UNIVERSIDAD DE GUANAJUATO

DIVISIÓN DE CIENCIAS NATURALES Y EXACTAS

Graph and string representation of trivalent 2-stratifolds with trivial fundamental group

presentada por

Myriam Hernández Ketchul

Tesis para obtener el grado de

Licenciada en Matemáticas

Asesor de Tesis: Dr. José Carlos Gómez Larrañaga

May 2023

I don't want my thoughts to die with me, I want to have done something. I'm not interested in power, or piles of money. I want to leave something behind. I want to make a positive contribution - know that my life has meaning.

Temple Grandin

Acknowledgements

I want to thank my supervisor, Dr. José Carlos, for all his patience and support during the time we worked on this project. Because even when we started at the beginning of the pandemic of COVID, we achieved the proposed goals.

Thank you very much to Dr. Jesús Rodríguez Viorato for his help during the implementation of the algorithm, because he had key ideas to make it work.

Thanks to all the members of the jury board, for their advice and comprehension and also for working on a thigh schedule.

I want to say special thanks to all the people that I met during my university studies. All the investigators, teachers, and of course classmates. I've learned a lot from you and I'm very glad our paths crossed.

Thank you to all my friends that listened me during this process. You were invaluable support and without you, it would have been very difficult to overcome the adversities.

And finally, thanks to my family that supports me on all my objectives and share with me my achievements.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

In topology we are interested in finding a classification of the objects that we work with and find invariants that allows us to identify them. The main reason of this is because in most of the cases we are unable to see the object of study but we need its properties in order to be capable of work with it.

An **invariant** for a family of objects $\mathcal{A}$ with a certain equivalence relationship is a function ϕ from $\mathcal{A}$ to a family $\mathcal{B}$, such that if α_1 and α_2 belong to the same class of equivalence then $\phi(\alpha_1) = \phi(\alpha_2)$. A complete invariant is an invariant ϕ that satisfies that $\phi(\alpha_1) = \phi(\alpha_2)$ if and only if α_1 and α_2 belong to the same class of equivalence.

One example of a theory that is based on this search of invariants is knot theory, which main goal is to find a complete invariant that not only differentiates but identifies uniquely every knot. The tables of knots diagrams were the beginning of this attempts to classify the knots. Also there exist even internet websites [1] specialized in the classification of known knots and the values of some invariants known for those knot.

As knots, other important objects studied in low dimensional topology are manifolds. Specially 2- and 3-manifolds. An **m-manifold** is a Hausdorff space M, second-countable, such that each point x of M has a neighborhood that is homeomorphic with an open subset of $\mathbb{R}^m$, as defined in [2]. A 2-manifold is called a **surface** and as in the case of knot theory, there have been many efforts to classify these mathematical objects.

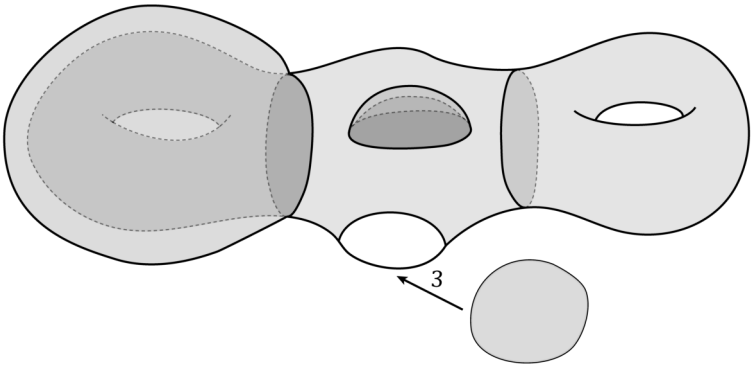
In [3], we can read about the remarkable work that was made to prove the Classification Theorem for Compact Surfaces. This theorem states that every compact connected closed surface is either a sphere, a connected sum of g tori for $g \geq 1$, or a connected

sum of k projective planes for $k \geq 1$. Here the complete invariant used to identify the surfaces is the Euler-Poincaré Characteristic.

The proof of this theorem is really complex. First we need some standard surfaces that can be easily categorized, these are the standard connected sum of g tori, the standard connected sum of k projective planes and the sphere. Then we need to prove two things. The first one is to prove that every surface can be triangulated and the second one is to prove that there are some operations that work on specific sets of triangles that allows us to find piecewise linear homeomorphisms from the set of triangles to the parts of the standard surfaces. Then with these two proofs, you can say that given any surface, it is possible to find a homeomorphism from any surface to a standard surface. More details about this proof can be found in [3].

This theorem is crucial as the motivation of this work.

In colloquial words, a 2-stratifold is some kind of generalization of the concept of surface. As defined before a surface is a Hausdorff space, second-countable such that each point x of the space has a neighborhood that is homeomorphic with an open subset of $\mathbb{R}^2$. But we can go further, we can think of a space that is similar to a surface in most of its points, but that also has a collection of disjoint curves where n half-planes intersect. In that case we will have an $n-$valent 2-stratifold.



——

FIGURE 1.1: An example of a 2-stratifold

Since surfaces can be completely classified, one can wonder if 2-stratifolds can be completely classified as well. We will focus on the next immediate case to the surfaces,which are the trivalent 2-stratifolds.

2-stratifolds are also important because they appear on the applications of Topological Data Analysis, since 2-stratifolds can be seen as 2-dimensional simplicial complexes. Classifying the 2-stratifolds can lead to a better understanding of simplicial complexes such as the ones that appear in [4] and [5]. Some of them are equivalent to trivalent 2-stratifolds graphs, which are defined further on this work.

On the other hand 2-stratifolds are a particular case of the mathematical object called **foam space**. This topological spaces are based on the geometry of the bubbles and foam, and are important in physics. The study of foam spaces is relevant for the development of better temperature or impact insulators as stated in [6]. Therefore, the study of the particular case of the 2-stratifolds can lead to generalizations applicable to foam spaces.

In Chapter 2, we will give basics definitions for understanding this thesis.

Next, Chapter 3 describes our objects of study, trivalent 2-stratifolds with trivial fundamental group, along with results that have been already published about them.

This leads us to Chapter 4, where it is explained how we can see every 2-stratifold as a graph. Moreover, we are going to explain the algorithm that allows us to build every trivalent 2-stratifold from two basic graphs called seeds.

The relation between 2-stratifolds and graphs is similar to the relation between surfaces and triangulations, that is why we define an invariant that works over the graph representation of the 2-stratifolds. This invariant is the **string representation** and it's completely explained on Chapter 5.

The construction of 2-stratifolds is an iterative process and during that process we can calculate their string representation. Part of the work done for this thesis was to implement the algorithm as a Python program that not only builds the graphs of the 2-stratifolds but also draws them and assigns them their string representation. More details of this implementation can be read in Chapter 6.

Finally, on Chapter 7 we discuss the conclusions from this work and propose more questions that could lead us to further work on the same topic, that may be developed on the future.

Chapter 2

Preamble

In this chapter we are going to define some key concepts that are necessary to understand this thesis. First we are going to talk about basic concepts of topology such as the definition of a 2-stratifold and a brief introduction to the fundamental group of a 2-stratifold. That would lead us to the main object of study in this work that is the trivalent 2-stratifolds with trivial fundamental group. Most of these definitions come from [7] and [2]. Afterwards we are going to show some results and definitions from graph theory that are necessary in further chapters.

2.1 2-Stratifolds

We are going to begin defining Hausdorff space, we will say that a topological space X that satisfies for each pair x_1, x_2 of distinct points of X there exist neighborhoods U_1, U_2 of x_1 and x_2, respectively, that are disjoint, then the space is a **Hausdorff space**. We will also say that it has a **countable basis at some point** $x \in X$ there is a countable collection $\mathcal{B}$ of neighborhoods of x such that each neighborhood of x contains at least one of the elements of $\mathcal{B}$. If a topological space X has a countable basis for its topology, the X is **second-countable**

A space X is said to be **connected** if any two points in X can be connected by a curve lying wholly within X. If X is a connected topological space, it is **pathwise-connected** if and only if for every two points $x, y \in X$ there is a continuous function $f : [0, 1] \to X$ such that $f(0) = x$ and $f(1) = y$. The spaces studied in this work fulfill the property of being locally pathwise-connected. Therefore, the fact that one of these

spaces is connected implies that it is also pathwise-connected, so we will assume that every connected space is also pathwise-connected.

We mentioned manifolds on Chapter 1. Let's remember the definition, a **n-manifold** is a Hausdorff space X, second-countable, such that each point x of X has a neighborhood that is homeomorphic with an open subset of $\mathbb{R}^n$. The 1-manifolds are often called **curves** and the 2-manifolds are called **surfaces**.

Now we can begin to define the main object of study that are the 2-stratifolds.

A **2-stratifold** is a compact, connected Hausdorff space X together with a filtration $\emptyset = X_0 \subset X_1 \subset X_2 = X$ by a closed subspace such that X_1 is a closed 1-manifold, each point $x \in X_1$ has a neighborhood homeomorphic to $\mathbb{R} \times CL$, where CL is the open cone on L for some (finite) set L of cardinality greater than 2 and each $x \in X_2 \backslash X_1$ has a neighborhood homeomorphic to $\mathbb{R}^2$.

As defined in [8], given a topological space L, the open cone CL is the result of collapsing one face of the cylinder $L \times [0, 1]$ to one point. Therefore $CL = (L \times [0, 1])/L \times \{1\}$.

The cardinality of the set L mentioned in the definition of the 2-stratifold, defines the valence of the points of X. In particular if the set L has cardinality 3, then we will say that we have a **trivalent point**.

One can see this kind of topological space on the real world, for example with bubbles, a single bubble is a sphere but if you put one near to other when they touch (usually) they don't form a bigger bubble, they join together with a wall in the middle as in Figure 2.1.

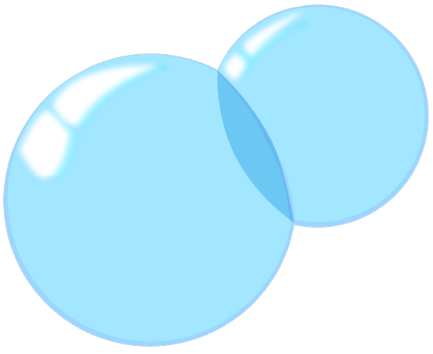


FIGURE 2.1: Bubbles forming a trivalent 2-stratifold

If you add more bubbles, you can manage to get a more complex 2-stratifold as in Figure 2.2.

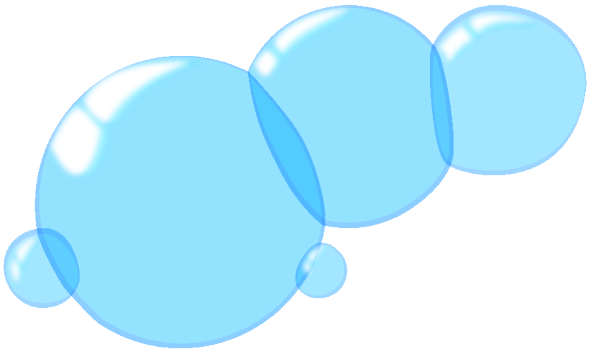


FIGURE 2.2: Bubbles as a 2-stratifold

The previous were orientable examples of 2-stratifolds, some that we can see. But there are also nonorientable 2-stratifolds, some that we can't embedded in our tridimensional world.

We have said that the main goal is to classify the 2-stratifolds, but trying to do it with all of them would be ambitious. We must narrow down the set of 2-stratifolds that we are going to work with.

Let us properly define the valence of the 2-stratifolds.

Definition 2.1. Given a 2-stratifold X which contains a closed (possibly disconnected) 1-manifold X_1 with empty boundary as a closed subspace. Let $C \approx S^1$ be a component of X_1, it has a neighborhood $N(C)$ that is homeomorphic to $(Y \times [0,1])/(y,1) \sim (h(y),0)$, where Y is the closed cone on the discrete space $\{1,2,...,d\}$ and $h : Y \to Y$ is a homeomorphism whose restriction to $\{1,2,...,d\}$ is the permutation $p : \{1,2,...,d\} \to \{1,2,...,d\}$. The components of $\overline{N(C) - C}$ are called the **sheets** of $N(C)$. Therefore $N(C)$ will have d sheets and the valence of C is d.

