite the reader to consult it for undefined terms.



(a) Hexagon. **(b)** Cube. **(c)** Hypercube.

Figure 2.1: Examples of polytopes.

We know from euclidean geometry that the n-dimensional space can be separated in two *halfspaces* by a *hyperplane*, that is an affine subspace of dimension $n - 1$. In a two dimensional space a hyperplane is a straight line and for three dimensions it is a plane (see Figure 2.2). Figures as the showed above can be described in different ways. The two more common descriptions are, referring to them either as the smallest *convex* region of space that contains a given set of points, or as the intersection of selected half spaces.



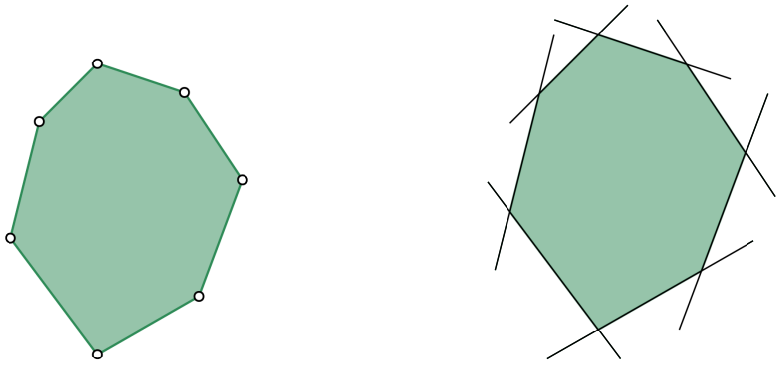
(a) A line separating a two dimensional space. **(b)** A plane separating a three dimensional space.

Figure 2.2: The corresponding hyperplanes for two and three dimensional spaces.

The spaces that we are referring to are the n dimensional spaces $\mathbb{R}^n$ which consists of all the tuples $(x_1, \ldots, x_n)$ where each $x_i \in \mathbb{R}$. These tuples are a way to represent a point in $\mathbb{R}^n$ and every x_i represents the *ith coordinate* of that point. An *affine subspace* of $\mathbb{R}^n$ is a subspace of $\mathbb{R}^n$ that has been shifted by a translation. The *affine hull* of a set of points P is the intersection of all affine subspaces that contain P. The *convex hull* of a set of points P is the smallest convex set containing P, that is, the intersection of all convex sets that contain P. Note that, in particular, the convex hull of P is a subset of its affine hull. If $P = \{\vec{a_1}, \ldots, \vec{a_k}\} \subseteq \mathbb{R}^d$, its convex hull can be expressed algebraically as

$$conv(P) = \left\{ \lambda_1 \vec{a_1} + \cdots + \lambda_k \vec{a_k} \;\middle|\; \sum_{i=1}^{k} \lambda_i = 1 \text{ and } \lambda_i \geq 0 \right\}.$$

With these facts in mind, we can give two different (but equivalent) definitions of a polytope.
An $\mathcal{H}$-polytope is an intersection of finitely many closed halfspaces in $\mathbb{R}^n$ that is bounded in the sense that it does not contain a ray $\{\vec{x} + t\vec{y} : t > 0\}$ for any $\vec{y} \neq 0$. A $\mathcal{V}$-polytope is the convex hull of a finite set of points in $\mathbb{R}^n$. In Figure 2.3 we illustrate the same polytope according to both definitions.



(a) $\mathcal{V}$-polytope. **(b)** $\mathcal{H}$-polytope.

Figure 2.3: Two ways of defining a polytope.

Theorem 2.1 (Minkowsky-Weyl's Theorem). *Called* main theorem for polytopes *in the book of Ziegler[25]. A subset $P \subseteq \mathbb{R}^d$ is the convex hull of a finite point set (a $\mathcal{V}$-polytope)*

$$P = conv(V) \text{ for some } V \in \mathbb{R}^{d \times n}$$

if and only if it is a bounded intersection of halfspaces (an $\mathcal{H}$-polytope)

$$P = P(A, z) \text{ for some } A \in \mathbb{R}^{m \times d}, z \in \mathbb{R}^m$$

According to Theorem 2.1 we can give the following definition.

Definition 1 (Polytope). *A* polytope *is a subset $P \subseteq \mathbb{R}^n$ which can be presented either as a $\mathcal{V}$-polytope or as an $\mathcal{H}$-polytope.*

The *(intrinsic) dimension* of a polytope is the dimension of its affine hull. A *d-polytope* is a polytope of dimension d in $\mathbb{R}^n$ with $d \leq n$, where n is said to be the dimension of the *ambient space*. We have then that 0-polytopes are points, 1-polytopes are line segments, 2-polytopes are polygons, 3-polytopes are three dimensional solids like the cube, 4-polytopes are figures like the *hypercube* in Figure 2.1c and so forth.

Remark. *If in the definition of an $\mathcal{H}$-polytope we remove the condition of being bounded, we obtain a more general family of objects known as* polyhedra.

A d-polytope P is said to be *integral* if all its vertices have integer coordinates. The *normalized volume* of an integral polytope is defined to be $d! \cdot V_P$, where V_P is its euclidean volume.

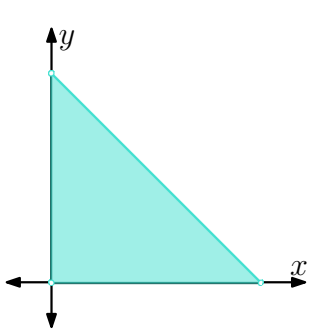
2.2 Triangulations of a polytope and the dual graph

In two dimensions the simplest 2-polytope we can think of is a triangle (see Figure 2.4a). A triangle is the convex hull of three points in general position (not colinear). If we move to three dimensions, the analogue of a triangle is the tetrahedron (see Figure 2.4b). Now we want to consider their d-dimensional generalizations. A *d-simplex* is a polytope of dimension d with $d + 1$ vertices.

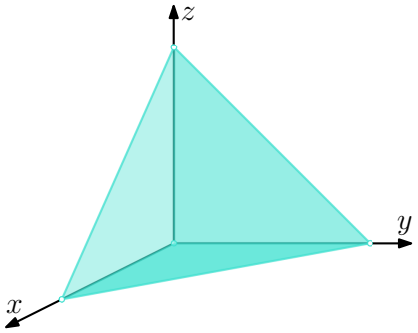
(a) Triangle. **(b)** Tetrahedron.

Figure 2.4: Simplices for two and three dimensions.

The *unitary* d-simplex is defined as $\Delta_d = \left\{(a_1, \ldots, a_d) \in \mathbb{R}^d \;\middle|\; \sum_{i=0}^{d} a_i \leq 1 \text{ and } a_i \geq 0\right\}$. We can see from this definition that $\Delta_d = conv(\{\vec{0}, e_1, \ldots, e_d\})$, that is, the convex hull of the origin (zero vector) and the canonical basis of $\mathbb{R}^d$, see the examples in Figure 2.4. Note that since the volume of Δ_d is $\frac{1}{d!}$ its normalized volume is 1 for every d.



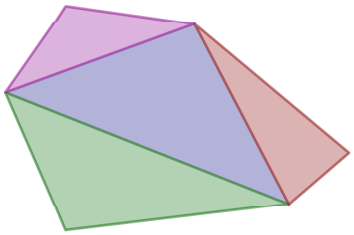
(a) The triangle with vertices $(0,0)$, $(1,0)$ and $(0,1)$.



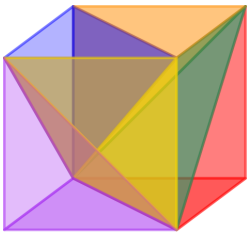
(b) The tetrahedron with vertices $(0,0,0)$, $(1,0,0)$, $(0,1,0)$ and $(0,0,1)$.

Figure 2.5: The simplices Δ_2 and Δ_3 in their respective ambient spaces.

Definition 2 (Triangulation of a polytope). *If we consider a polygon, we can obtain a subdivision of it into triangles that use the vertices of the polygon (see Figure 2.6a), this is called a* triangulation *of the polygon. Generalizing that fact to a d-dimensional polytopes we say that a triangulation of a d-polytope is a subdivision in which every part is a d-simplex whose vertices are vertices of the original polytope.*



(a) A triangulation of the hexagon.

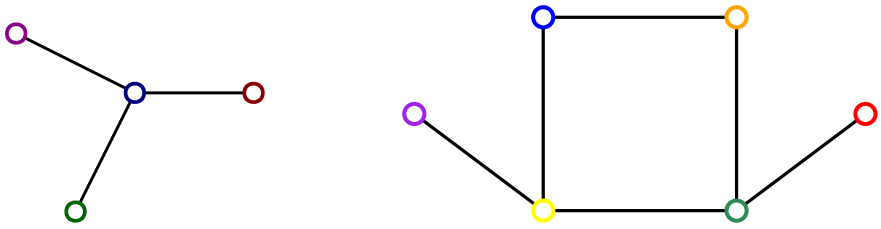


(b) A triangulation of the cube.

Figure 2.6: Examples of triangulations of the polytopes in Figure 2.1.

Graphs are mathematical structures used to abstract or resume information of pairwise relations, like friendships in a social network, computers connected in a network, transportation routes between cities and many other examples. Graphs are formally defined as follows.
A *(simple) graph* is an ordered pair $G = (V, E)$ where V is a set (of *vertices*) and $E \subseteq \{\{x, y\} \subseteq V \mid x, y \in V\}$ (is the set of *edges*). A *directed graph* is a graph where each edge carries a direction. For example, a family of directed graphs in which we are interested are the Caracol graphs, see Figure 4.1a.
A *path* on a graph is a sequence of different edges $e_1, \ldots, e_n$ with $e_k = \{x_k, y_k\}$ in which $y_k = x_{k+1}$ for $k \in [n-1]$. We say that the path is between the vertices v and w (and call them the initial and final vertices of the path) when $v = x_1$ and $w = y_n$. A *cycle* (or closed path) over the graph is a path from a vertex to itself, in other words a path in which $v = w$. A *connected* graph is a graph in which there is a path between every two vertices.

Definition 3. *Every triangulation of a polytope induces a graph called the* dual graph of the triangulation. *The vertices of this dual graph correspond to the parts in the triangulation and there is an edge whenever two parts intersect maximally, that is, if two simplices in the triangulation share a common face of maximal dimension. For the example triangulations in Figure 2.6 the dual graphs are shown below.*



(a) Dual graph of the triangulation of the hexagon in Figure 2.6a.

(b) Dual graph of the triangulation of the cube in Figure 2.6b.

Figure 2.7: Examples of dual graphs associated to the triangulations in Figure 2.6.

2.3 Catalan combinatorics

The *Catalan numbers* (denoted by C_n) are a frequently found number sequence in mathematics, specially in enumerative combinatorics. First appearances in published works date from the 1730's and we can find it as the sequence A000108 in OEIS (The Online Encyclopedia of Integer Sequences [20]). An explicit formula for these numbers is given by

$$C_n = \binom{2n}{n} \frac{1}{n+1}.$$

2.3.1 Combinatorial interpretations

Stanley [22] has curated a list of more than 200 different combinatorial interpretations for the Catalan numbers. We show some of these families in this section.