As a remark is useful to say that the space $N(C)$ depends only on the conjugacy class of $p \in S_d$ and therefore is determined by a partition of d. Also for two components $C, C' \in X_1$, the neighborhoods $N(C), N(C')$ are chosen sufficiently small so that $N(C)$ is disjoint from $N(C')$.

This definition is equivalent to this other version: Given a 2-stratifold X which contains a closed (possibly disconnected) 1-manifold X_1 with empty boundary as a closed subspace. Let $C \approx S^1$ be a component of X_1 and a point $x \in C$. Consider a regular neighborhood $N(x)$ of x. By definition of 2-stratifold, $N(x) - C$ is a set of disjoint components locally homeomorphic to $\mathbb{R}^2$. We will say that x is **d-valent** if $|N(x) - C| = d$

and we will name the components of $N(x) - C$ as **sheets**. By definition, each point of C has the same valence, therefore we will say that C is **d-valent**.

The sheets looks as follows:

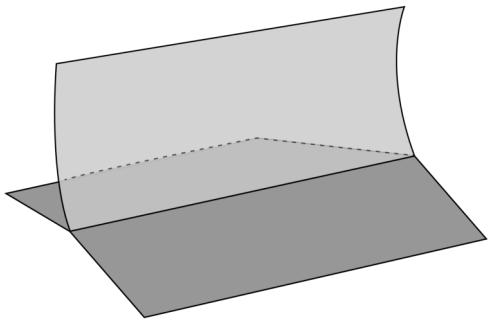


FIGURE 2.3: Sheets of $N(C)$

If for every connected component C of X_1, $N(C)$ has n sheets, we will say that the 2-stratifold is n-**valent**. In particular we will focus on the case where for every component C of X_1, $N(C)$ has exactly 3 sheets, these are called the **trivalent 2-stratifolds**. This is our first reduction of cases but we will need another one, which involves the following definitions from [2].

Let X be a space and x, y two points of X. A **path** in X from x to y is a continuous map $f : [0, 1] \to X$ such that $f(0) = x$ and $f(1) = y$. One can think of it as a continuous curve in X that connects x and y.

But (unless X is a 1-manifold) there are several ways of connecting every two points, so we are going to define a relation of equivalence between paths. This relation is called path homotopy.

Definition 2.2. Given two paths f and f', from x to y on X, they are said to be **path homotopic** if there is a continuous map $F : [0, 1] \times [0, 1] \to X$ such that

$$\begin{aligned} F(s, 0) = f(s) \quad &\text{and} \quad F(s, 1) = f'(s), \\ F(0, t) = x \quad &\text{and} \quad F(1, t) = y, \end{aligned}$$

for each $s \in [0, 1]$ and each $t \in [0, 1]$.

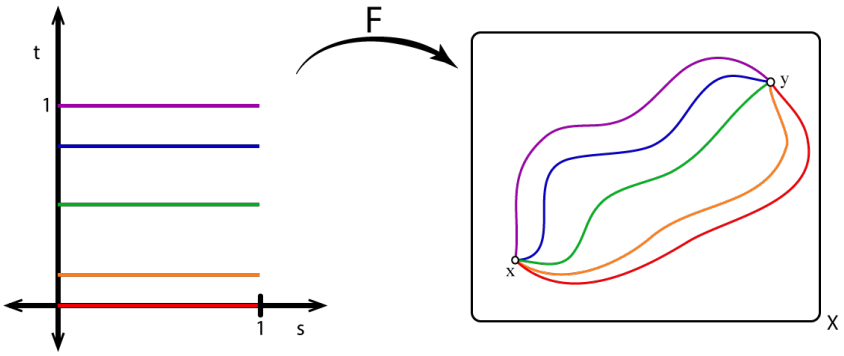


FIGURE 2.4: Path homotopy

Now we are ready to define the fundamental group of any space X.

Definition 2.3. Let x_0 be a point of X. A path in X that begins and ends at x_0 is called a **loop** based at x_0. The set of path homotopy classes of loops based at x_0, with the operation composition of paths, is called the **fundamental group** of X relative to the base point x_0. It is denoted by $\pi_1(x_0, X)$.

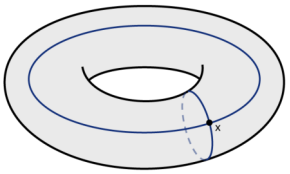


FIGURE 2.5: Class representatives of the fundamental group of the Torus

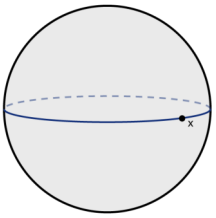


FIGURE 2.6: Class representative of the fundamental group of the Sphere

We can see that in the case of the sphere, in Figure 2.6, the representative path can be contracted with a continuous transformation to the single point x_0. As seen in Figure 2.7.

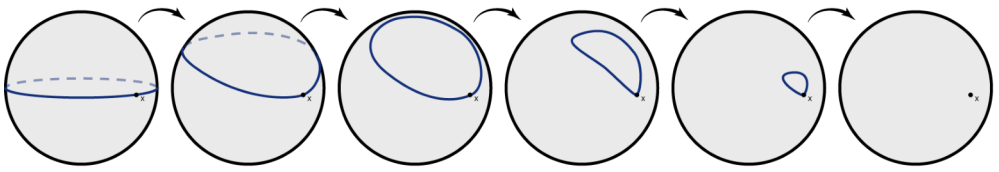


FIGURE 2.7: Trivial loop

This is called the trivial element. When the fundamental group based on $x_0 \in X$ of a space X consists on this single element, we say that X is **simply connected** or that it has **trivial fundamental group**. It is implied that if $\pi_1(x_0, X)$ is the trivial fundamental group, for any $x \in X$, $\pi_1(x, X)$ is going to be the trivial fundamental group.

One can also calculate the fundamental group of 2-stratifolds. We have the following examples of 2-stratifolds with a nontrivial fundamental groups:

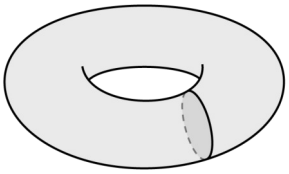


FIGURE 2.8: Trivalent 2-stratifold with a nontrivial fundamental group

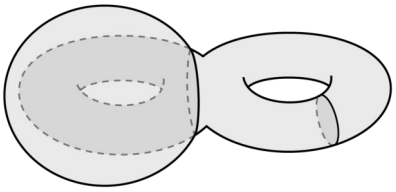


FIGURE 2.9: Trivalent 2-stratifold with a nontrivial fundamental group

Again, the possibilities are endless and trying to work with all of them would be overwhelming. So we are just going to focus on the trivalent 2-stratifolds with trivial fundamental group.

Now we are able to give a proper definition of our object of study.

A **trivalent 2-stratifold with trivial fundamental group** is a compact, connected Hausdorff space X together with a filtration $\emptyset = X_0 \subset X_1 \subset X_2 = X$ by a closed subspace such that X_1 is a closed 1-manifold, each point $x \in X_1$ has a neighborhood homeomorphic to $\mathbb{R} \times CL$, where CL is the open cone on L for some (finite) set L of cardinality 3 and each $x \in X_2 \backslash X_1$ has a neighborhood homeomorphic to $\mathbb{R}^2$.

2.2 Introduction to Graph Theory

In this section we are going to introduce the first notions and nomenclature necessary to understand the further theory that we are going to develop. Most of the definitions of this section are from the book [9, Chapter 1].

First is necessary to introduce the definition of a graph. A **graph** is a nonempty set V of elements called **vertices** (**vertex** in singular) together with a set E of subsets of two vertices, called **edges**. We will denote as $V(G)$ and $E(G)$ the set of vertices and edges of a given graph G.

Normally one can draw the graphs as diagrams, representing each vertex as a dot and every edge as an arc. For example, if an edge is the subset of vertices $\{u, v\}$ therefore the representation of that edge in the diagram is an arc that connects the dot representing u with the dot representing v. We will refer to the diagram as the graph itself.

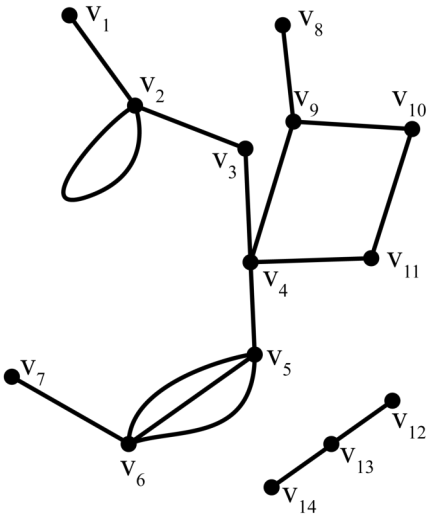


FIGURE 2.10: Diagram of a Graph

Given a graph G, for a vertex v of G, we will define the **degree** of v as the number of edges in G containing v, it will be denoted as $deg(v)$. We will also say that if an edge e contains the vertex v then, e and v are **incident** to each other. If the degree of v is 1, then v is called a **leaf** of G. In Figure 2.10, the vertices $v_1, v_7, v_8, v_{12}, v_{14}$ are the leaves of the graph.

If $\{u, v\}$ is an edge of G, then u and v are **adjacent vertices**. Also it is said that u and v are **neighbors**. An edge containing twice the same vertex is called a **loop**.

Let u, v be vertices of G. A $u - v$ **walk** W in G is a sequence of vertices of G that starts in u and ends in v, such that consecutive vertices in W are adjacent in G. The number of edges encountered in W is the **length of W**. If the beginning and ending vertices of a walk are different we have an **open walk**, otherwise it is a **closed walk**.

A walk with no repeated edges is called a **trail**. Conversely, when the walk has no repeated vertices is called a **path**. A nontrivial (with more than one vertex) closed walk in which no edge is repeated is a **circuit**. And a circuit where all the vertices (except from the ones in the extremes) are different is called a **cycle**.

For example, in Figure 2.10, the walk $(v_8, v_9, v_{10}, v_{11}, v_4, v_9)$ is a trail. The walk (v_{12}, v_{13}, v_{14}) is a path. On the other hand, $(v_9, v_{10}, v_{11}, v_4, v_9)$ is a cycle.

For u, v vertices of G, we will say that u is **connected** to v if there exists a $u - v$ path in G. And the graph G is a **connected graph** if every two vertices of G are connected.

The graph of Figure 2.10, is not connected since there is no path that connects v_5 and v_{12}.

2.3 Special kinds of graphs

We have mentioned in the definition of a graph that it is a collection of vertices with a collection of subsets of those vertices called edges. We can go further and associate a number to every edge of a graph, this number would be called a **weight** and the resulting graph is called a **weighted graph** as the one in Figure 2.11.

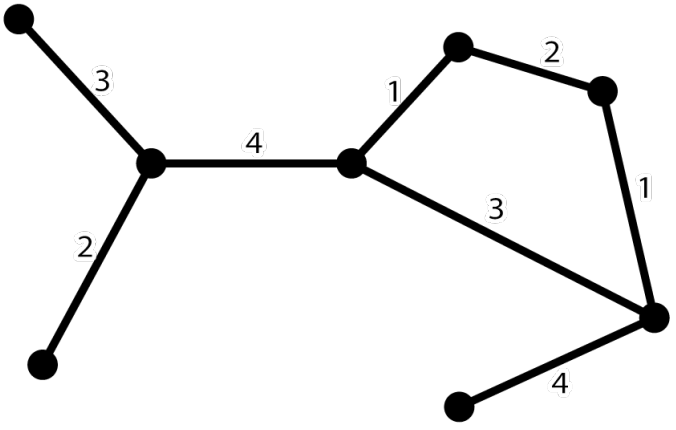


FIGURE 2.11: Example of a weighted graph

One can also associate a number to every vertex of the graph, this process is known as **labeling** and the number of a vertex is a **label**. This kind of graphs are called **labeled graphs**. More information about these graphs can be consulted in [10].

Other interesting kind of graphs are the bipartite graphs. As defined in [9], a nontrivial graph G is **bipartite** if it is possible to partition $V(G)$ into two subsets B and W, called **partite sets**, such that every edge of G joins a vertex of B and a vertex of W. We will normally refer to the set B as the set of black vertices of G and W as the set of white vertices of G, denoted as $B(G)$ and $W(G)$, respectively. The graph in Figure 2.12 is an example of a bipartite graph.

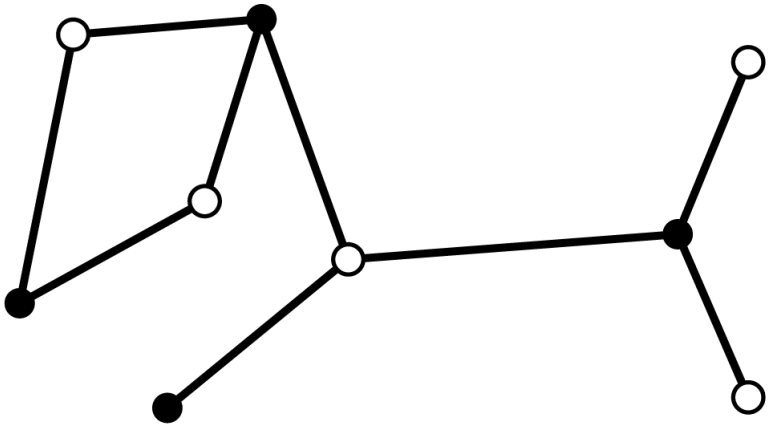


FIGURE 2.12: Example of a bipartite graph

Now is time to talk about trees, a **tree** is a connected graph with no cycles nor loops. An example can be seen in Figure 2.13.

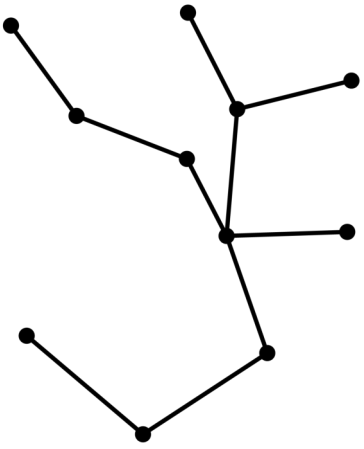


FIGURE 2.13: Example of a tree

An important property of the trees to take in consideration is the fact that for any pair of vertices u, v on a tree, there exists a *unique* $u-v$ path. More properties about graphs are going to be necessary but we will discuss them as we need them.

A tree with a marked vertex is called a **rooted tree**. And the vertex that is marked is known as the **root** of the graph. Once we have a root, we can order the graph putting the root at the top and the other vertices below it. This will induce an order on the graph as in Figure 2.14.

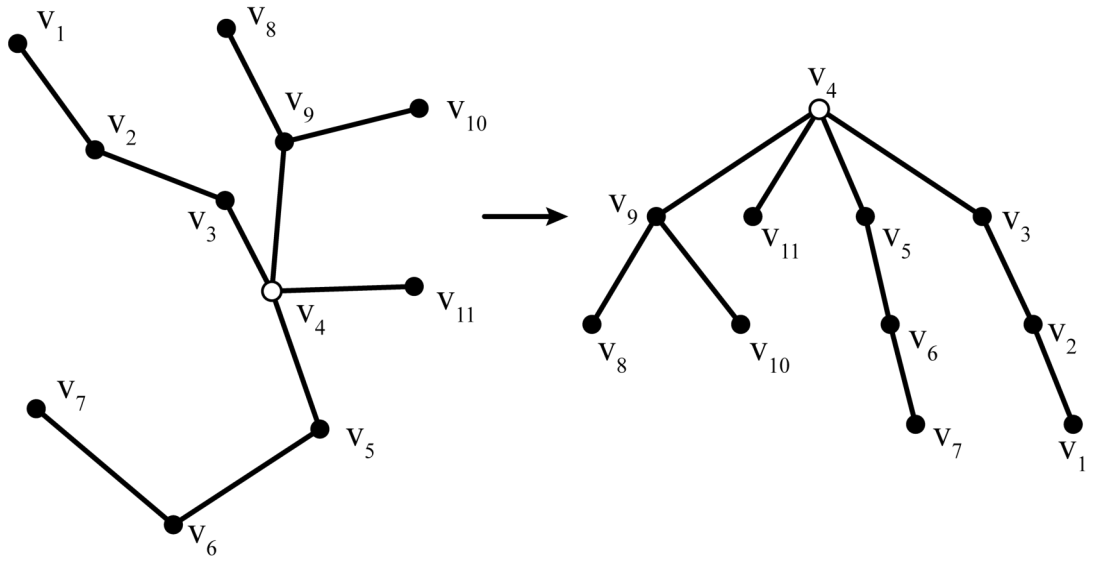


FIGURE 2.14: Order induced by the root on a rooted tree.

Using this order, we will define that a vertex u is **father** of a vertex v if u and v are adjacent and u is nearer to the root than v. In this case we will also say that v is **child** of u. If two vertices v and w have the same father, those vertices are called **siblings**. In Figure 2.14, v_3 is father of v_2, v_6 is child of v_5, and v_8, v_{10} are siblings since they have the same father v_9.

A **subgraph** H of a graph G is a graph such that $V(H) \subset V(G)$ and $E(H) \subset E(G)$. It is denoted as $H \subset G$.

The **barycentric subdivision** of G, represented as $sd(G)$, is the graph induced by G by putting an extra vertex on the medium point of every edge of G.

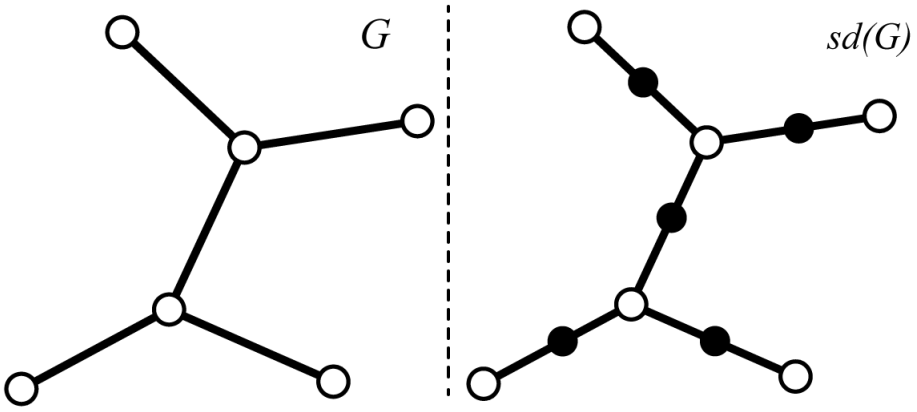


Figure 2.15: A graph G and sd(G)

Chapter 3

Trivalent 2-stratifolds with trivial fundamental group

Our objects of study are the trivalent 2-stratifolds with trivial fundamental group. As we can recall the definition given on Chapter 2 is:

Definition 3.1. A **trivalent 2-stratifold with trivial fundamental group** is a compact, connected Hausdorff space X together with a filtration $\emptyset = X_0 \subset X_1 \subset X_2 = X$ by a closed subspace such that X_1 is a closed 1-manifold, each point $x \in X_1$ has a neighborhood homeomorphic to $\mathbb{R} \times CL$, where CL is the open cone on L for some (finite) set L of cardinality 3 and each $x \in X_2 \backslash X_1$ has a neighborhood homeomorphic to $\mathbb{R}^2$.

On this chapter we are going to discuss how to build the trivalent 2-stratifolds with trivial fundamental group and how to have a useful representation of them to work with. Most of this work was published by J. C. Gómez-Larrañaga, F. González-Acuña and W. Heil.

3.1 Trivalent 2-stratifolds models

We know how the 2-manifolds look like, but in the case of 2-stratifolds there are few images of them. We are going to show some pictures of the simpliest 2-stratifolds but most of them can not be embedded in $\mathbb{R}^3$ so we will need another represention for them distinct from the picture itself.

One of the simplest trivalent 2-stratifolds is the sphere S^2 with a disk D in the interior such that the boundary of the disk belongs to the sphere, $\partial D \subset S^2$. This stratifold will be homeomorphic to the following stratifold:

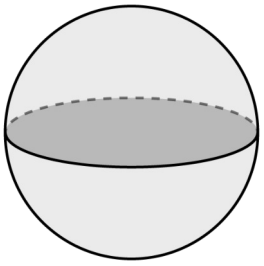


FIGURE 3.1: Trivalent 2-stratifold called B111-stratifold

We will refer to each element of the family of all the 2-stratifolds homeomorphic to Figure 3.1 as a B111-stratifold.

On the other hand, let D_1, D_2 be two disks and S^1 the circle, one representation of the B12-stratifold is the result of identify ∂D_1 with S^1 using the identity map and ∂D_2 with S^1 so the boundary of D_2 loops around S^1 twice, as in Figure 3.2.

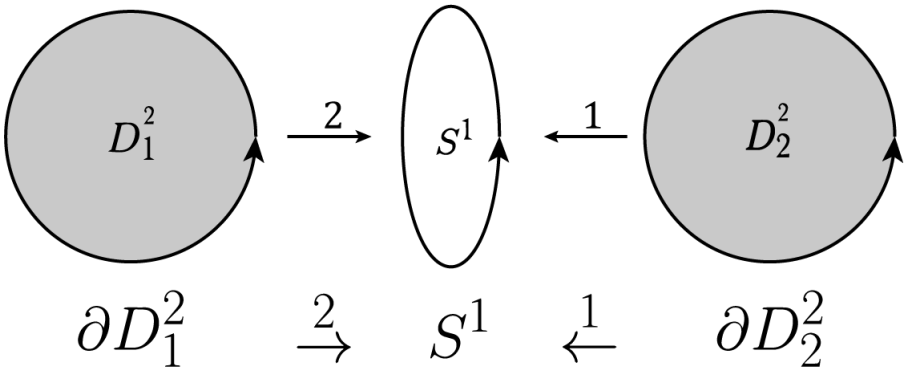


FIGURE 3.2: Trivalent 2-stratifold called B12-stratifold.

We will refer to each element of the family of all the 2-stratifolds homeomorphic to Figure 3.2 as a B12-stratifold.

Now we have a representation of the two basic trivalent simply connected 2-stratifolds. But we are interested in building all the other ones.

3.2 Relationship between trivalent 2-stratifolds with graph theory.

The space X_1 defined in Definition 3.1 is called the **1-skeleton** of X. This 1-skeleton contains a collection of many disjoint simple closed curves. On the other hand, $X - X_1$ is a collection of many disjoint closed 2-manifolds without boundary.

Considered a 2-stratifold (X, X_1) there exists an associated bipartite graph $G_X = G(X, X_1)$ embedded in X as defined in [11].

This graph is built as follows: for each component B_j of X_1, let $N(B_j) \subset (X, X_1)$ be a regular neighborhood of B_j and create a black vertex b_j. For each component W_i of $M = X - X_1$ create a white vertex w_i. Create an edge that joins a white vertex w_i with a black vertex b_j, for every disjoint component of $S = W_i \cap N(B_j)$. Note that the number of boundary components of W_i is the number of adjacent edges of w_i.

In the general case, we label the graph G_X by assigning to the white vertices the genus of their corresponding 2-manifolds (here we use Neumann's [12] convention of assigning negative genus to nonorientable surfaces) and by labeling the edge that joins w_i with b_j by k, where k is the degree of the covering map $\phi|_{\partial W_i} : \partial W_i \to B_j \subset X_1$.

A **covering map**[13] is a surjective open function $f : X \to Y$ such that there exists a discrete space D and for every $y \in Y$ an open neighborhood $U \subset Y$, such that $f^{-1}(U) = \bigsqcup_{d \in D} V_d \subset X$ and $f\,|_{V_d}$: $V_d \to U$ is a homeomorphism for every $d \in D$.