Triangulations of a regular $(n+2)$-gon

It is a classic result that the number of triangulations of a regular $(n+2)$-gon (see Definition 2) is given by the Catalan number C_n (see Figure 2.8).

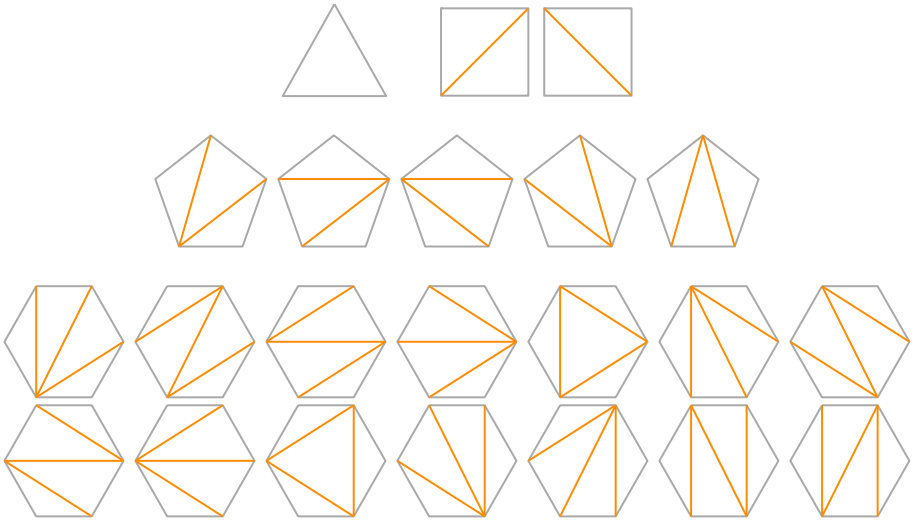


Figure 2.8: Examples of the different triangulations for polygons with 3, 4, 5 and 6 vertices.

Dyck paths

A *Dyck path* of length $2n$ is a lattice path in $\mathbb{Z}^2$ from $(0,0)$ to (n,n) with *north* steps $N := (0,1)$ and *east* steps $E := (1,0)$ such that it never crosses down the axis $x = y$. The set of all Dyck paths of length $2n$ is denoted by D_n and it have size C_n.

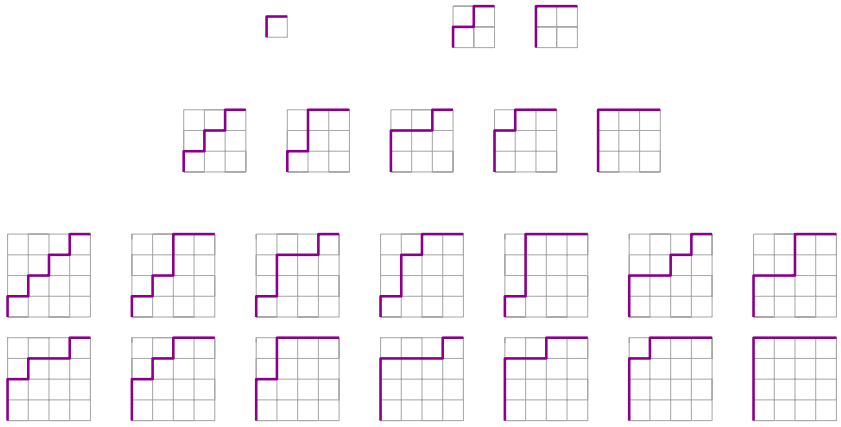


Figure 2.9: Examples of D_n for $n = 1,\ 2,\ 3$ and 4.

Tubings

The following construction is based on Carr and Devadoss [4]. Let $G = (V, E)$ be a simple graph. For a set $T \subseteq V$ the *induced subgraph* by T (denoted by G_T) is the graph whose edges are in $E_S = \{\{v_i, v_j\} \in E \mid v_i, v_j \in T\}$. A *tube* of G is a subset $T \subset V$ such that the induced subgraph G_T is connected. For two different tubes T and W of G we say that they are *compatible* if one of the following conditions is satisfied:

- If $T \cap W \neq \varnothing$ then $T \subset W$ or $W \subset T$.
- If $T \cap W = \varnothing$ then $T \cup W \subset V$ and it is not a tube.

A *tubing* $\mathfrak{T}$ of G is a family of tubes of G such that every pair of tubes is compatible. We say that a tubing $\mathfrak{T}$ is *maximal* if for every tube T of G that is not in $\mathfrak{T}$ the set $\mathfrak{T} \cup \{T\}$ is not a tubing. In other words, if $T \notin \mathfrak{T}$ then T is not compatible with some of the tubes in $\mathfrak{T}$.
The n-line graph L_n is the graph whose edges are defined by $\{(i, i+1)\ with\ 1 \leq i \leq n-1\}$. It is known that the number of maximal tubings of L_n is given by C_n (see [11]). In Figure 2.10 we have illustrated the maximal tubings of L_1, L_2, L_3 and L_4.

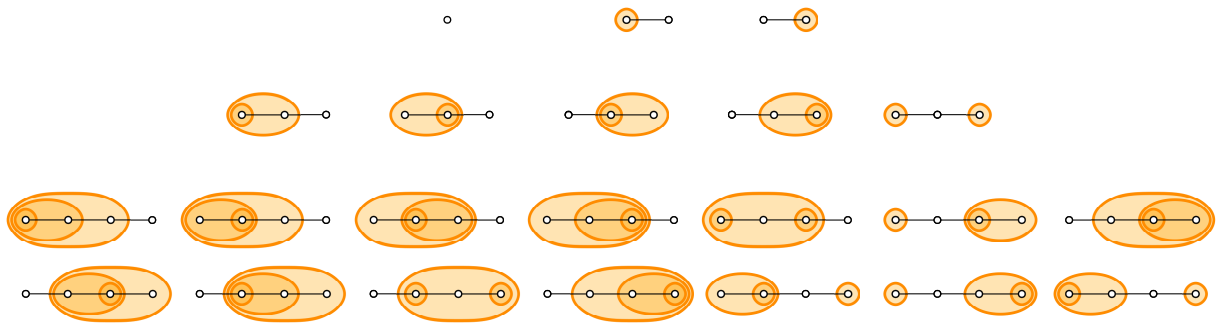


Figure 2.10: Tubings of line graphs L_1, L_2, L_3 and L_4.

2.3.2 Two graphs on the set of Catalan objects

Toggle graph on the set of Dyck paths

We consider the following graph $Togg_n$ on the set of Dyck paths D_n. Given a Dyck path $D \in D_n$ we call a switch to the change between a consecutive sequence of steps EN and NE whenever we obtain a valid Dyck path $D' \in D_n$. We call such a switch a *toggle operation* that obtains D' from D and whenever D and D' are related by a toggle operation we obtain an edge (D, D') in $Togg_n$. The graph $Togg_n$ is called the *Toggle graph* on D_n (see Figure 2.11 for an example).

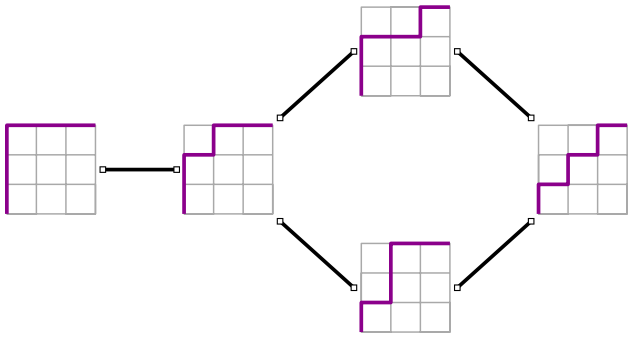
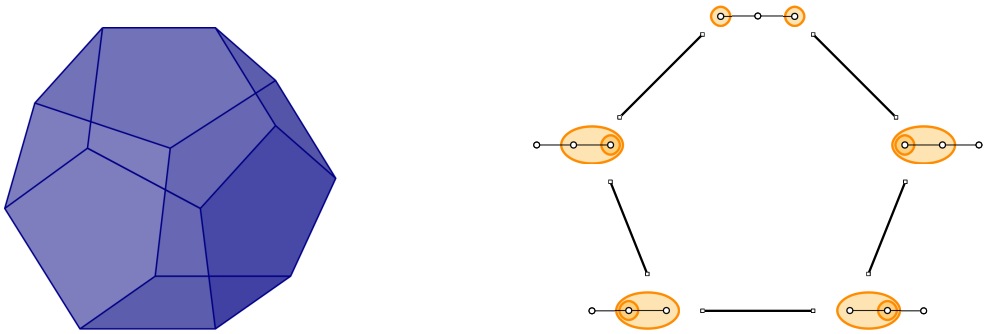


Figure 2.11: Toggle graph $Togg_3$ on D_3.

The associahedron and the Tamari graph

The associahedron K_n (also known as Stasheff polytope) is an n-dimensional convex polytope studied in combinatorics and algebra (see [1] and [13]). This polytope has interesting properties and different but equivalent representations. The number of vertices of this polytope is given by the Catalan number C_{n+1}. One of the multiple ways of constructing the associahedron is given by tubings of line graphs. The 1-skeleton of the associahedron K_n matches with the graph whose vertices are maximal tubings of L_{n+1} and where there is an edge between two tubings $\mathfrak{T}$ and $\mathfrak{S}$ if they differ just in one tube. If we talk about posets, this graph is also known as the Hasse diagram of the Tamari lattice.



(a) The three dimensional associahedron K_3.

(b) 1-Skeleton of the 2-associahedron K_2 constructed with tubings of L_3

Figure 2.12: The associahedron K_3 and the 1-Skeleton of K_2.

If the reader wants to know some more about this polytope we invite to see [18] and the work of Devadoss [4, 9, 10, 11] and Ceballos et al. [5].

Home is behind, the world ahead, and there are many paths to tread through shadows to the edge of night, until the stars are all alight.

J. R. R. Tolkien

3 Triangulations of flow polytopes

3.1 Flow polytopes

Flow polytopes are a family of polytopes that have been studied in combinatorics and geometry by many mathematicians. There has been work of Baldoni and Vergne [2] and unpublished work of Postnikov and Stanley. According to literature (more preciselly in Meszáros et al. [17] and Benedetti et al. [3]) a flow polytope is defined as follows.