An example of the construction of this graph for a general 2-stratifold is the following:

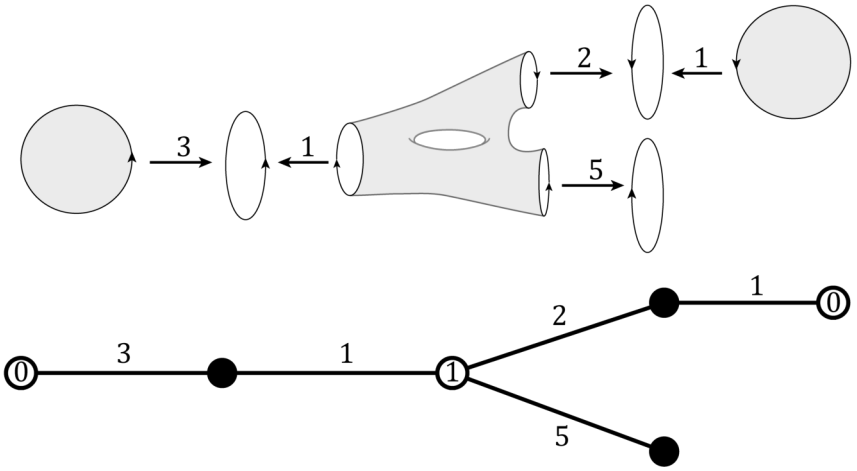


FIGURE 3.3: Example of the graph G_X given X a 2-stratifold

Proposition 3.2. *[7] If X is a 2-stratifold with trivial fundamental group, then G_X is a tree, all white vertices of G have genus 0 and all terminal vertices are white.*

As a consequence of this proposition, in the case of the trivalent 2-stratifols homotopy equivalent to S^2, every component W of $X - X_1$ is going to have genus equal to zero, therefore we are going to omit the step of labeling the white vertices since all of them are going to have the label 0. Also, for every black vertex of the associated graph, the sum of the weights of the incident edges is always going to be 3, since the 2-stratifold is trivalent. There are only two options, the black vertex is either incident to two edges where one has label 1 and one has label 2, or three edges, each of label 1.

The graphs corresponding to the B111 and B12 2-stratifolds would be the ones in Figures 3.4 and 3.5, respectively.

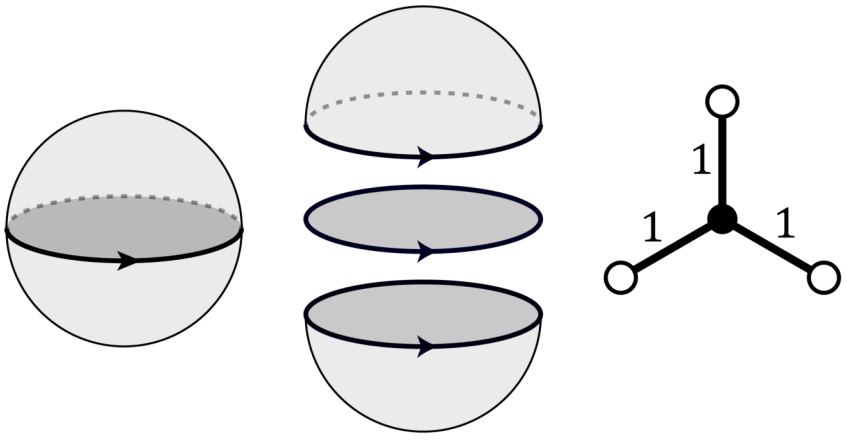


FIGURE 3.4: Decomposition of the B111 2-stratifold and its graph

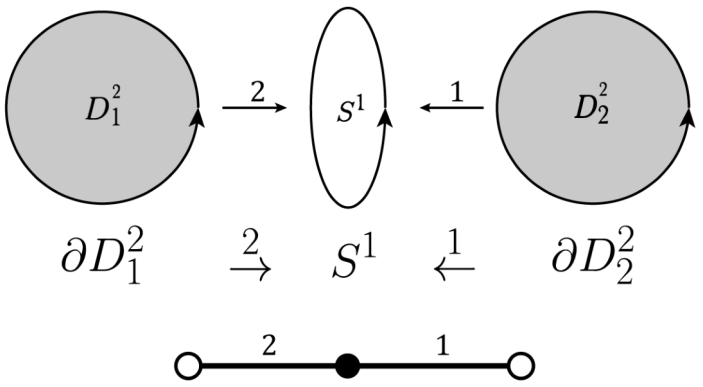


FIGURE 3.5: Graph corresponding to B12 2-stratifold

The previous graphs are the **B111-tree** and **B12-tree** formally defined as

Definition 3.3. [14]

1. The **B111-tree** is the bipartite tree consisting of one black vertex incident to three edges each of label 1 and three terminal white vertices each of genus 0.
2. The **B12-tree** is the bipartite tree consisting of one black vertex incident to two edges one of label 1, the other of label 2 and two terminal white vertices each of genus 0.

As stated in [11] if the graph associated to a 2-stratifold is a tree, then the labeled graph determines X uniquely. Given Proposition 3.2 we have that the function described before that builds the graph G_X is inyective in our case of study. Therefore the labeled graph G_X is a characterization of X, for X a trivalent 2-stratifold with trivial fundamental group.

3.3 Operations to build all the simply connected trivalent 2-stratifolds

We have now the first part, given a trivalent 2-stratifold with trivial fundamental group, we have described an algorithm that builds the corresponding graph associated to that 2-stratifold.

Proposition 3.2 has a stronger version proved in [14].

Theorem 3.4. *[14] Let X be a trivalent connected 2-stratifold and the graph G_X. The following are equivalent:*

1. *X is simply connected.*
2. *G_X is a tree with all white vertices of genus 0 and all terminal vertices white such that the components of $G_X - st(B)$ are $(2,1)$-collapsible trees and the reduced graph $R(G_X)$ contains no horned tree.*

For the better understanding of this theorem is necessary to recall some definitions from [15]. First, B denotes the union of all the black vertices of degree 3 of G_X and $st(B)$ is the (open) star of B in G_X.

A $(2,1)$-**collapsible tree** is a bipartite tree constructed as follows:
Start with a rooted tree T (which may consist of only one vertex), color with white and label 0 the vertices of T, take the barycentric subdivision $sd(T)$ of T, color with black the new vertices (the barycenters of the edges of T) and finally label an edge e of $sd(T)$ with 2 (resp. 1) if the distance from e to the root r is even (resp. odd). We allow a one-vertex tree (with white vertex) as a $(2,1)$-collapsible tree.

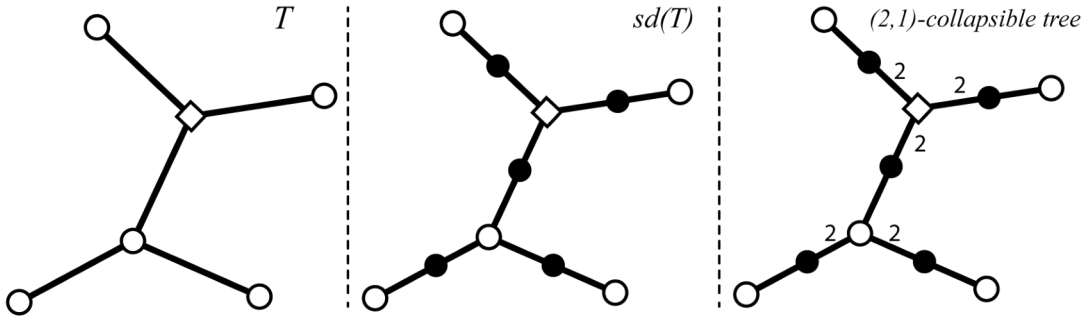


FIGURE 3.6: Construction of a $(2,1)$-collapsible tree, where the squared vertex is the root.

The **reduced subgraph** $R(G_X)$ is defined for a bipartite labeled tree G_X for which the components of $G_X - st(B)$ are $(2,1)$-collapsible trees. It is the graph obtained from $St(B)$(the closed star of B) by attaching to each white vertex w of $St(B)$ that is not a root, a B12-tree as the one in Figure 3.5, such that the terminal edge has label 2.

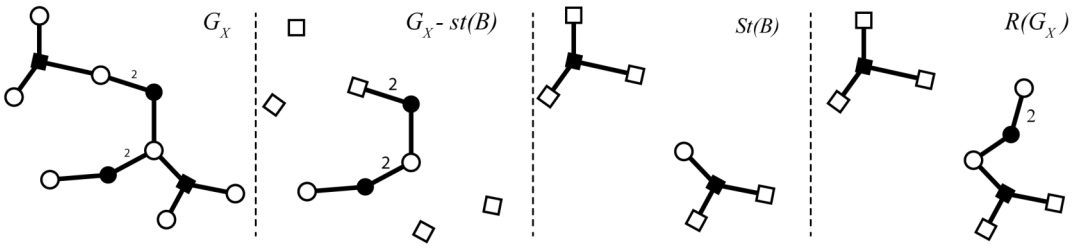


FIGURE 3.7: Construction of a reduced subgraph, where the squared black vertices are the set B and the squared white vertices are the roots of the $(2, 1)$-collapsible trees of $G_X - st(B)$.

And last one, a **horned tree** is a bipartite tree constructed as follows:
Start with a tree T that has at least two edges and all of whose not terminal vertices have degree 3. Color a vertex of T white (resp. black) if it has degree 1 (resp.3). Trisect the terminal edges of T and bisect the not terminal edges, obtaining the graph H_T. Color the additional vertices v so that H_T is bipartite, that is, v is colored black if v is a neighbor of a terminal vertex of H_T and white otherwise. Then label the edges such that every terminal edge has label 2, every not terminal edge has label 1.

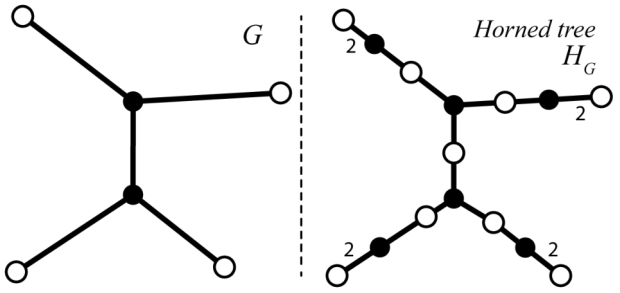


FIGURE 3.8: Construction of a horned tree

Now we can notice that not all the bipartite trees with all white vertices of genus 0, all terminal vertices white and edges labeled with 1 or 2 come from a simply-connected trivalent 2-stratifold.

Definition 3.5. We will say that a weighted graph G is a **trivalent-stratifold graph** if it comes from a simply-connected trivalent 2-stratifold X.

In [15], it is proven that with a recursive algorithm that consists of only three operations one can build every trivalent-stratifold graph. At the end of this section we are going to enunciate that result but first we need to describe the graph operations that are going to be needed.

Every operation is described in terms of how it modifies the trivalent-stratifold graph, but it has the corresponding action in terms of how it affects the trivalent 2-stratifold associated to the graph.

Consider G a bipartite trivalent-stratifold graph, let w be a white vertex and $r_1, r_2, ..., r_n$ be the edges incident to w ($n \geq 0$) and let b_i the black vertex incident to r_i ($1 \leq i \leq n$).

Definition 3.6 (Operation $O1$)**.** Let $0 \leq k \leq n$. Attach one white vertex of a B111-tree to w, cut off $b_{k+1}, b_{k+2}, ..., b_n$ from w and attach them to another white vertex of the B111-tree.