Definition 4 (Flow polytope). *Let $G = (V, E)$ a connected graph with $V = [n+1]$ where every element of E is directed from the smallest vertex to the largest. Let m denote the number of edges of G. Given such a graph and a vector $\vec{a} = (a_1, \ldots, a_n) \in \mathbb{Z}^n$ an $\vec{a}$-flow on G is a tuple $(x_{i,j})_{(i,j)\in E}$ of real numbers (i.e. $(x_{i,j}) \in \mathbb{R}^m$) such that for $j \in [n]$*

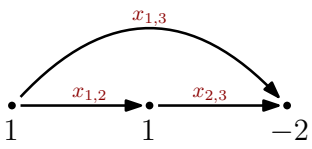
$$\sum_{(j,k)\in E} x_{jk} - \sum_{(i,j)\in E} x_{i,j} = a_j \tag{3.1}$$

One can view an $\vec{a}$-flow as an assignment of flow $b_{i,j}$ to each edge (i,j) such that the net flow at vertex j is a_j and such that the net flow at vertex $n+1$ is $-\sum_{j=1}^{n} a_j$. Define $\mathcal{F}_G(\vec{a})$ to be the set of all $\vec{a}$-flows of G with non-negative entries. Hence, we also have that, in addition to Equation (3.1), the elements of $\mathcal{F}_G(\vec{a})$ satisfy the inequalities

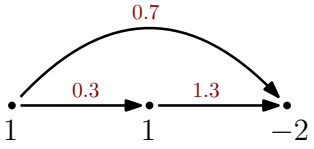
$$x_{i,j} \geq 0 \text{ for all } (i,j) \in E. \tag{3.2}$$

Because of Equations (3.1) and (3.2) we have that $\mathcal{F}_G(\vec{a})$ is an $\mathcal{H}$-polytope and hence a polytope in $\mathbb{R}^m$. This polytope is called the flow polytope *of G with net flow $\vec{a}$. Sometimes we will refer to G as the* flow graph *associated to the polytope $\mathcal{F}_G(\vec{a})$ and to the vertices 1 and $n+1$ as the* source *and* sink *of G respectively.*

To show how to obtain a polytope from a graph we have the following example. Consider the flow graph showed in the Figure 3.1a.



(a) Flow graph G.



(b) Example of flow over G.

Figure 3.1: Example of a flow graph G and some $\vec{a}$-flow over it.

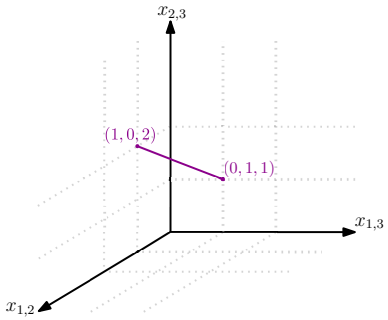
We can view each edge (i,j) as a variable $x_{i,j}$ and we have to make sure that the equations with the form of the Equation (3.1) are satisfied for every vertex in $[n]$. Those equations are

$$x_{1,2} + x_{1,3} = a_1$$
$$x_{2,3} - x_{1,2} = a_2.$$

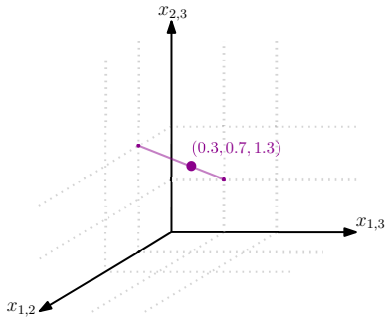
Note that the net flow in this case is $\vec{a} = (1,1)$, so we obtain the equations

$$x_{1,2} + x_{1,3} = 1$$
$$x_{2,3} - x_{1,2} = 1.$$

The above equations have to be satisfied at the same time, this means that we are looking for the intersection of the hyperplanes defined by each of them. In addition, since the $\vec{a}$-flows in $\mathcal{F}_G(\vec{a})$ have only non-negative entries, we are just considering the segment that lives in the first octant of $\mathbb{R}^3$ (i.e. the section in which $x_{1,2} \geq 0, x_{1,3} \geq 0$ and $x_{2,3} \geq 0$) see the following figure.



(a) $\mathcal{F}_G(\vec{a})$ of the graph G in Figure 3.1a.



(b) Point corresponding to the flow in Figure 3.1b.

Figure 3.2: Flow polytope $\mathcal{F}_G(\vec{a})$ of the example graph G in Figure 3.1.

Remark. *Note that with the above example one can realize that every flow over a graph G correspond to a point on their flow polytope $\mathcal{F}_G(\vec{a})$.*

3.2 Totally reductive Postnikov-Stanley triangulations

Mezsáros, Morales and Striker [17] proposed a procedure to construct different subdivisions of a flow polytope, named *framed Postnikov-Stanley triangulations*, because they are based on previous unpublished work of Postnikov and Stanley.

Definition 5. *A weak composition of n in k parts is a vector $(a_1, \ldots, a_k)$ in which every a_i is a non-negative integer and $\sum_{i=1}^{k} a_i = n$. The set of all those compositions is denoted by $\mathbb{WCOMP}(n, k)$.*

Definition 6. *A* bipartite graph *is a graph in which the set of vertices V can be written as the union of two disjoint sets V_1 and V_2 with the condition that there is no edges of the form (v, w) where v and w are either both in V_1 or both in V_2. We call V_1 and V_2 the sets of* left *and* right *vertices respectively.*

Definition 7. *A* tree *is a simple connected acyclic graph. A* bipartite noncrossing tree *is a tree with left vertices $l_1, \ldots, l_i$ and right vertices $r_1, \ldots, r_o$ with no pair of edges (l_n, r_p) and (l_m, r_q) where $n < m$ and $p > q$. The set of such bipartite noncrossing trees is denoted $\mathcal{T}(I, O)$ where $I = (l_1, \ldots, l_i)$ and $O = (r_1, \ldots, r_o)$.*

As it is explained in [17], the bipartite noncrossing trees are in bijection with weak compositions of $o - 1$ into i parts. The bijection is given by sending a tree $T \in \mathcal{T}(I, O)$ to a composition $(a_1, \ldots, a_i)$ where the integer a_k denote the outgoing edges minus 1 at vertex l_k, see Figure 3.3.

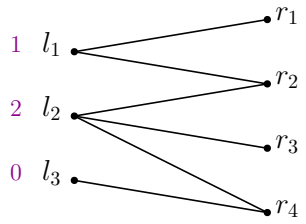


Figure 3.3: Bipartite noncrossing tree associated to the weak composition $(1, 2, 0)$.

Definition 8. *(Danilov et al. [8]) For an* inner *vertex v of a flow graph G (a vertex different to the source and sink) a* framing *at v is a pair of linear orders $(\prec_I, \prec_O)$ over the incoming and outgoing edges of v respectively. If there is a framing at every inner vertex, G is called a* framed graph.

Let G be a framed graph with vertices $V = [n + 1]$ and let I_i and O_i denote the sets of incoming and outgoing edges at vertex i, respectively (ordered according the framing of G). Consider a tree $T \in \mathcal{T}(I_i, O_i)$, for each edge (e_l, e_r) of T where $e_l = (i, j) \in I_i$ and $e_r = (j, k) \in O_i$ let $e_l + e_r$ denote an edge of the form (i, k) which we call the *sum of edges* e_l and e_r. For the sums of edges of the form $(i, j) + (j, k)$, that actually correspond to paths from i to k passing by j, we will use $[i, j, k]$ notation.
Let $G_T^{(i)}$, with $T \in \mathcal{T}(I_i, O_i)$, be the graph obtained by removing the vertex i and all the edges of G that include it and adding the set of edges $\{e_l + e_r | (e_l, e_r)$ edge of $T\}$. A *reduction* of G at the vertex i with respect of T replaces G by the graphs $G_T^{(i)}$ corresponding to each $T \in \mathcal{T}(I_i, O_i)$, keeping which sum of edges of G is each edge of each new graph.
The framing of $G_T^{(i)}$ is inherited from the framing of G as follows:

(i) For each vertex j smaller than i the incoming edges $\mathcal{I}_j(G_T^{(i)})$ are in bijection with $\mathcal{I}_j(G)$. These edges are ordered in the same way as they are ordered in $\mathcal{I}_j(G)$.

(ii) For each vertex j greater than i the outgoing edges $\mathcal{O}_j(G_T^{(i)})$ are in bijection with $\mathcal{O}_j(G)$. These edges are ordered in the same way as they are ordered in $\mathcal{O}_j(G)$.

(iii) For each vertex j smaller than i consider the set $\mathcal{O}_j(G) = \{a_1, \ldots, a_k\}$ ordered linearly according to the framing of G. Since the edges in $\mathcal{O}_j(G_T^{(i)})$ are sums (probably empty) of edges of G with an edge of $\mathcal{O}_j(G)$ thus denote by $S(a_l)$, with $l \in [k]$, to the set of those edges that were summed to the edge a_l. Then let any edge in $S(a_p)$ be less that any edge in $s(a_q)$ for $p < q$. To specify the ordering of the edges within the sets $S(a_l)$, the case where $S(a_l) = \{a_l\}$ is already ordered. Now if $S(a_l) \neq \{a_l\}$ then draw T with the left and right sets of vertices ordered vertically following the framing of G. The edges in $S(a_l)$ are ordered following the order on the edges of the noncrossing bipartite tree T when viewed from top to bottom (smallest edge to largest).

(iv) For each vertex j greater than i the set $\mathcal{I}_j(G_T^{(i)})$ inherit the framing of G analogously to (iii).

We call a *total reduction* of a framed graph G to make reductions on every inner vertex recursively, i.e. make a reduction of G at vertex i and then make reductions of the graphs $G_T^{(i)}$ for every $T \in \mathcal{T}(I_i, O_i)$ at a different vertex j and so on until reductions have been made at all inner vertices of G. We call a *backward total reduction* of G to a total reduction make in sequence from the vertex n to the vertex 2. The set of all those graphs obtained after a total reduction is what we know as a *totally reductive Postnikov-Stanley (TRPS) triangulation* of G (backward *TRPS* triangulation in the case of a backward total reduction). The graphs in the *TRPS* triangulation are called simplices, since their respective flow polytopes are integrally equivalent to simplices of $\mathcal{F}_G(\vec{a})$.

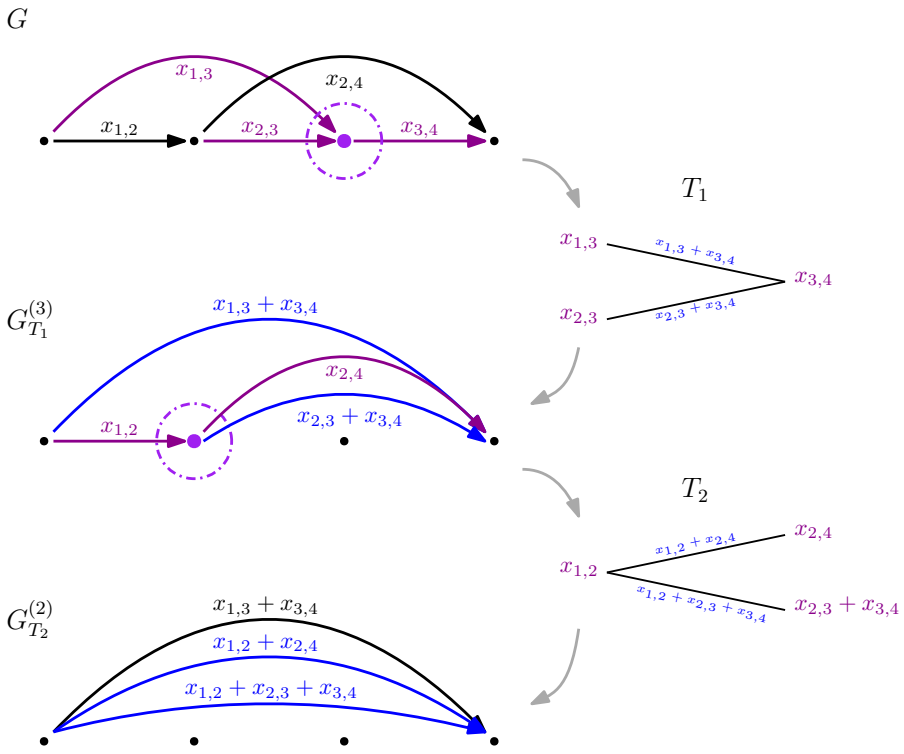


Figure 3.4: Mészáros-Morales-Striker algorithm example applied on a graph G.

We will refer to the entire process to obtain a *TRPS* triangulation as the *Mészáros-Morales-Striker (MMS) Algorithm* and for one step of the respective total reduction we will say that it is an *iteration* of the *MMS* algorithm, see Figure 3.4 for an example. For net flow vector $\vec{a} = (1, 0, \ldots, 0)$ any subdivision of the mentioned above happens to be a (geometric) triangulation of the flow polytope $\mathcal{F}_G(\vec{a})$.

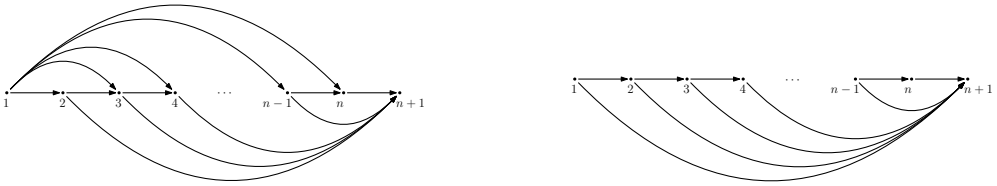
How puzzling all these changes are! I'm never sure what I'm going to be, from one minute to another.

Lewis Carroll

4

Dual graphs of some triangulations of the Caracol flow polytope

Benedetti et al. [3] studied a particular family of flow graphs that they called the *Caracol graph* (because of its similarity in shape with a snail, see Figure 4.1a), denoted by Car_n for every integer n. This graph has $n+1$ vertices and directed edges of the form $(1,i)$, $(i,i+1)$ and $(i,n+1)$ for each $i=2,\ldots,n$. Some readers can be see this graph as some type of merge of two Pitman-Stanley graphs (see Figure 4.1b).



(a) Caracol graph with $n+1$ vertices. **(b)** Pitman- Stanley graph with n vertices.

Figure 4.1: Comparison between Car_n and $PitS_n$ graphs.

According to the description of Car_n is natural to think about its associated flow polytopes. When we consider $\mathcal{F}_{Car_n}(\overrightarrow{a})$ where $\overrightarrow{a} = (1,0,\ldots,0)$ the authors in [3, 17] give two different proofs of the following result.

Theorem 4.1 ([3, 17]). *The normalized volume of $\mathcal{F}_{Car_n}(1,0,\ldots,0)$ is C_{n-2}, the $n-2$ Catalan number.*

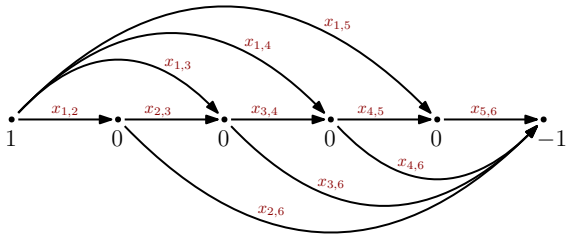


Figure 4.2: The Caracol graph Car_5 with net-flow $(1, 0, \ldots, 0)$.

4.1 *TRPS* triangulations with lex-revlex ordering

Given the results mentioned above is natural to think about further relations between Catalan objects and the Caracol polytopes. We will show characterizations of two *TRPS* triangulations of the $\mathcal{F}_{Car_n}(1, 0, \ldots, 0)$ polytope and its dual graphs in terms of known combinatorial objects in the Catalan family.
Recall that in the context of the flow polytope $\mathcal{F}_{Car_n}(1, 0, \ldots, 0)$ we have a bijection between vertices of the polytope and complete directed paths between the vertex 1 and the vertex $n + 1$. So we will talk indistinctively about the vertices of a simplex in a triangulation of the polytope and a directed path from 1 to $n + 1$.
The *lex-revlex (lexicographical-reverse lexicographical) ordering* on the set of edges of a graph G consist of a linear order in which for a pair of edges (i, j), (k, l) we have $(i, j) \prec (k, l)$ if $i < k$ or $i = k$ and $j > l$. This ordering turns G to a framed graph (since is a total order in the set of the edges of G) with the order restricted to the edges at every inner vertex as framing.

Lemma 4.2. *In Car_n, with the lex-revlex ordering as framing on its edges, when we are applying the MMS algorithm with a backward total reduction at vertex k we have that the ordering on the collection of outgoing paths is given by $\{(k, n + 1), [k, k + 1, P_1], [k, k + 1, P_2], \ldots, [k, k + 1, P_S], [k, k + 1, \cdots, n, n + 1]\}$. Where $P_1, \ldots, P_S$ is the ordering on the edges (paths) that were incident to the edge $(k, k + 1)$ according to the noncrossing bipartite tree chosen in the previous step at vertex $k + 1$.*

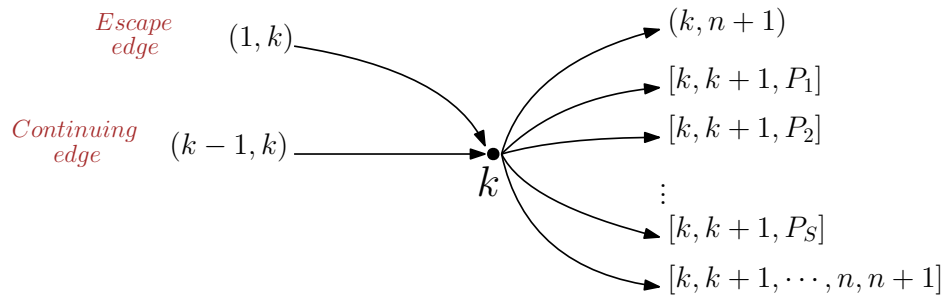


Figure 4.3: Graphic description of the behavior of the *MMS* algorithm at vertex k.

See in Figure 4.3 that in the left hand the lex-revlex ordering is maintained since the smallest edge $(1, k)$ is on top and the largest $(k - 1, k)$ is below it. We will call the edges at the left as *the escape edge* and *the continuing edge* respectively, because the paths that connects with the first one end in that step, and the paths that connects with the other one continue in the next iteration of the algorithm.

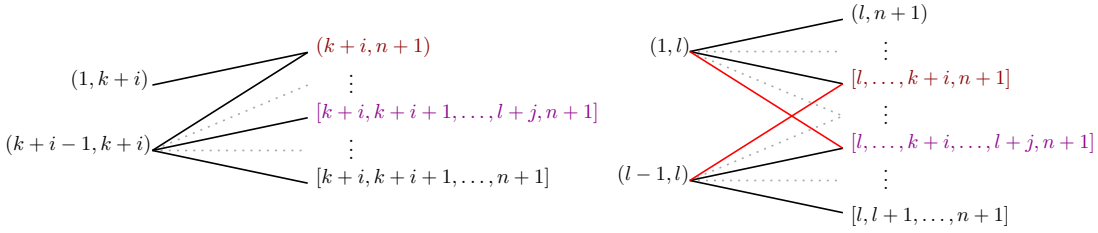
Corollary 4.3. *The paths of the form* $[1, k, n+1]$ *for* $k = 2, \ldots, n$ *and the path* $[1, 2, \ldots, n, n+1]$ *appear as vertices in every simplex in the backward TRPS triangulation. Such paths will be called as the* fixed paths *of the triangulation.*

Proof. This follows from the fact that in a noncrossing bipartite tree, because of the lex-revlex ordering, the escape edge must always be connected to the edge $(k, n+1)$ and the continuing edge must always be connected to the path $[k, k+1, \ldots, n, n+1]$ creating the complete path from 1 to $n+1$. □

We note here that any other path that is not fixed, according to the definition in Corollary 4.3, is of the form $[1, k, \ldots, k+i, n+1]$ for some $k \geq 2$ and $i \geq 1$.

Proposition 4.4. *In any simplex in the backward TRPS triangulation of the Caracol flow polytope* $\mathcal{F}_{Car_n}(1, 0, \ldots, 0)$ *with the lex-revlex ordering as framing, for every pair of edge-paths* $[1, k, \ldots, k+i, n+1]$ *and* $[1, l, \ldots, l+j, n+1]$ *where* $k < l \leq k+i$ *does not happen that* $k+i < l+j$.

Proof. Suppose that we apply the *MMS* algorithm to the graph Car_n within the lex-revlex ordering as framing in every vertex and in some of the simplices we have a pair of edge-paths $p = [1, k, \ldots, k+i, n+1]$ and $q = [1, l, \ldots, l+j, n+1]$ in which $k \leq l \leq k+i < l+j$, see Figure 4.5. Then in the iteration at vertex $k+i$ of the *MMS* algorithm, according to Lemma 4.2, the edge $(k+i, n+1)$ comes before the path $q' = [k+i, k+i+1, \ldots, l+j, n+1]$ (see Figure 4.4a). From the framing inheritance paths continuing both of these paths will preserve the ordering, so for the unique path from the vertex l to $k+i$, $[l, \ldots, k+i, n+1] < [l, \ldots, k+i, \ldots, l+j, n+1]$.



(a) Tree for vertex $k+i$.

(b) Tree for vertex l.

Figure 4.4: Bipartite noncrossing tree representations for the proof of the Proposition 4.4.

Now in the *MMS* algorithm on vertex l we have that $[l, \ldots, k+i, \ldots, l+j, n+1]$ must connect with the escape edge and $[l, \ldots, k+i, n+1]$ must connect with the continuing edge in the noncrossing bipartite tree but this is not possible since this will cause a crossing in the tree, see Figure 4.4b. □

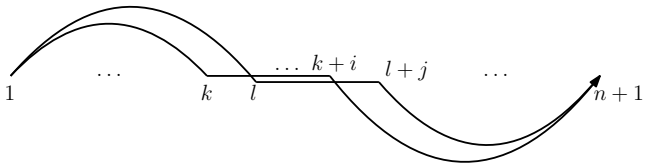
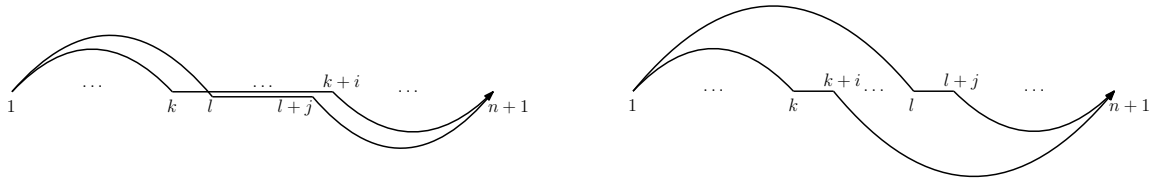


Figure 4.5: Pair of paths $[1, k, \ldots, k+i, n+1]$ and $[1, l, \ldots, l+j, n+1]$ with $k \leq l \leq k+i < l+j$.