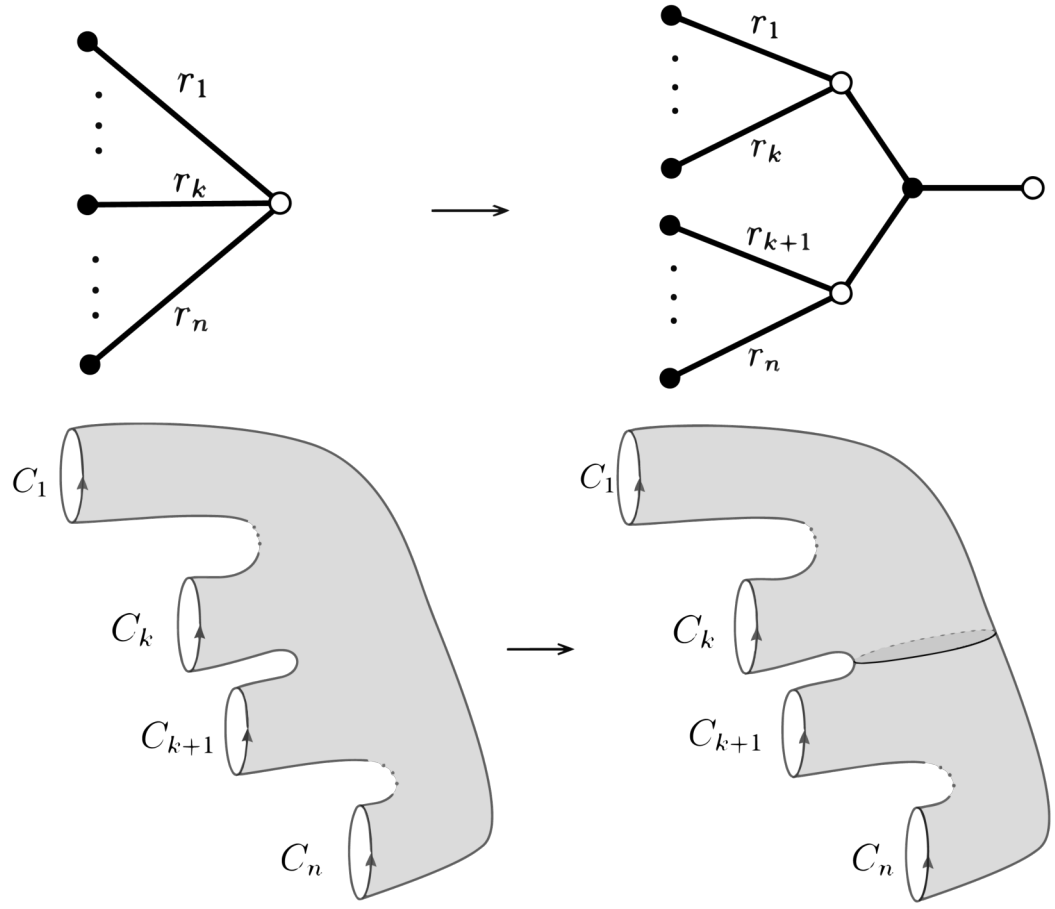


FIGURE 3.9: Operation $O1$ with $k \neq n$

As pictured in Figure 3.9, consider the 2-stratifold X we can consider the 2-manifold corresponding to w in X and a closed curve in the interior of that surface, call it C, that separates the curves $C_1, C_2, ..., C_k$ from $C_{k+1}, C_{k+2}, ..., C_n$ corresponding to $b_1, b_2, ..., b_k$ and $b_{k+1}, ..., b_n$ respectively. Applying the operation $O1$ consists on attaching a disk to X with its boundary equal to C.

One can simply choose $k = n$, then the operation $O1$ looks like simply attaching a B111-tree to one white vertex of w. We will refer to this case as **apply operation** $O1$ **on** w. In case that we choose $k \neq n$ we will specify that operation as **operation** $O1$ **with** k **on** w

This operation is equivalent to attaching a dome to a component of $X - X_1$ along a closed curve. As in Figure 3.10.

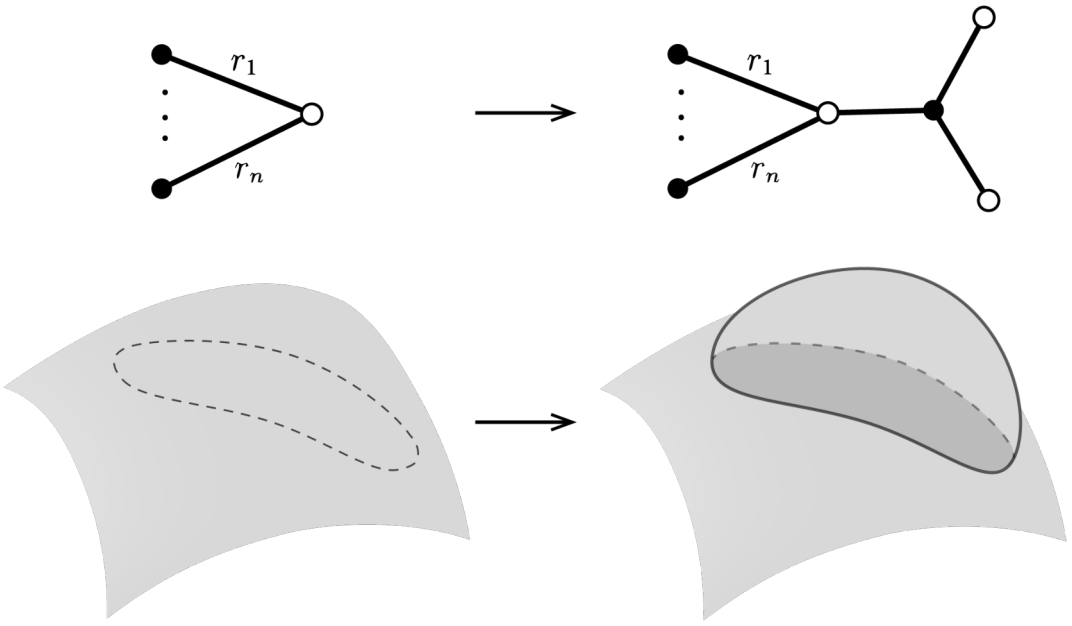


FIGURE 3.10: Operation $O1$ simple

It is easy to prove that if X is simply connected, the operation $O1$ does not change the fundamental group. And as a consequence we have the next theorem.

Theorem 3.7. *[15] Let X be a trivalent 2-stratifold such that each edge of G_X has label 1. Then the following statements are equivalent:*

1. *X is simply connected.*
2. *G_X is a tree (containing at least one black vertex) with all white vertices of label 0 and all terminal vertices white.*
3. *G_X can be constructed from the B111-tree by successively performing operation $O1$.*

Now we can proceed to define operation $O2$. Using the same terminology as before.

Definition 3.8 (Operation $O2$)**.** Attach a B12-tree to a white vertex w so that the terminal edge has label 1.

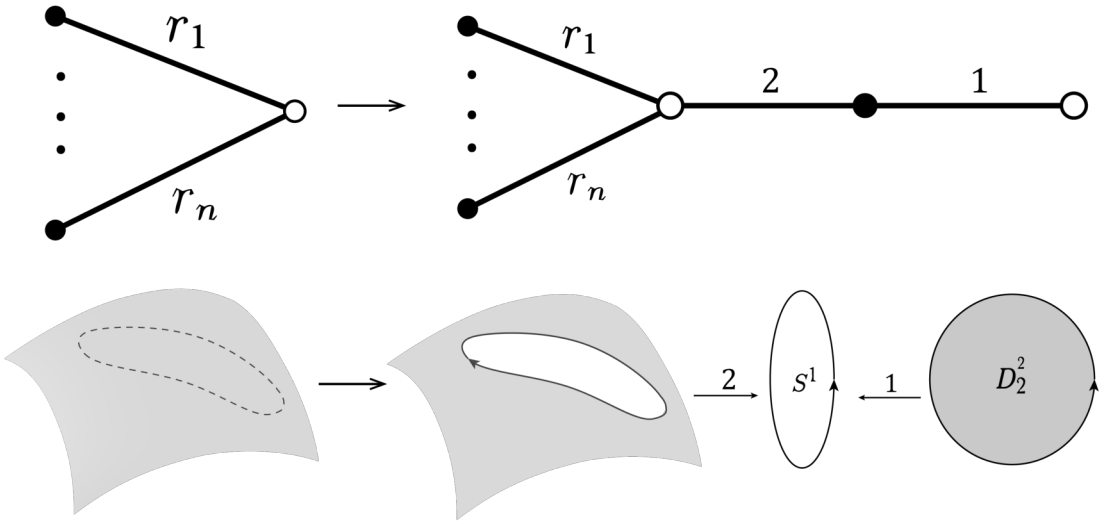


FIGURE 3.11: Operation $O2$

As in the case of operation $O1$, the operation $O2$ does not change the fundamental group of the 2-stratifold.

The last operation, operation 01^*, allows us to merge two different 2-stratifolds on one. Let G_1 and G_2 be two disjoint trivalent-stratifold graphs and w_1, w_2 be white vertices of G_1 and G_2, respectively.

Definition 3.9 (Operation $O1^*$)**.** Attach a B111-tree to $G_1 \cup G_2$ so that w_1 and w_2 are identified with two distinct white vertices of the B111-tree.

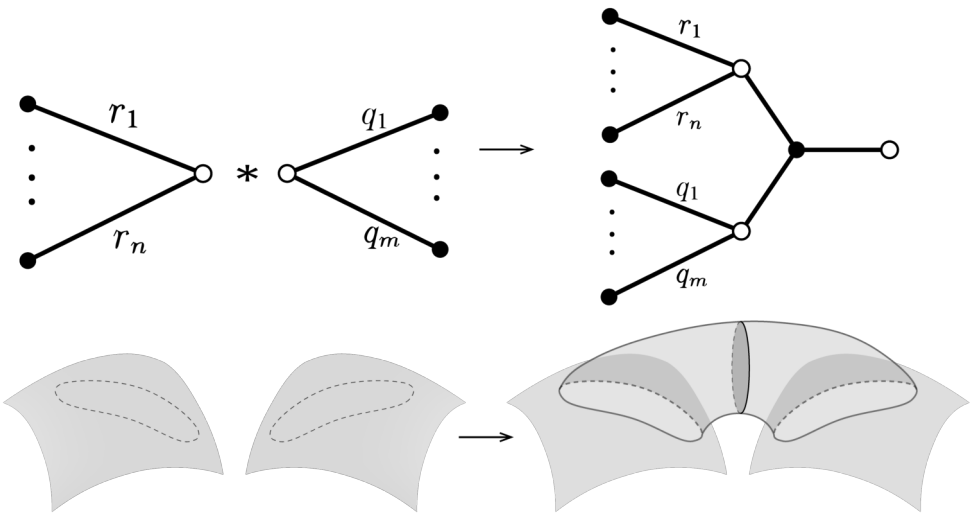


FIGURE 3.12: Operation $O1^*$

For the associated spaces X_1, X_2, the operation $O1^*$ identifies a whole disk on the interior of X_1 with a whole disk in the interior of X_2. Therefore, this operation does change the fundamental group and the resulting 2-stratifold has fundamental group isomorphic to $\pi_1(X_1) * \pi_1(X_2)$. If both spaces X_1, X_2 are simply connected, the resulting space would be simply connected as well.

By Theorem 3.4, and how the operations $O1, O2$ and $O1^*$ are defined, we can assure that the resulting graph of applying any of this operations to a trivalent-stratifold graph is a trivalent-stratifold graph. This is because none of the operations changes the fundamental group and adding B111 or B12-trees maintains the conditions on the second part of the theorem.

Chapter 4

The collection of trivalent-stratifold graphs

On the previous chapter we have described how to model every simply connected trivalent 2-stratifold as a graph and also how to get more from one that we already know is a trivalent-stratifold graph.

The next question is: Is that enough? On this chapter we are going to show the sufficient and necessary conditions to build all the trivalent-stratifold graphs. And we are going to talk about some of their properties.

4.1 The collection $\mathcal{G}$

Definition 4.1. We will define $\mathcal{G}$ as the collection of all the graphs that can be obtained from a single white vertex by successively applying operations $O1, O2$ and $O1^*$.

Since applying operation $O1$ and $O2$ to a white vertex gives us the B111-tree and B12-tree and it was stated before that the operations do not change the fundamental group, all the elements of $\mathcal{G}$ are trivalent-stratifold graphs.

Theorem 4.2. *[15] Let X be a trivalent 2-stratifold. Then X is simply connected if and only if $G_X \in \mathcal{G}$.*

This result was first proven in [15], but here we are going to follow a slightly different proof. Nevertheless, we will define some notation first.

Given a trivalent-stratifold graph Γ we will denote as $O1(\Gamma)$ (resp. $O2(\Gamma)$) to the set of all trivalent-stratifold graphs obtained by applying the operation $O1$ (resp. $O2$) to any white vertex of Γ.

Let A be a set of trivalent-stratifold graphs, we will denote

$$O1(A) = \bigcup_{\Gamma \in A} O1(\Gamma) \qquad \text{and} \qquad O2(A) = \bigcup_{\Gamma \in A} O2(\Gamma)$$

On the other hand, for two trivalent-stratifold graphs Γ_1 and Γ_2 we will denote $\Gamma_1 * \Gamma_2$ as the set of all trivalent-stratifold graphs resulted by choosing every pair of white vertices conformed by a vertex $u_1 \in \Gamma_1$ and a vertex $u_2 \in \Gamma_2$ and applying operation $O1^*$ to Γ_1 and Γ_2 on u_1, u_2.