Corollary 4.5. *In the MMS algorithm with a backward total reduction of the Caracol flow polytope, for two paths* $p = [l, \ldots, r, n+1]$ *and* $q = [l, \ldots, s, n+1]$ *if* $l < r < s$ *then* $p < q$ *and* p *appears first (above) in the bipartite noncrossing tree representation.*

Definition 9. *Consider any two nonfixed paths (vertices of $\mathcal{F}_{Car_n}(1,0,\dots,0)$) $p = [1,k,\dots,k+i,n+1]$ and $q = [1,l,\dots,l+j,n+1]$ with $k \leq l$. After Proposition 4.4 we will say that p and q are* coherent *if either $k+i < l$ or, if $l \leq k+i$, it does not happen that $k+i < l+j$. We will also say that a collection of nonfixed paths such that are pairwise coherent is called a* coherent collection *of paths. A coherent collection $\mathcal{P}$ such that for any path $p \notin \mathcal{P}$ the condition of being coherent is no longer fulfilled by $\mathcal{P} \cup \{p\}$ is called a* maximal *collection of paths.*

The two situations where two nonfixed paths are coherent are represented in Figure 4.6.



(a) Coherence when $l+j \leq k+i$.

(b) Coherence when $k+i < l$.

Figure 4.6: Representation of a coherent pair of paths in a simplex.

Proposition 4.6. *For a coherent collection $\mathcal{P}$ of nonfixed paths on Car_n there exists a simplex, in the backward TRPS triangulation of $\mathcal{F}_{Car_n}(1,0,\dots,0)$ with the lex-revlex ordering as framing, that contains all elements of $\mathcal{P}$ as vertices.*

Proof. We will show that in the *MMS* algorithm there exists a sequence of bipartite noncrossing trees $T_n, T_{n-1}, \dots, T_2$ where each T_i is chosen at vertex i and that ensures that the paths in $\mathcal{P}$ will be vertices in some simplex of the *TRPS* triangulation. According to the *MMS* algorithm this can be interpreted as for each vertex i we can find a tree T_i such that the paths that pass through i and continue are connected with the continuing edge $(i-1,i)$ and the paths that escape at vertex i are connected with the escape edge $(1,i)$.
Suppose that in some vertex s with $2 < s < n$ is not possible to construct the tree T_s with the mentioned conditions. That means that there exist two different paths $p = [1,k,\dots,s,\dots,k+i,n+1]$ and $q = [1,l,\dots,s,\dots,l+j,n+1]$ in $\mathcal{P}$ with $k+i < l+j$ (this by corollary 4.5 implies that we have $p < q$) such that in the bipartite noncrossing tree T_s we have that the tail $[s,\dots,k+i,n+1]$ of p must be connected to the continuing edge $(s-1,s)$ and the tail $[s,\dots,l+j,n+1]$ of q must be connected to the escape edge $(1,s)$. If this were to be true, in particular we would have that $l = s$ since q escapes at s, $k < s$ since p continues at s, and so $k < l = s < k+i$. However this is will imply by because this would imply according to Proposition 4.4 and Definition 9 that the paths p and q are not coherent. This is a contradiction with the hypothesis that $\mathcal{P}$ is a collection of coherent paths. We conclude then that it is possible to have such a sequence of bipartite noncrossing trees that ensures that the paths in $\mathcal{P}$ will be vertices in some simplex of the *TRPS* triangulation. □

As a consequence of Corollary 4.3, and Propositions 4.4 and 4.6, we can conclude the following proposition.

Proposition 4.7. *In the backward TRPS triangulation of $\mathcal{F}_{Car_n}(1,0,\dots,0)$ with the lex-revlex ordering as framing, simplices are in bijection with maximal collection of coherent nonfixed paths. Furthermore, the collection of vertices of any simplex is the union of the collection of fixed paths and a maximal collection of coherent nonfixed paths.*

After Proposition 4.7 we can characterize the simplices in our triangulation by maximal collections of coherent nonfixed paths. We will turn into a more natural device in the family of Catalan objects that will be useful to characterize these simplices.
Let $\mathcal{P}_{Car_n}$ denote the set of all nonfixed paths on Car_n and $\mathcal{T}_{L_{n-2}}$ the set of tubes of the line graph L_{n-2}. We can define a function

$$\varphi : \mathcal{P}_{Car_n} \longrightarrow \mathcal{T}_{L_n-2}.$$

Consider the set of edges $E = \{(i, i+1) \mid 2 \leq i \leq n-1\}$ of Car_n, since the size of that set is $n-2$ each edge $(i, i+1)$ can be associated to the vertex v_{i-1} of L_{n-2}. In this way the path from 1 to $n+1$ that pass through consecutive edges of E will be sent to the tube that contains exactly the vertices associated to those edges. More precisely the tube $\{v_{i-1}, \ldots, v_{i+j-1}\}$ of L_{n-2} will be the image through φ of the path $[1, i, \ldots, i+j, i+j+1, n+1]$, see Figure 4.7. Observe that φ is clearly a bijective function.

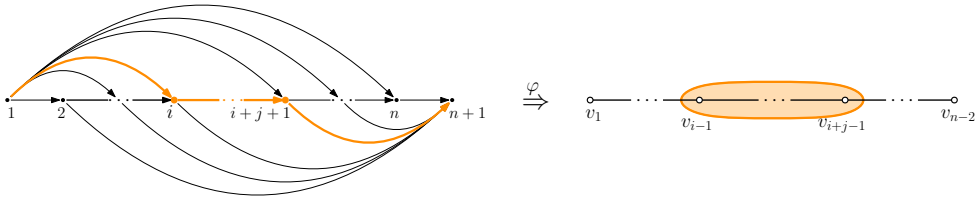


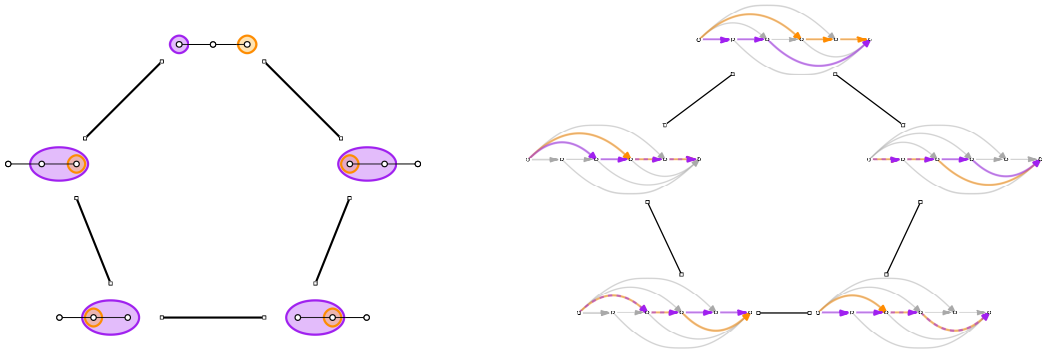
Figure 4.7: Correspondence between tubes and paths.

Proposition 4.8. *The function φ induces a bijection between the sets of coherent collections of nonfixed paths on the Caracol graph Car_n and tubings of the line graph L_{n-2}.*

Proof. We already know that tubes of L_{n-2} correspond to paths on Car_n, we are left to show that p and q are two coherent nonfixed paths in Car_n if and only if $\varphi(p)$ and $\varphi(q)$ are two compatible tubes of L_{n-2}. If p and q are coherent then they are in any of the two situations that appear in Figure 4.6. In the situation in 4.6a we have that $\varphi(p)$ and $\varphi(q)$ are nested. In the situation in 4.6b we have that $\varphi(p)$ and $\varphi(q)$ does not intersect and they are separated at least one vertex. This is exactly the definition of compatibility of tubes given in Section 2.3. □

Theorem 4.9. *The dual graph of the backward TRPS triangulation of the Caracol flow polytope $\mathcal{F}_{Car_n}(1, 0, \ldots, 0)$ given by the lex-revlex ordering as framing on the edges is the 1-skeleton of the $(n-3)$-associahedron.*

Proof. The maximal collections of coherent nonfixed paths characterize the simplices in our triangulation of $\mathcal{F}_{Car_n}(1, 0, \ldots, 0)$ and are in bijection to maximal tubings of L_{n-2}. We are left to prove that the respective geometric simplices connect in the same way that their corresponding set of maximal tubings do. Recall that in the dual graph of the triangulation adjacency of simplices is given by the intersection in maximal facets. This is equivalent as saying that two simplices are adjacent if they differ by exactly one vertex. In the language of tubings this is equivalent to say that they are adjacent if they differ by exactly one tube. This corresponds with the characterization of the 1-skeleton of the $(n-3)$-associahedron that we gave in Section 2.3.2. □



(a) Tubings of a 3-line graph.

(b) Paths on the Caracol graph Car_5.

Figure 4.8: Example of the correspondence between the construction of the 1-skeleton of the associahedron using tubings and the dual graph of our triangulation.

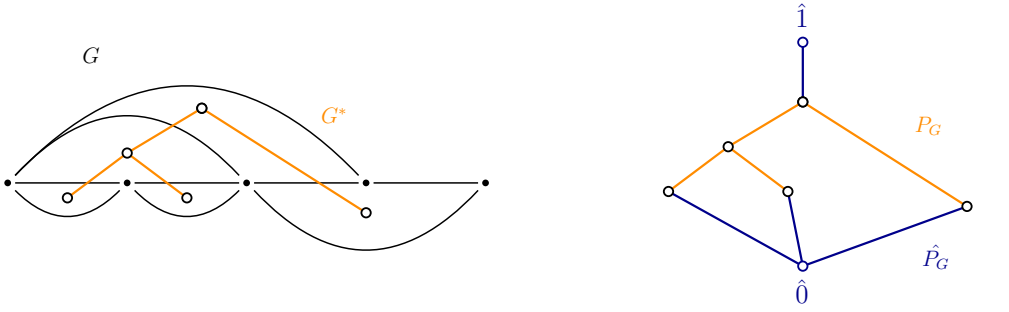
4.2 Order polytopes and *TRPS* triangulations with planar ordering

Stanley [21] studied a family of polytopes called *order polytopes* which are polytopes associated to partially ordered sets (*posets*). We first present results of Mezaros et. al [17], Postnikov (unpublished) and Stanley [21] that relate flow and order polytopes.

Definition 10 (Order polytope). *Let P be a poset with elements $\{t_1, \ldots, t_n\}$, the* order polytope $\mathcal{O}(P)$ *is the set of points $(x_1, \ldots, x_n)$ of $\mathbb{R}^n$ that satisfies $0 \leq x_i \leq 1$ and if $t_i \leq_P t_j$ then $x_i \leq x_j$ for any $i, j = \in [n]$. We can identify each point $(x_1, \ldots, x_n)$ of $\mathcal{O}(P)$ with a monotone function $f : P \to \mathbb{R}$ in which $f(t_i) = x_i$, where $\mathbb{R}$ is considered with its natural total order.*

A *planar graph* is a graph G that has a planar embedding, i.e. it can be drawn without edges crossing. For our purposes our planar graphs will have an embedding satisfying that if $i < j$, and vertex i is in the position (x_i, y_i) and vertex j is in the position (x_j, y_j) in the plane, then $x_i < x_j$.
The *dual graph* of a connected planar graph is a graph whose vertices correspond to the faces of G (the regions delimited by its edges) and it has an edge if two faces are separated by an edge. The truncated dual graph, denoted G^*, is the dual graph of G removing the vertex corresponding to the *exterior* or *infinity* face, this graph can be seen as a dual graph for some polytope subdivision in which the polytope is the polygon delimited by G. The orientation of the edges of G give us an orientation of the edges of its dual G^* from lower to higher y coordinates what allows us to consider that graph as the Hasse diagram of a poset denoted P_G, see Figure 4.9a for an example.
Let P be a finite poset and define the poset $\widehat{P}$ by adding a maximum and a minimum element, denoted $\hat{1}$ and $\hat{0}$ respectively, see Figure 4.9b. P is called a *strongly planar* poset if the Hasse diagram of $\widehat{P}$ has a planar embedding with y coordinates respecting the order of the poset. For example, the poset defined by the relations $a < c$, $a < d$, $b < c$ and $b < d$ is planar but not strongly planar.
Mészáros-Morales-Striker in [17], following unpublished work of Postnikov on flow and order polytopes, proved that if G is a planar graph then the flow polytope $\mathcal{F}_G$ is integrally equivalent to the order polytope $\mathcal{O}(P_G)$. Conversely if P is a strongly planar poset then the order polytope $\mathcal{O}(P)$ is integrally equivalent to the flow polytope $\mathcal{F}_{G_P}$ where G_P is the truncated dual graph of the Hasse diagram of P with two additional edges between $\hat{0}$ and $\hat{1}$ one to each side.