Consider A_1 and A_2 two set of trivalent-stratifold graphs, we will write

$$A_1 * A_2 = \{\Gamma_1 * \Gamma_2 : \Gamma_1 \in A_1 \quad \text{and} \quad \Gamma_2 \in A_2\}$$

Since all the operations $O1, O2$ and $O1^*$ are performed on white vertices, it is natural to think in the number of white vertices of a trivalent-stratifold graph as a way of classification. So we are going to analyze how the operations affect the number of white vertices of the graphs.

Remark 4.3. For Γ a trivalent-stratifold graph, with k white vertices:

1. Applying operation $O1$ to any white vertex of Γ results in a trivalent-stratifold graph with $k + 2$ white vertices.
2. Applying operation $O2$ to any white vertex of Γ results in a trivalent-stratifold graph with $k + 1$ white vertices.

In the case of operation $O1^*$:

Remark 4.4. Consider Γ_1 and Γ_2 two trivalent-stratifold graphs, with k_1 and k_2 white vertices, respectively. The resulting graph of applying operation $O1^*$ to any pair of white vertices u, v of Γ_1, Γ_2, would have $k_1 + k_2 + 1$ white vertices.

An advantage of using the number of white vertices instead of the number of black vertices is the fact that for the tree operations the number of black vertices increases on 1, but the increase on number of white vertices is different depending on the operation.

As a result of the remarks one can notice, that use any of the previous operations increase the number of white vertices of the original graph.

Starting with one white vertex, the least increase possible is one white vertex, using operation $O2$. Therefore the simplest trivalent-stratifold graph is the B12-tree that has 2 white vertices and is the only one with that number of white vertices.

Definition 4.5. Let $\mathcal{G}_i$ be the set of all the trivalent-stratifold graphs with i white vertices, for $i \geq 2$. We will define $\mathcal{G}_1 = \emptyset$.

By the previous remark $\mathcal{G}_2 = \{B12 - tree\}$. Starting with one white vertex, using operation $O1$, we can increase the number of white vertices by 2, obtaining the B111-tree, also we can use operation $O2$ on each of the white vertices of the B12-tree to obtain another 2 different graphs with 3 white vertices. Therefore $\mathcal{G}_3 = \{B111 - tree, O2(\mathcal{G}_2)\}$.

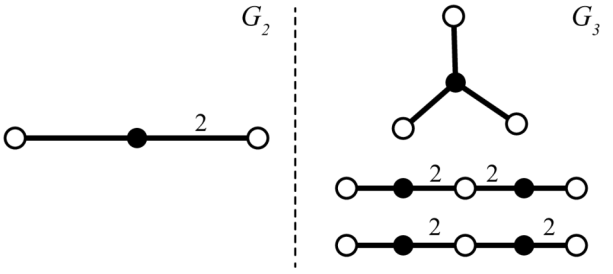


FIGURE 4.1: $\mathcal{G}_2$ and $\mathcal{G}_3$

Now we have all the tools to build the collection $\mathcal{G}$ of all the trivalent-stratifold graphs.

Theorem 4.6. *[16] For any n an integer greater than 3.*

$$\mathcal{G}_n = O1(\mathcal{G}_{n-2}) \cup O2(\mathcal{G}_{n-1}) \cup \left\{ \bigcup_{m=2}^{n-3} \mathcal{G}_m * \mathcal{G}_{n-m-1} \right\} \tag{4.1}$$

And $\mathcal{G} = \cup_{n=2}^{\infty} \mathcal{G}_n$

Before giving the proof of this theorem we need to prove the following lemmas.

Lemma 4.7. *[16] Let G be a trivalent-stratifold graph, there exists at least one leaf w of G such that the weight of the incident edge is 1.*

Proof. First, notice that for B12 and B111-tree there exists w a leaf such that the weight of the adjacent edge to w is 1.

Let G be a trivalent-stratifold graph, if we perform $O1$ in one of its vertices, we are attaching a B111-tree by one of its white vertices, letting 2 white vertices incident to edges with weight 1 be leaves of G.

On the other hand, if we perform $O2$ in one of the vertices of G, we are attaching a B12-tree to it by the only white vertex whose incident edge weight is 2, leaving the white vertex incident to the edge with weight 1 as a leaf of G.

Finally, if we perform $O1^*$ on G and other graph, by definition we take a B111-tree and attach one white vertex to G, one to the other graph and the last one is free, which is the leaf whose incident edge has weight 1.

We have proven that applying any of the valid operations on a trivalent-stratifold graph leaves us with a graph such that it has a leaf w with incident edge of weight 1. Because of Theorem 4.2, given that every trivalent-stratifold is the result of applying successive operations on a white vertex then the lemma is true for any trivalent-stratifold on $\mathcal{G}$. □

Lemma 4.8. *Given any trivalent-stratifold graph, the operation of erasing any black vertex with its incident edges results on two or three disconnected components that are either white vertices or a trivalent-stratifold graph.*

Proof. Let Γ be a trivalent-stratifold graph. By Theorem 4.2, Γ can be obtained from applying a succession of operations O_1, O_2, O_1^* to a white vertex. We can notice that each of this operations adds a unique black vertex to the previous graph. And that previous graph was part of $\mathcal{G}$, because it was the result of applying the valid operations, therefore it was a trivalent-stratifold graph.

And since the following operations are performed on white vertices, it's equivalent to performing them on a single white vertex. Therefore when you erase the black vertex that is connecting them the disconnected components can be obtained as a succession of operations O_1, O_2, O_1^* to a white vertex or a single white vertex. □

Now we can proceed with the proof of the Theorem 4.6.

Proof. By definition $\mathcal{G} = \cup_{n=1}^{\infty}\mathcal{G}_n$. Let n be an integer greater than 3. Through the Remarks 4.3.1, 4.3.2 and 4.4 it is clear that

$$O1(\mathcal{G}_{n-2}) \cup O2(\mathcal{G}_{n-1}) \cup \left\{ \bigcup_{m=2}^{n-3} \mathcal{G}_m * \mathcal{G}_{n-m-1} \right\} \subset \mathcal{G}_n$$

Now we will prove the reverse contention.

Let G be a trivalent-stratifold graph with $n > 3$ white vertices and w a leaf of G such that the edge incident to it has weight 1, it exists by Lemma 4.7 and it is white. Let b the black vertex adjacent to w. If b has degree 2, let v be the other vertex adjacent to b, when we erase the vertices w, b we get a new graph G' with $n-1$ white vertices such that after performing $O2$ on G' in the vertex v we get G and due to the number of vertices of G' we have $G' \in \mathcal{G}_{n-1}$.

If b has degree 3, then b is part of a B111-subtree, let v_1 and v_2 the vertices adjacent to b different from w. Without loss of generality, suppose that v_1 has degree 1, when we erase the vertices w, b, v_1, we get a new trivalent-stratifold graph G' with $n-2$ white vertices such that after performing operation $O1$ on v_2 the result is G and due to the number of vertices of G' we have $G' \in \mathcal{G}_{n-2}$.

On the other hand if neither v_1 nor v_2 have degree 1, when we erase the vertices b and w we get two trivalent-stratifold graphs H and H' such that the sum of their white vertices is $n-1$. We can assume that $v_1 \in H$ and $v_2 \in H'$. The result of applying $H * H'$ on v_1 and v_2 is the graph G. Since $H, H' \in \mathcal{G}$ we have that $H \in \mathcal{G}_j$, $H' \in \mathcal{G}_k$ for some j, k such that $j + k = n - 1$. Since all the trivalent-stratifold graphs are trees we can assure that H and H' are disjoint.

Since these are all the cases we can conclude that the equality is true. And the set $\mathcal{G}_n$ can be constructed as stated on the Equation 6.1. □

4.2 Graph isomorphisms

On the case of trivalent-stratifold graphs with 2 or 3 white vertices there is no ambiguity on how to obtain them. But when you want to have four or more white vertices there are more than one way to obtain some graphs. For example the result of applying operation $O2$ on any white vertex of the B111-tree and the result of applying

operation $O1$ on the white vertex incident to the edge with weight 2 of the B12-tree result in the same graph.

We have not defined what "the same graph" means, but intuitively it means that the graphs look the same. They have the same number of vertices and edges and the edges have the same weight. The importance of detecting when two graphs are the same lies on the fact that two graphs are the same if and only if they come from the same trivalent 2-stratifold. Therefore if we want to classify the trivalent 2-stratifolds is important to identify and eliminate repetitions.

Now we are going to properly define what "the same graph" means, formally we will describe them as isomorphic graphs.

Definition 4.9. Two weighted trees G and H are **isomorphic** if there exists a bijective function $\phi : V(G) \to V(H)$ such that two vertices u and v are adjacent in G if and only if $\phi(u)$ and $\phi(v)$ are adjacent in H. And for every edge, $u - v$ in G, the edge $\phi(u) - \phi(v)$ in H has the same weight as $u - v$. If there is no such function ϕ as described above, then G and H are **nonisomorphic trees**. (Definition from [9])

Definition 4.10. Two trivalent-stratifold graphs G and H are **isomorphic as trivalent-stratifold graphs** if there is an isomorphism ϕ as weighted graphs such that, if $B(G), B(H)$ are the set of black vertices of G, H and $W(G), W(H)$ are the sets of white vertices of G, H the functions $\phi|_{B(G)} : B(G) \to B(H)$ and $\phi|_{W(G)} : W(G) \to W(H)$ are bijective.

Lemma 4.11. *[16] Any two trivalent-stratifold graphs are isomorphic as trivalent-stratifold graphs if and only if they are isomorphic as weighted trees.*

The isomorphism of trivalent-stratifold graphs is really useful as an equivalence relation, because every trivalent-stratifold graph is uniquely associated with a 2-stratified space. Therefore if we can count and classify the trivalent-stratifold graphs up to isomorphism, we would have a complete classification for the 2-stratified spaces with trivial fundamental group which is the main goal.

But as we have said, for some graphs there are more than one way to build them using the valid operations. So we need an invariant that identifies the graphs and doesn't depends on the operations.

4.3 Trivalent-stratifold graphs as rooted trees.

In order of being able of identify differences between two graphs, we need to give them an order and on Section 2.3 we have mentioned that given a tree, selecting a vertex as its root induces an order to the graph.

We would like to apply this knowledge to the trivalent-stratifold graphs using the fact that they are trees. But the new query is: How to choose the root? For answering this question we will need the following definitions.

Definition 4.12. Let v be a vertex of a graph G, its **eccentricity** $e(v)$ is the length (without weights) of the largest path from v to another vertex in G. The **radius** of G, $rad(G)$ is the smallest eccentricity among the vertices of G. For any vertex v, such that $e(v) = rad(G)$ we say that u is the **center** of G. Finally the **diameter** of G, $diam(G)$, is the maximum eccentricity among the vertices of G.

Theorem 4.13 (Existence of the center)**.** *[16] Let G be a trivalent-stratifold graph, there always exists c center of G.*

Proof. Let v a vertex of G outside of a diametrical path of G, since G is a tree, it is true that $e(v) > diam(G)/2$, because if you consider the distance from v to the ends of the diameter, one of them must be longer than half of the diameter.

On the other hand if we look at the diameter of G, both ends of it must be leaves which implies that both of them are white. We know that G is bipartite, so the length of the diameter is even and it has a unique vertex in the middle. Such vertex would be the **center** of G. □

This result assure us that using the center of the trivalent-stratifold graph as its root avoids ambiguities. The importance of this lies on the fact that two isomorphic trees could be not isomorphic as rooted trees. For example Figure 4.2.

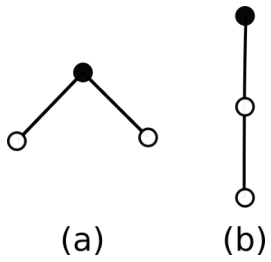


FIGURE 4.2: This is an example of two isomorphic trees that aren´t isomorphic as rooted trees. On each tree, we marked in bold black the root.

The next step is use the order induced by the root to identify differences between trivalent-stratifold graphs.

Chapter 5

String Representation

We have explained how trivalent 2-stratifolds with trivial fundamental group can be identified as rooted trivalent-stratifold graphs in a unique way. Now we will use this representation to identify repetitions on the process of generating all the trivalent-stratifold graphs. On this chapter we will focus on create a characterization that identifies isomorphic trivalent-stratifold graphs in a reasonable amount of time, computational speaking.