For a poset $P = \{t_1, \ldots, t_n\}$ a *linear extension* of P is a tuple $(t_{a_1}, \ldots, t_{a_n})$ such that whenever $t_{a_i} < t_{a_j}$ in P then we have $a_i < a_j$. Stanley in [21] showed that there exists a triangulation whose simplices are in bijection to the *linear extensions* of the associated poset.



(a) A graph G and its truncated dual G^*.

(b) The poset P_G with added $\hat{1}$ and $\hat{0}$.

Figure 4.9: Example of truncated dual graph and its associated poset (lattice).

Definition 11 (Canonical triangulation of an order polytope [21]). *For a linear extension $(t_{a_1}, \ldots, t_{a_n})$ of a poset P, define the simplex*

$$\Delta_{t_{a_1},\ldots,t_{a_n}} := \{(x_1, \ldots, x_n) \in [0,1]^n \mid x_{a_1} \leq \cdots \leq x_{a_n}\}$$

Note that the $n+1$ vertices of this simplex are $0,\ 1$ vectors whose 0 coordinates are indexed by length k prefixes $t_{a_1}, \ldots, t_{a_k}$ of the linear extension for $k \in [n]$. The simplices $\Delta_{t_{a_1},\ldots,t_{a_n}}$ corresponding to all linear extensions of P are top dimensional simplices in a triangulation of $\mathcal{O}(P)$ that is called the canonical triangulation *of $\mathcal{O}(P)$. In this triangulation two simplices $\Delta_{t_{a_1},\ldots,t_{a_n}}$ and $\Delta_{t_{b_1},\ldots,t_{b_n}}$ intersect maximally if there is an index i such that a_i and a_{i+1} are not comparable in P, and $b_i = a_{i+1}$, $b_{i+1} = a_i$ and $b_j = a_j$ for any other j. In this case we will say that the two linear extensions $(t_{a_1}, \ldots, t_{a_n})$ and $(t_{b_1}, \ldots, t_{b_n})$ are* adjacent.

In Mészáros et al.[17] they prove that a *TRPS* triangulation of the flow polytope $\mathcal{F}_G$ is closely relate to the Stanley canonical triangulation of the corresponding order polytope $\mathcal{O}(P_G)$. More precisely one can obtain a triangulation integrally equivalent to this canonical triangulation using a planar framing in the *MMS* algorithm for the *TRPS* triangulation. Based on that results we can present another *TRPS* triangulation of the Caracol flow polytope $\mathcal{F}_{Car_n}(1, 0, \ldots, 0)$ that corresponds to the canonical triangulation of the order polytope $\mathcal{O}(P_{Car_n})$ associated to the Caracol graph. Note that the representation that we have used for the Caracol graph Car_n makes evident the fact that it is planar.
As we said, the simplices of the canonical triangulation of an order polytope correspond to linear extensions of its associated poset. In the case of the Caracol Car_n the form of P_{Car_n} (see Figure 4.10) allows us to relate the linear extensions of that poset with Dyck paths.

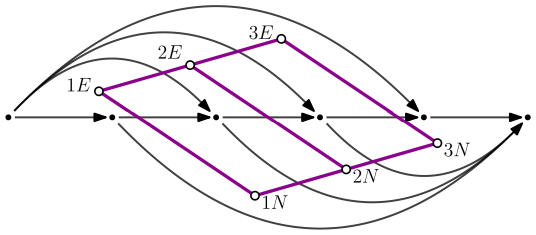


Figure 4.10: Hasse diagram of the poset P_{Car_5} associated to the flow graph Car_5.

The Hasse diagram of the poset P_{Car_n} can be seen as two parallel rows. Let us denote the elements of this poset then as iN or iE if the element is either the i-th element from bottom to top in the lower row or in the higher row respectively, see Figure 4.10 for an example. Note that the order relation in P_{Car_n} is then given by $iN < jN$ or $iE < jE$ whenever $i < j$ and $iN < jE$ whenever $i \leq j$. We can associate the lower row with north (N) steps and the higher to east (E) steps in a lattice path. In this way we associate to a linear extension of P_{Car_n} a lattice path in $\mathbb{Z}^2$ from $(0,0)$ to $(n-2, n-2)$ (since there are $n-2$ elements in each row). Note that, since $iN < jE$ whenever $i \leq j$ in P_{Car_n}, if an element jE comes in a linear extension before iN then we must have $j < i$ which implies that for any linear extension $(t_1, t_2, \ldots, t_{2(n-2)})$ and for every $1 \leq k \leq 2(n-2)$ the number of elements in $(t_1, t_2, \ldots, t_k)$ from the lower row must be at least the same number of elements from the higher row. This precisely corresponds to the path being a Dyck path in D_{n-2}, see Figure 4.11 for an example.

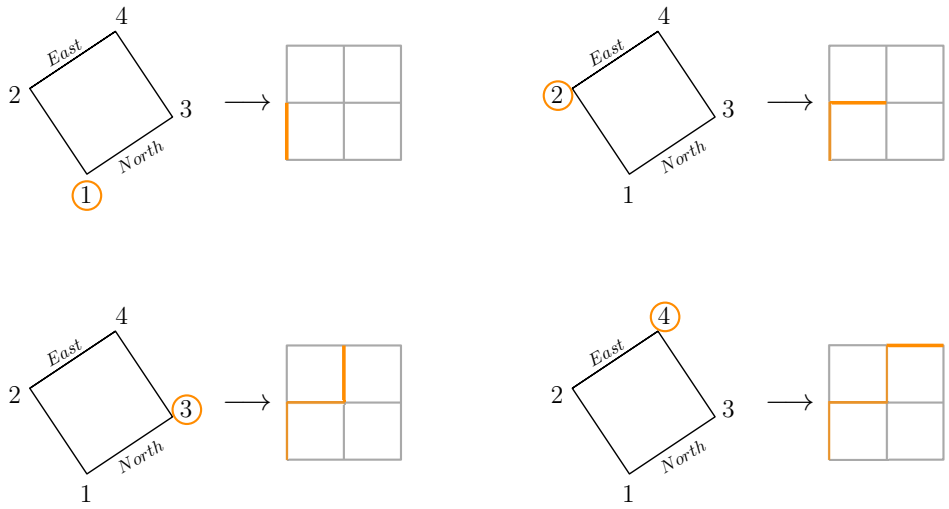


Figure 4.11: Example of how to obtain a Dyck path from a linear extension of P_{Car_4}.

Theorem 4.10. *The dual graph of the backward TRPS triangulation of the Caracol flow polytope $\mathcal{F}_{Car_n}(1,0,\ldots,0)$ given by the planar ordering as framing on the edges is the toggle graph $Togg_{n-2}$ on the set of Dyck paths D_{n-2}.*

Proof. We know that linear extensions correspond to simplices in the triangulation. In order to know how these simplices are connected in the dual graph we can use the notion of adjacency of linear extensions given in Definition 11. Two linear extensions are adjacent if they differ only in a pair of consecutive elements that are not comparable in P_{Car_n} and that appear switched between the two linear extensions. Let $(t_1, t_2, \ldots, t_{2(n-2)})$ be a linear extension of P_{Car_n}. For an index i the following cases can occur for the pair (t_i, t_{i-1}):

1. Case $(t_i, t_{i-1}) = (iN, jN)$ or $(t_i, t_{i-1}) = (iE, jE)$. In this case it must happen that $i < j$ and the elements t_i and t_{i-1} are comparable and cannot be switched.

2. Case $(t_i, t_{i-1}) = (iE, jN)$. In this case it must happen that $i < j$ and the elements t_i and t_{i-1} are not comparable and can be switched.

3. Case $(t_i, t_{i-1}) = (iN, jE)$. In this case the elements t_i and t_{i-1} are not comparable if and only if $i > j$, which corresponds to the case where there are more N steps than E steps in the lattice path associated to the preamble $(t_1, t_2, \ldots, t_i)$.

We can summarize the three cases above in terms of Dyck paths as we can always switch consecutive steps *EN* and we can switch consecutive steps *NE* only when there are a larger number of *N* steps than *E* steps before in the path, that is, only when we get a valid Dyck path. If we recall the construction of $Togg_{n-2}$ in Section 2.3.2, the adjacency of paths is given precisely in those cases. Hence the dual graph of this *TRPS* triangulation is the toggle graph $Togg_{n-2}$, see Figure 4.12 for an example. □

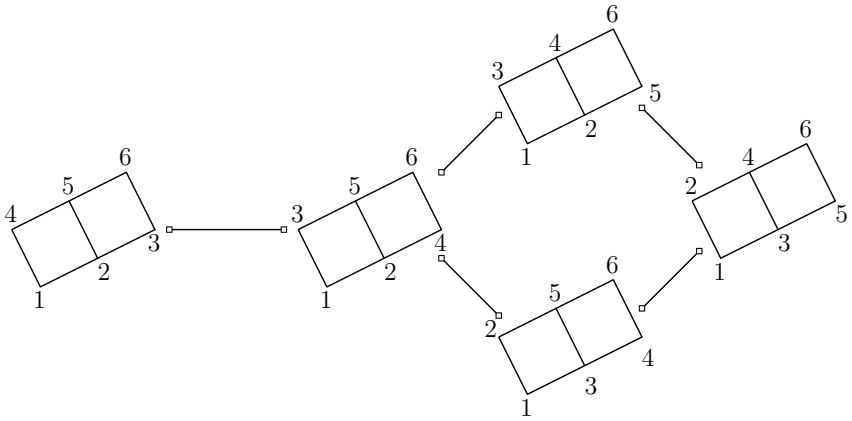


Figure 4.12: The toggle graph $Togg_3$ obtained from linear extensions of P_{Car_5}

A mathematician is a device for turning coffee into theorems.

Alfréd Rényi

5 Conclusions and future works

In this thesis we have provided new ways of how to interpreting some triangulations of the Caracol flow polytopes, using combinatorial objects. This kind of procedures gives a tool to encode and totally characterize the triangulations of this family of polytopes.

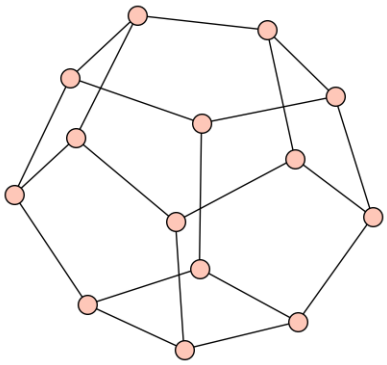
Since we start working from conjectures made using computational aids, we have given a new useful tool to visualize and conjecture what can happen with the triangulations of flow polytopes and their respective dual graphs. The firsts runs we made of the codes were to find something interesting about the dual graphs of the *TRPS* triangulations, but we avoided the condition of inheritance of the framing. With random orderings as framing we also can seen that some dual graphs of these triangulations have crosses, which can implies that the triangulation made in that way is not a regular triangulation. It is then for readers interested in this type of objects to study what happens when the framing is not inherited, and what can happens if the triangulation is not obtained by a backward total reduction but a total reduction made in some different order of iteration (maybe randomly).

Other interesting problem for future researches is to consider the graphs $Car_n^{k,l}$ instead Car_n and study the dual graphs of their flow polytopes, for net flow $(1, 0, \ldots, 0)$. The family $Car_n^{k,l}$ is known as (k, l)-Caracol graphs, and were defined by Sánchez [23] as the Caracol graph removing the k edges $(1, s)$ for $s = n - (k - 1), \ldots, n$ and the l edges $(s, n + 1)$ for $s = 2, \ldots, l + 1$ whose normalized volume is given in terms of counting Dyck paths.

Finally, the problem in which many mathematicians are working on, calculating in a combinatorial way the volume of the $\mathcal{CRY}$ polytope, probably can be studied from the dual graph of some of its triangulations. It is possible that the dual graph of *TRPS* triangulations of that polytope can be obtained from kind of products of the dual graphs that were studied in this work, but it is necessary to find useful combinatorial objects that allows us it.

A Code

The following code was made for the free open-source mathematics software system SageMath and were used to find and visualize examples of flow polytopes and its triangulations. We hope that they can be used for new mathematical researches.



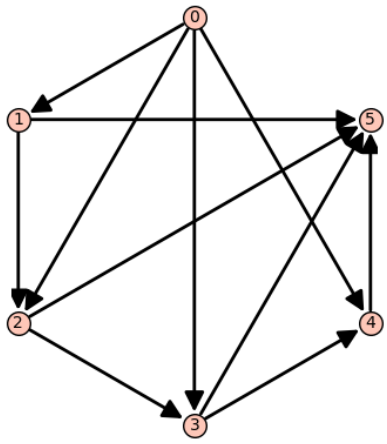
There is a folder in Google Drive to get access to the scripts for the interested reader. The versions showed here are only commented scripts for illustration purposes but the downloadable ones have their own docstring documentations.

Listing A.1: caracol_graph

```
def caracol_graph(n, labels=None):
    #defines the number of vertices and the graph object
    n = n+1
    G = DiGraph(multiedges=True)
    G.add_edge(0,1) #the first edge
    #creates the edges between inner vertices, the edges from 1 to inner vertices and the edges
        from inner vertices to n+1
    for i in range(1,n-2):
        G.add_edges([[i,i+1],[0,i+1],[i,n-1]])
    G.add_edge(n-2,n-1) #the last edge
    #verifies if labes are not given or if the given ones have the correct cardinal
    if labels == None or len(labels) != 3*n-7:
        #if no valid labels were given creates the names of the variables/edges, these edges are
            named according to their corresponding vertices. 'X_i_j' correspond to an edge from
            vertex i to vertex j
        variable_names = ['X%i_%i'%(edge[0]+1,edge[1]+1) for edge in G.edges()]
        reverse_variable_names = copy(variable_names)
        reverse_variable_names.reverse()
        #PolynomialRing object in the rational numbers, with variables names in the list '
            variable_names'
        ring = PolynomialRing(QQ,reverse_variable_names)
        ring.inject_variables() #this create the variables as script objects
        #now polynomial variables are assigned as labels to every edge in the graph
        for i,j,l in G.edges():
            G.set_edge_label(i,j,(eval(variable_names[G.edges().index((i,j,l))]),eval(
                variable_names[G.edges().index((i,j,l))])))
        #shows a message telling that the graph object are succesfully created
        print str(n-1)+"-Caracol graph with edge labels in a polynomial ring."
        return G,ring #returns a tuple with the graph and the polynomial ring
    #if labels with correct cardinal are given, assingn these labels to the edges in the given
        order
    for i,j,l in G.edges():
        #the order of the labels are not related with the names of the edges
        G.set_edge_label(i,j,labels[G.edges().index((i,j,l))])
    #shows a message telling that the graph object are succesfully created
    print "Edge labeled "+str(n-1)+"-Caracol graph."
    return G #returns only the graph with the given labels
```

This function constructs the Caracol flow graph for any positive integer n. Since the flows are represented by polynomials, one can specify the names of the variables.

```
>>> Car_5 = caracol_graph(5)
Defining X5_6, X4_6, X4_5, X3_6, X3_4, X2_6, X2_3, X1_5, X1_4, X1_3, X1_2
5-Caracol graph with edge labels in a polynomial ring.
>>> Car_5[0].plot(layout='circular')
```



Listing A.2: flowgraph_polytope

```
def flowgraph_polytope(graph, net_flow_vector=None, ring=None):
    #keeps the edges and vertices into lists
    edges = graph.edges()
    vertices = graph.vertices()
    #if 'ring' is not given, one PolynomialRing is created according to 'graph'
    if ring == None:
        ring = PolynomialRing(QQ,['X%i_%i'%(edge[0],edge[1]) for edge in graph.edges()])
        ring.inject_variables()
    #if 'net_flow_vector' is not given, a 'default' list is created
    if net_flow_vector == None:
        net_flow_vector = [0 for i in range(len(vertices))] #0 for the entries
        net_flow_vector[0] = 1 #1 in the first entry
        net_flow_vector[-1] = -1 #-1 in the last entry
    #uses the PolynomialRing variable names for find their coefficients
    variable_names = ring.variable_names()
    #the inequalities that describes the polytope are given as lists of coefficients where the
        first entry is the constant term (0) and the other are the coefficients of the label (
        polynomial) of every edge of the graph
    inequalities = [[0]+[edge[2][1].monomial_coefficient(eval(var)) for var in variable_names]
        for edge in edges]
    #creates an empty list for the equations which depends of the edges
    equations = []
    #each vertex have an associated equation depending of its edges
    for vertex in vertices[:-1]:
        #an initial zero polynomial in which saves the corresponding to edges, remember that the
            second entry of an edge is the label (a polynomial)
        polynomial = 0
        #first adding the polynomials of the incoming edges of vertex 'v'
        for incoming in graph.incoming_edges(vertex):
            polynomial = polynomial+incoming[2][1]
        #then subtracting the polynomials of the outgoing edges of vertex 'v'
        for outgoing in graph.outgoing_edges(vertex):
            polynomial = polynomial-outgoing[2][1]
        #save the polynomial only if there are any information
        if polynomial != 0:
            #the net_flow_list at 'v' is the constant term of the polynomial
            equations.append([net_flow_vector[vertex]]+[polynomial.monomial_coefficient(eval(var)
                ) for var in variable_names])
    #the function returns the polytope corresponding to the generated equalities and equations
    return Polyhedron(eqns=equations, ieqs=inequalities)
```

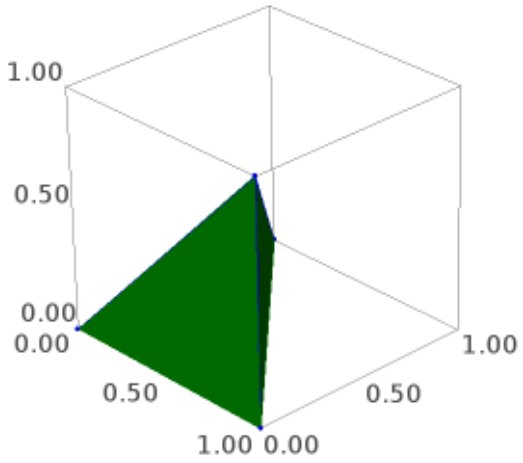
This function constructs the flow polytope associated to a flow graph, a net flow vector and a polynomial ring that is the parent of the labels of the edges of the graph.

Listing A.3: simple_projection

```
def simple_projection(P):
    #Returns the projection of a polytope P into a three dimensional space.
    proj_matrix = matrix.zero(3,P.ambient_dim())
    for i in range(0,3):
        proj_matrix[i,i] = 1
    vert = proj_matrix*P.vertices_matrix()
    return Polyhedron(vertices=vert.transpose())
```

This function constructs a three dimensional projection of the polytope given.

```
>>> net_flow_5 = [1,0,0,0,0,-1]
>>> F_car_5 = flowgraph_polytope(Car_5[0],net_flow_5,Car_5[1])
>>> proj_F_car_5 = simple_projection(F_car_5)
>>> proj_F_car_5.plot()
```