As explained on the previous chapter, generating the collection $\mathcal{G}$ is an iterative process that uses the sets $\mathcal{G}_n$. And generate a set $\mathcal{G}_n$ implies an exhaustive application of a specific operation to every white vertex of a previous set $\mathcal{G}_i$ (for $2 \leq i \leq n-1$). This process creates a lots of repetitions, some of them from the application of the same operation to different vertices, others from the application of two completely different succession of operations in the right vertices.

For example, the trivalent-stratifold graph in Figure 5.1 can be obtained in at least 4 different ways and it only has four white vertices.

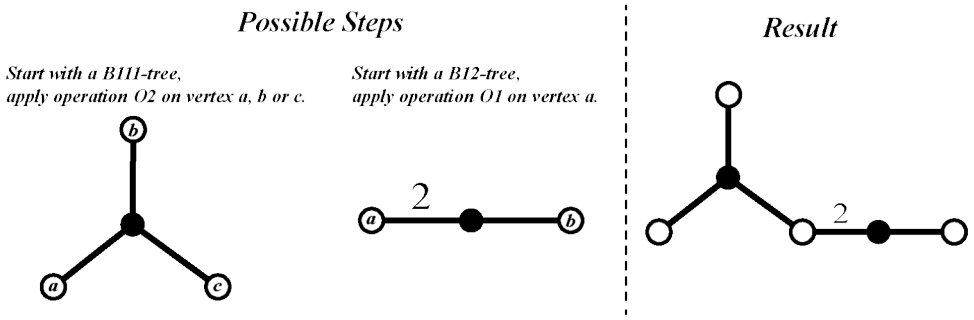


FIGURE 5.1: Operations generating isomorphic graphs.

5.1 The problem explained

We have defined a graph as a set of vertices together with a set of edges. In this case we are not naming neither the vertices nor the edges, because we are only interested on identifying the trivalent 2-stratifolds with trivial fundamental group without counting the isomorphisms.

On simple graphs, one can identify by eye if two of them are isomorphic or not. By the definition of *isomorphic trivalent-stratifold graphs* there are some observations that are useful to identify if two trivalent-stratifold graphs are not isomorphic. For example, the number of white vertices, the number of black vertices, the number of edges with weight 2 and weight 1, the length of the longest and shortest path, among other things. The issue is that two graphs can match on all this characteristics but don't be isomorphic. We can see an example in Figure 5.2.

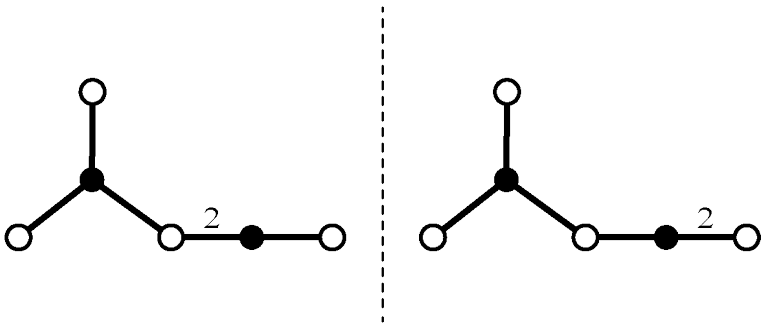


Figure 5.2: Two almost isomorphic graphs

When two trivalent-stratifold graphs match on all the previous characteristics but are not isomorphic, is normally because in one of the graphs at least one B12-subtree of the graphs has the weights of its edges on the inverse order from the other graph as seen in Figure 5.2. On big trivalent-stratifold graphs with many vertices, this subtle differences are difficult to identify, even having the graphic representation of the graph.

Now talking from a computational perspective, the computer does need to name the vertices and the edges. So it's not that simple to build the trivalent-straifold graph isomorphism as it is build it by hand. Because the first vertex of a graph can be equivalent to the third one of other graph, but without a renaming of the vertices, the computer can overlook this equivalence. That's why it is needed and algorithm that can build this isomorphism with lack of ambiguity, in case it exists.

5.2 The AHU algorithm

The AHU algorithm is a known algorithm that allows us to identify if two not weighted rooted trees with n vertices are isomorphic in $O(n)$ time. The time means that it's lineal only depending on the number of vertices of the trees, which is considered a *fast* algorithm. The AHU algorithm was proposed on the book *The Design and Analysis of Computer Algorithms* [17] and its named by the initials of its authors Aho, Hopcroft and Ullman.

The original algorithm presented by Aho, Hopcroft and Ullman is the Algorithm 1.

Algorithm 1 Original AHU Algorithm(T_1:rooted tree, T_2:rooted tree)

Given two rooted not weighted trees T_1, T_2;

1. Assign to all leaves of T_1 and T_2 the integer 0.
2. Inductively, assume that all vertices of T_1 and T_2 at level $i-1$ have been assigned integers. Assume L_1 is a list of the vertices of T at level $i-1$ sorted by not decreasing value of the assigned integers. Assume L_2 is the corresponding list T_2.
3. Assign to the not leaves of T_1 at level i a tuple of integers by scanning the list L_1 from left to right and performing the following actions: For each vertex v on list L_1 take the integer assigned to v to be the next component of the tuple associated with the father of v. On completion of this step, each not leaf w of T_1 at level i will have a tuple $(i_1, i_2, \ldots, i_k)$ associated with it, where $i_1, i_2, \ldots, i_k$ are the integers, in not decreasing order, associated with the sons of w. Let S_1 be the sequence of tuples created for the vertices of T_1 on level i.
4. Repeat the previous step for T_2 and let S_2 be the sequence of tuples created for the vertices of T_2 on level i.
5. Sort S_1, S_2 using lexicographical order. Let S_1' and S_2' be the sorted sequences of tuples.
6. If S_1' and S_2' are not identical, then halt; the trees are not isomorphic. Otherwise, assign the integer 1 to those vertices of T_1 on level i represented by the first distinct tuple on S_1', assign the integer 2 to the vertices represented by the second distinct tuple and so on. As these integers are assigned. Append to the front of L_1 all leaves of T_1 on level i. Let L_2 be the corresponding list of vertices of T_2. These two lists can now be used for the assignment of tuples to vertices at level $i+1$ by returning to step 3.

Here is an example of the application of this algorithm

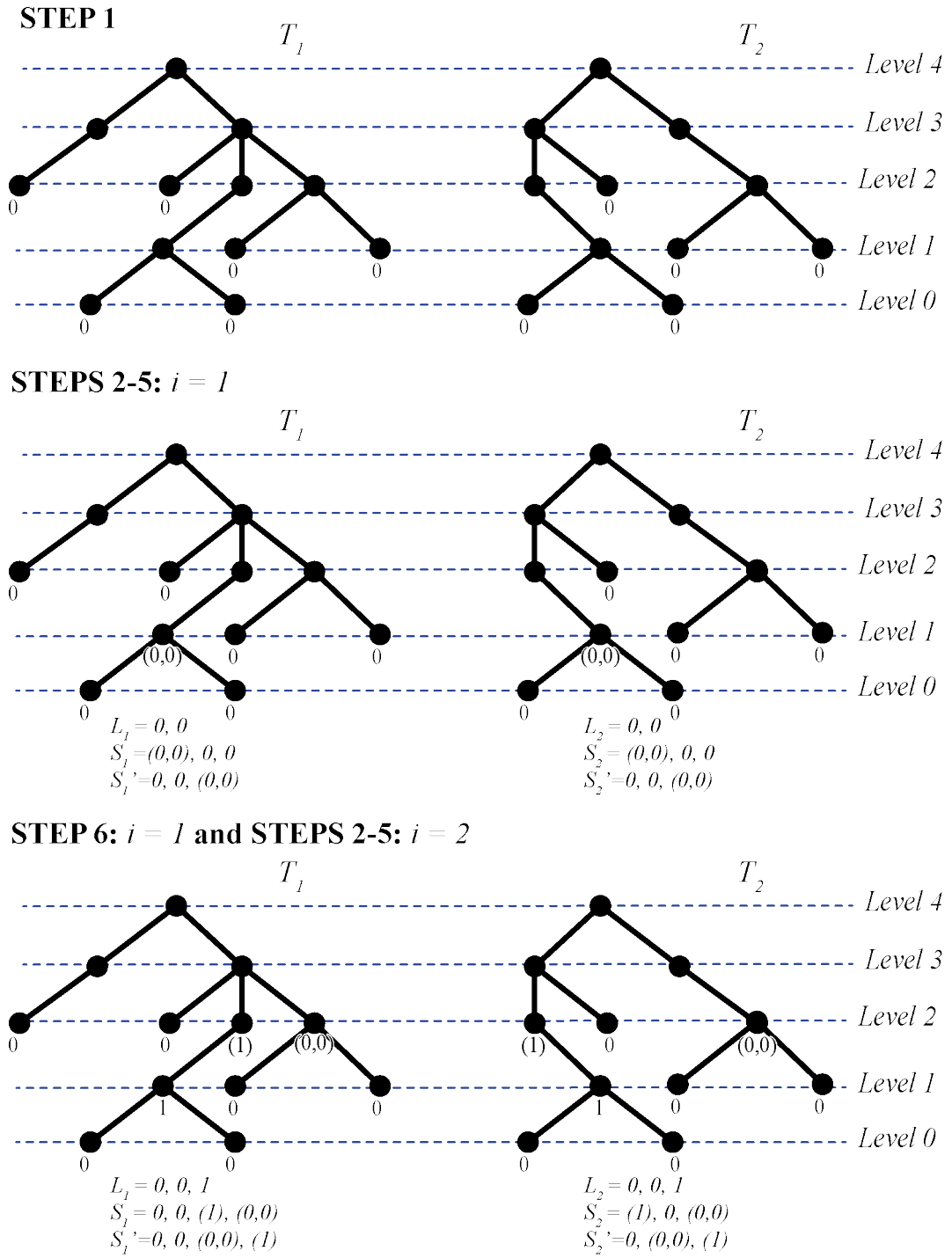


FIGURE 5.3: AHU Algorithm Applied

The great advantage of this algorithm is that for two trees it is really fast as I already mentioned. But for the problem that we are working on, it would be necessary to

compare every set of two trivalent-stratifold graphs with the same number of vertices in order to separate the 2-stratifolds that are not isomorphic. But also the algorithm doesn't give you a way to identify each graph after applying it.

An algorithm that uses the original idea of the authors Aho, Hopcroft and Ullman of comparing the trees by level is the assignation of **Knuth tuples**, also known as **parenthical tuples**, proposed by Donald E. Knuth on the volume 4, fascicle 4, of his serie *The Art of Computer Programming* [18]. He points out the relationship that exists between the rooted trees and the nested parenthesis. More over, he states an algorithm that allows to identify every rooted tree with a unique string of nested parenthesis.

The algorithm that assigns the Knuth tuples is the following:

Algorithm 2 Assigning Knuth Tuples(T:Rooted tree)

1. Set the tuple $()$ for every leaf of the tree.
2. Inductively assume that all vertices of T at level $i-1$ have a string of nested parenthesis assigned.
3. Given a vertex v at level i, not leave of T. Let L be the tuple of nested parenthesis of the vertices of T that are children of v at level $i-1$.
4. Let L' be the sorted sequence of the elements of L using lexicographical order.
5. The string of v is the tuple L' nested on parenthesis, as (L').
6. Return to step 2 until all the vertices have an assigned string.