Listing A.4: flowgraph_vertex_reduction

```
def flowgraph_vertex_reduction(graph, vertex, net_flow_vertex, polynomial, random=False):
    #defines an empty list in which the result of the reduction will be stored
    reduction = []
    #list of the edges at 'vertex' for use in the method
    incoming_edges = graph.incoming_edges(vertex)
    outgoing_edges = graph.outgoing_edges(vertex)
    #a flag to check if the set of incoming edges was ordered
    incoming_ordered = 0
    while incoming_ordered == 0:
        incoming_ordered = 1
        for edge in range(len(incoming_edges)-1):
            #insertion sort, reversing the ordering
            if incoming_edges[edge][2][0] < incoming_edges[edge+1][2][0]:
                temp = incoming_edges[edge]
                incoming_edges[edge] = incoming_edges[edge+1]
                incoming_edges[edge+1] = temp
                incoming_ordered = 0
                break
    #a flag to check if the set of outgoing edges was ordered
    outgoing_ordered = 0
    while outgoing_ordered == 0:
        outgoing_ordered = 1
        for edge in range(len(outgoing_edges)-1):
            #insertion sort, reversing the ordering
            if outgoing_edges[edge][2][0] < outgoing_edges[edge+1][2][0]:
                temp = outgoing_edges[edge]
                outgoing_edges[edge] = outgoing_edges[edge+1]
                outgoing_edges[edge+1] = temp
                outgoing_ordered = 0
                break
    #verifies if 'random' is setted as True, if it is True make a shuffle in the edges
    if random:
        shuffle(outgoing_edges)
    #adds an edge to keep the net flow at 'vertex' for use it in the algorithm
    incoming_edges.append((vertex,vertex,(net_flow_vertex,net_flow_vertex)))
```

```
    incoming = len(incoming_edges) #number of incoming edges
    outgoing = len(outgoing_edges) #number of outgoing edges
    #it will runs over all the integer compositions of the number of outgoing edges less 1 on the
         number of incoming edges parts
    for composition in list(IntegerVectors(outgoing-1,incoming)):
        #creates an auxiliar graph for keepeng the original at every composition
        graph_aux = graph.copy()
        graph_aux.delete_vertex(vertex) #to remove all the edges of 'vertex'
        graph_aux.add_vertex(vertex) #and create the same vertex with no edges
        #the polynomial label of the new graphs depending of the composition
        new_polynomial = polynomial+sum([incoming_edges[a][2][1]*composition[a] for a in range(
            incoming)])
        Polynom = 0 #initializes the polynomial used for the labels of the edges
        pos_out = 0 #the position on the list of outgoing edges
        #it will runs over all the indices of the entries of the composition
        for pos_in in range(len(composition)):
            #verify if there are only an edge on that vertex in bipartite tree
            if composition[pos_in] == 0:
                #verifies if it is the last element of the incoming edges
                if pos_in == len(composition)-1:
                    #makes the polynomial and add the new edge to the graph
                    Polynom = outgoing_edges[pos_out][2][1]-Polynom
                    graph_aux.add_edge(incoming_edges[pos_in][0],outgoing_edges[pos_out][1],(
                        incoming_edges[pos_in][2][0]+outgoing_edges[pos_out][2][0],Polynom))
                else:
                    #makes the polynomial and add the new edge to the graph
                    graph_aux.add_edge(incoming_edges[pos_in][0],outgoing_edges[pos_out][1],(
                        incoming_edges[pos_in][2][0]+outgoing_edges[pos_out][2][0],incoming_edges
                        [pos_in][2][1]))
                    Polynom = Polynom+incoming_edges[pos_in][2][1]
            else:
                #makes the polynomial and add the new edge to the graph
                Polynom = outgoing_edges[pos_out][2][1]-Polynom
                graph_aux.add_edge(incoming_edges[pos_in][0],outgoing_edges[pos_out][1],(
                    incoming_edges[pos_in][2][0]+outgoing_edges[pos_out][2][0],Polynom))
                pos_out = pos_out+1 #passes to the next outgoing edge index
                #initializes the number of tree edges at that vertex
                max_tree_edges=1
                while max_tree_edges < composition[pos_in]:
                    #makes the polynomial and add the new edge to the graph
                    graph_aux.add_edge(incoming_edges[pos_in][0],outgoing_edges[pos_out][1],(
                        incoming_edges[pos_in][2][0]+outgoing_edges[pos_out][2][0],outgoing_edges
                        [pos_out][2][1]))
                    Polynom = Polynom+outgoing_edges[pos_out][2][1]
                    pos_out = pos_out+1 #passes to the next outgoing edge index
                    max_tree_edges = max_tree_edges+1 #passes to the next edge
                #verifies if it is the last element of the incoming edges
                if pos_in == len(composition)-1:
                    #makes the polynomial and add the new edge to the graph
                    graph_aux.add_edge(incoming_edges[pos_in][0],outgoing_edges[pos_out][1],(
                        incoming_edges[pos_in][2][0]+outgoing_edges[pos_out][2][0],outgoing_edges
                        [pos_out][2][1]))
                else:
                    Polynom = incoming_edges[pos_in][2][1]-Polynom
                    graph_aux.add_edge(incoming_edges[pos_in][0],outgoing_edges[pos_out][1],(
                        incoming_edges[pos_in][2][0]+outgoing_edges[pos_out][2][0],Polynom))
        #saves the new graph and the corresponding label polynomial in a list
        reduction.append([graph_aux,new_polynomial])
    #returns the list of all the new graphs and its polynomials
    return reduction
```

This function applies a reduction at one specified inner vertex of the flow graph, following the *MMS* algorithm. The order is inherited from the polynomial order given at the edges. One can change the method to give a particular ordering (framing) and obtain different triangulations.

Listing A.5: flowgraph_backward_total_reduction

```
def flowgraph_backward_total_reduction(graph, net_flow_vector, random=False):
    #initializes the list of the triangulation with "graph" in there for the recursive process
    reduction = [[graph,0]]
    #list the inner vertices for which will be applyed the vertex reduction
    vertices = graph.vertices()[1:-1]
    #reverses the list of vertices to allow the application the 'backward total reduction'
    vertices.reverse()
    #runs over every vertex in the graph for make the corresponding reduction
    for vertex in vertices:
        #the reduction will be applied in the graphs in the current iteration
        for G in reduction:
            #to the reduction we append the result of the vertex reduction
            reduction = reduction+flowgraph_vertex_reduction(G[0],vertex,net_flow_vector[vertex],
                G[1],random)
            #and we delete the first one, that is the initial graph
            reduction.pop(0)
    #after the reduction in every vertex returns a list with the simplex graphs
    return reduction
```

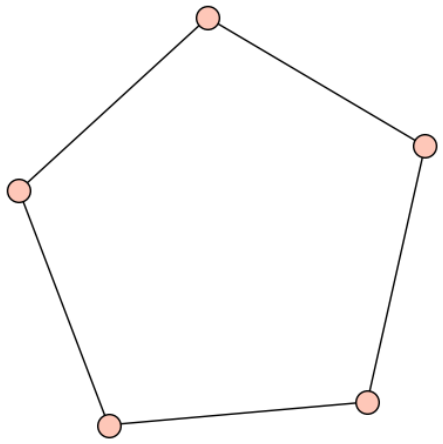
This function generates a list of the flow graphs obtained after applied a backward total reduction according to the *MMS* algorithm, which form a triangulation of the flow polytope associated to the original flow graph.

Listing A.6: flowgraph_dual

```
def flowgraph_dual(graph, net_flow_vector, ring, random=False):
    #calculates the dimension of the flow polytope associated to 'graph'
    dimension = flowgraph_polytope(graph,net_flow_vector,ring).dim()
    #list of flow graphs obtained after applying the MMS algorithm to 'graph'
    graphs = flowgraph_backward_total_reduction(graph,net_flow_vector,random)
    #initializes the lists of the full dimensional simplices(polytopes) and their corresponding
        graphs
    full_dimensional_graphs, full_dimensional_polytopes, volume = [], [], 0
    for G in graphs:
        polytope = flowgraph_polytope(G[0],net_flow_vector,ring)
        if polytope.dim() == dimension:
            #saves in the lists of the simplices and graphs if the actual simplex have the same
                dimension of the initial polytope
            full_dimensional_graphs.append(G)
            full_dimensional_polytopes.append(polytope)
            volume = volume+1
    #initializes the dual graph as a simple undirected graph
    dual_graph = Graph()
    #runs over all the polytopes twice for verify the pairwise adjacency
    for polytope1 in range(volume):
        for polytope2 in range(volume):
            if polytope1 != polytope2:
                #the polytopes are adjacent if their intersection have the dimension of the
                    original polytope less 1
                if full_dimensional_polytopes[polytope1].intersection(full_dimensional_polytopes[
                    polytope2]).dim()==dimension-1:
                    #and if them are adjacent, creates an edge in the dual graph
                    dual_graph.add_edge(full_dimensional_graphs[polytope1][1],
                        full_dimensional_graphs[polytope2][1])
    #the function returns the dual graph of the TRPS triangulation
    return dual_graph
```

This function constructs the dual graph associated to the *TRPS* triangulation obtained after apply the *MMS* algorithm.

```
>>> dual_F_car_5 = flowgraph_dual(Car_5[0],net_flow_5,Car_5[1])
>>> dual_F_car_5.plot(vertex_labels=False)
```



References

[1] Aguiar, M. & Ardila, F. (2017). Hopf monoids and generalized permutahedra. *arXiv preprint arXiv:1709.07504*.

[2] Baldoni, W. & Vergne, M. (2008). Kostant partitions functions and flow polytopes. *Transformation Groups*, 13(3-4), 447–469.

[3] Benedetti, C., González D'León, R., Hanusa, C., Harris, P., Khare, A., Morales, A., & Yip, M. (2019). A combinatorial model for computing volumes of flow polytopes. *Transactions of the American Mathematical Society*, 372(5), 3369–3404.

[4] Carr, M. & Devadoss, S. L. (2006). Coxeter complexes and graph-associahedra. *Topology and its Applications*, 153(12), 2155–2168.

[5] Ceballos, C., Santos, F., & Ziegler, G. M. (2015). Many non-equivalent realizations of the associahedron. *Combinatorica*, 35(5), 513–551.

[6] Chan, C. S., Robbins, D. P., & Yuen, D. S. (2000). On the volume of a certain polytope. *Experimental Mathematics*, 9(1), 91–99.

[7] Corteel, S., Kim, J. S., & Mészáros, K. (2017). Flow polytopes with catalan volumes. *Comptes Rendus Mathematique*, 355(3), 248–259.

[8] Danilov, V. I., Karzanov, A. V., & Koshevoy, G. A. (2012). Coherent fans in the space of flows in framed graphs. *Discrete Mathematics & Theoretical Computer Science*.

[9] Devadoss, S. L. (2009). A realization of graph associahedra. *Discrete Mathematics*, 309(1), 271–276.

[10] Devadoss, S. L., Forcey, S., Reisdorf, S., & Showers, P. (2015). Convex polytopes from nested posets. *European Journal of Combinatorics*, 43, 229–248.

[11] Devadoss, S. L. & O'Rourke, J. (2011). *Discrete and computational geometry*. Princeton University Press.

[12] Hille, L. (2003). Quivers, cones and polytopes. *Linear algebra and its applications*, 365, 215–237.

[13] Lee, C. W. (1989). The associahedron and triangulations of the n-gon. *European Journal of Combinatorics*, 10(6), 551–560.

[14] Mészáros, K. & Dizier, A. S. (2017). From generalized permutahedra to grothendieck polynomials via flow polytopes. *arXiv preprint arXiv:1705.02418*.

[15] Mészáros, K. & Morales, A. H. (2015). Flow polytopes of signed graphs and the kostant partition function. *International Mathematics Research Notices*, 2015(3), 830–871.

[16] Mészáros, K. & Morales, A. H. (2019). Volumes and ehrhart polynomials of flow polytopes. *Mathematische Zeitschrift*, 293(3-4), 1369–1401.

[17] Mészáros, K., Morales, A. H., & Striker, J. (2019). On flow polytopes, order polytopes, and certain faces of the alternating sign matrix polytope. *Discrete & Computational Geometry*, 62(1), 128–163.

[18] Müller-Hoissen, F., Pallo, J. M., & Stasheff, J. (2012). *Associahedra, Tamari lattices and related structures: Tamari memorial Festschrift*, volume 299. Springer Science & Business Media.

[19] Schrijver, A. (2003). *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media.

[20] Sloane, N. J. A. (1996). *The On-line Encyclopedia of Integer Sequences (OEIS).* OEIS Foundation, Incorporated.

[21] Stanley, R. P. (1986). Two poset polytopes. *Discrete & Computational Geometry*, 1(1), 9–23.

[22] Stanley, R. P. (2015). *Catalan numbers*. Cambridge University Press.

[23] Sánchez Vargas, C. (2019). *El volumen del politopo de flujo de un subgrafo del Caracol*. Universidad de los Andes.

[24] Zeilberger, D. (1999). Proof of a conjecture of chan, robbins, and yuen. *Electron. Trans. Numer. Anal*, 9(147-148), 1–2.

[25] Ziegler, G. M. (1995). *Lectures on Polytopes*. Graduate texts in mathematics 152. Springer.

This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 12 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface.

Most of the images were made using the Ipe extensible drawing editor, developed by Otfried Cheong since 1993 and initially worked on SGI workstations only. It can be downloaded by free on their webpage.

A template that can be used to format a PhD thesis with this look and feel has been released under the permissive MIT (X11) license, and can be found online at github.com/suchow/Dissertate.