Here is an example of the application of this algorithm:

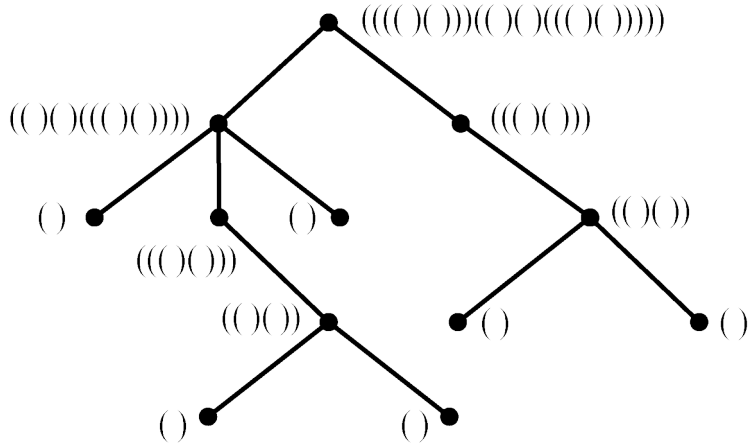


Figure 5.4: Assignation of Knuth Tuples

On 1991, Campell and Radford [19] suggested a modern version of this algorithm where they proposed to use the number "1" instead of the opening parenthesis "("; and the number "0" instead of the closing parenthesis ")". Generating a string of numbers that represents in a unique way each rooted not weighted tree. This is the algorithm commonly known as the **AHU algorithm**. Here is the pseudo-code on Algorithm 3 and an example can be seen in Figure 5.5

Algorithm 3 AHU(v: vertex)

```
if v is childless then
    Give v the tuple name "10"
return "10"
else
    Set L = ∅
    for all w child of v do
        tag = AHU(w);
        Append tag to L
    end for
    Sort L using binary order
    Set temp = Concatenation of tags in L
    Give v the tuple name "1temp0"
end if
```

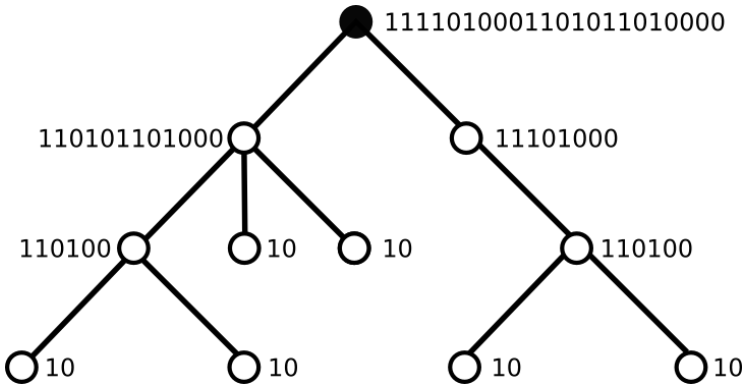


Figure 5.5: Example of labels given by running the AHU to a rooted tree

Based on this idea, we thought that this algorithm could be applied to our problem with a few modifications. Taking into account that we needed to identify the weights of the edges on the final string.

5.3 Characterizing weighted trees with a string

One fundamental property that is necessary to apply the AHU Algorithm is that the tree must have a root. This is important because this gives you an order of the tree. Because the root is the vertex at the top and then the other vertices are arranged by levels.

Since the trivalent-stratifold graphs can be seen as rooted trees, something important to say is that (except for the root) every vertex has a unique father, even when some fathers have more than one child. The AHU algorithm result is obtained from the bottom to the top (being the root the top of the tree), this is the same as saying that the algorithm is a result of going from child to father. During the application of the AHU Algorithm, given a vertex v, we will say that a change of level is when you pass from the children of v to v when applying the algorithm.

On the Algorithm 3 when assigning a tuple to a vertex, the process consist on taking the tuples of its children, sort them and write them between a 1 and a 0. In our proposal, instead of writing a 1 and a 0 for every vertex, one must check which is the weight of the edge that connects this vertex to its father. Since this edge is unique, there's no place to ambiguity.

The trivalent-stratifold graphs only have two type of edges, edges with weight 1 and with weight 2. Therefore we only need two different ways of represent the weights.

We decided that if the edge connecting the vertex to its father had weight 1 it would be represented with '01' and if it had weight 2 it would be represented with '23'. The pseudo-code of this modification is Algorithm 4.

Algorithm 4 AHU-modified(v:vertex)

```
if v is childless then
    if v has no father or Weight[v, father(v)] = 1 then
        Give v the tuple name "01";
    else
        Give v the tuple name "23";
    end if
return The tuple name of v
else
    Set L = ∅
    for all w child of v do
        tag = AHU-modified(w);
        Append tag to L
    end for
    Sort L using base four order
    Set temp = Concatenation of tags in L
    if v has no father or Weight[v, father(v)] = 1 then
        Give v the tuple name "0temp1";
    else
        Give v the tuple name "2temp3";
    end if
return The tuple name of v
end if
```

This algorithm stills run in $O(n)$ time, in other words, it is linear. And for every rooted trivalent-stratifold graph, when this algorithm is applied to the root it would return a string or tuple which uniquely identifies the graph.

On the previous chapter, we have proved that every trivalent-stratifold graph has a center and defining this center as the root would lead us to a unique representation of the trivalent-stratifold graph as a rooted tree.

Definition 5.1. Given a trivalent-stratifold graph G we call the output of applying Algorithm 4 to the center of G as the **string representation** of G.

Theorem 5.2. *Given two trivalent-stratifold graphs, they are isomorphic if and only if they have the same string representation.*

By Algorithm 4 we know that for every trivalent-stratifold graph there exists a unique string representation of it. We are only missing that given a string representation

it builds a unique trivalent-stratifold graph. This two parts will give us a bijection between the trivalent-stratifold graphs and the string representations. Let's notice that Algorithm 5 builds a trivalent-stratifold graph from a string representation.

Algorithm 5 String_to_TG(S: string, $father$: vertex)

```
if father is NONE then
    Draw a vertex v;
    State father as v;
end if
if The first element of S is 0 then
    Set Close as 1;
else
    Set Close as 3;
end if
Set i as 2
while The i-th element of S is different from Close do
    if The i-th element of S is 0 then
        Draw a vertex w connected to father with weight 1;
    else
        Draw a vertex w connected to father with weight 2;
    end if
    Set P as the string S without its first element;
    Set i = String_to_TG(P, w) + 2;
end while
return i;
```

Since the Algorithm 4 defines an injective function from the rooted trivalent-stratifold graphs to the string representations and the Algorithm 5 defines its inverse function, the Theorem 5.2 is proven.

Now we have a unique way to represent every trivalent-stratifold graph that the computer can easily recognize. This would be helpful during the implementation of the algorithm that builds all the trivalent-stratifold graphs because it would help us to erase the repetitions of graphs.

Both algorithms can be extended for n-colored trees in general, also it can be extended for trees with a greater amount of weights, it is only needed to add more labels to identify the different weights.

Chapter 6

Computational Implementation

We have so far explained the theory that is necessary to build and count all the trivalent-stratifold graphs without isomorphisms. On this chapter we will explain the pseudo-code that is necessary to implement this algorithm as a program.

Dr. Jesús Rodríguez Viorato and I implemented this program on Python. The complete code can be found on `https://github.com/MyHerket/TrivalentStratifoldsGraphs` [20]. This program was implemented using the classes created by Yair Hernández [21] as a base.

6.1 String Representation of a Trivalent-Stratifold Graph

On Chapter 4 we have proven that the center of every trivalent-stratifold graph exists. But we haven't proposed a way to obtain it. In [17] pages 176 to 179, Aho, Hopcroft and Ullman describe the algorithm *Depth-first search* whose purpose is to find the largest path in a tree. It is proven that the number of operations that this algorithm needs to finish is the maximum number between the number of vertices and the number of edges of the tree. This is helpful because that means that the algorithm is linear, just as the AHU Algorithm and its modifications.

The algorithm Depth-first search visits every vertex going deeper on each branch before continuing to another branch. Here is the pseudo-code of the algorithm presented by Aho, Hopcorft and Ullman.

Algorithm 6 Depth-first_search(v:vertex)

if v is childless **then return** v
else
  Set $length = 0$ and $longestPath = \emptyset$
  for all w child of v **do**
    Set $path =$`Depth-first_search(`w`)` and L as the length of $path$.
    if $L > length$ **then**
      Set $length = L$ and $longestPath = path$
    end if
  end for
return $v \cup longestPath$
end if

By applying the algorithm twice, one on any vertex and the second one on the beginning of the longest path found, we can assure that we will find the longest path of the tree, also defined as the diameter of the tree. Since the middle vertex of the diameter is the center of any trivalent-stratifold graph, this is an excellent way of finding the vertex that would be root of the trivalent-stratifold graph.

Therefore, the algorithm to find the center of a trivalent-stratifold graph is the Algorithm 7.

Algorithm 7 center(G: trivalent-stratifold graph)

Set v a vertex of G
Set $longestPath =$`Depth-first_search(`v`)`
Set w as the last vertex of the path $longestPath$.
Set $longestPath =$`Depth-first_search(`w`)`
Set $center$ as the middle vertex of $longestPath$
return $center$

To prove that we can find a diameter with two runs of DFS or Depth-first Search algorithm we proceed as follows. Given a tree G with center c, let v a vertex of G. Running the first DFS algorithm from v would give us the longest path from v to any other vertex, let w be the end of this path. It is known that w would be the end of a diameter of G. Then, running the second DFS algorithm from w would give us the longest path from w, which is also the diameter of G. This proof relies in the fact that for every pair of vertices in a tree, there exists a unique path that connects them. This property of uniqueness and existence lead us to the fact the after applying the DFS algorithm from any vertex, we would end in the end of a diameter.

The Algorithm 7 runs in twice the number of operations that the Algorithm 6 needs plus a constant. As we have said Algorithm 6 runs in linear time, meaning that Algorithm 7 is linear as well.

Now that we have found a fast way to find the center of any trivalent-stratifold graph and turn it into a rooted trivalent-stratifold graph. By applying Algorithm 4 to a rooted trivalent-stratifold graph, we can obtain its string representation that uniquely identifies the graph.

The complete pseudo-code of this sequence of steps is the Algorithm 8 whose output is the string representation of the graph and the Figure 6.1 is a visual example of applying this algorithm.

Algorithm 8 TG_to_string(G: trivalent-stratifold graph)

Set c as the output of `center(`G`)`;
Set c as the root of G.
AHU-`modified(`c`)`;
return the tuple name of c;

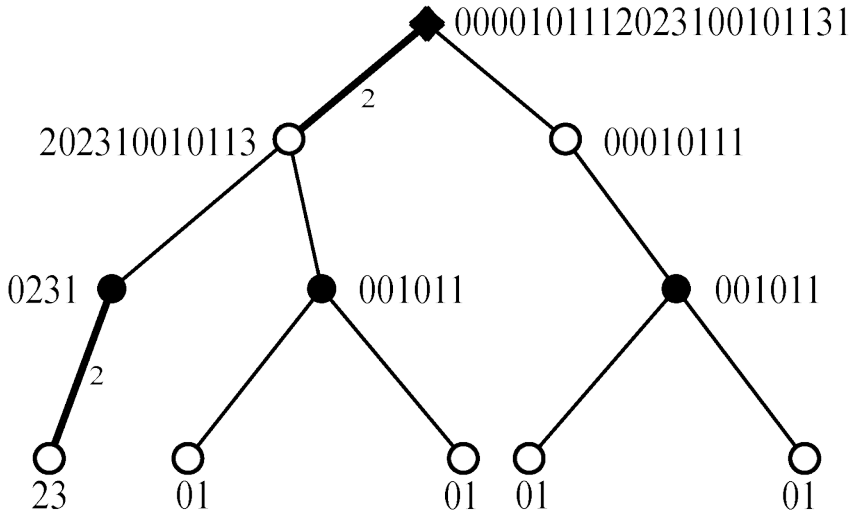


Figure 6.1: Generating the string representation given a trivalent-stratifold graph. The squared vertex is the center of the graph.

6.2 The collection $\mathcal{G}$

On Chapter 4 we have define the sequence of steps to build the collection $\mathcal{G}$ of all the trivalent-stratifold graphs. But computational speaking we need to build the collections $\mathcal{G}_m$ one by one. In other words, we can only build a finite amount of trivalent-stratifold graphs.

We need to recall some definitions from Chapter 4.

1. The **B111-tree** is the bipartite tree consisting of one black vertex incident to three edges each of label 1 and three terminal white vertices each of genus 0.
2. The **B12-tree** is the bipartite tree consisting of one black vertex incident to two edges one of label 1, the other of label 2 and two terminal white vertices each of genus 0.

The pseudo-code of the implementation of Theorem 4.6, which states how to build the collection $\mathcal{G}$, is the Algorithm 9.

Theorem 4.6 For any n an integer greater than 3.

$$\mathcal{G}_n = O1(\mathcal{G}_{n-2}) \cup O2(\mathcal{G}_{n-1}) \cup \left\{ \bigcup_{m=2}^{n-3} \mathcal{G}_m * \mathcal{G}_{n-m-1} \right\} \tag{6.1}$$

And $\mathcal{G} = \cup_{n=2}^{\infty} \mathcal{G}_n$

On Algorithm 9 there are some steps that are "add the graph to the list if the graph is not already there". In order to identify if the graph is on the list or not, we used an object class that allows us to save the string representation of the graph. Once we obtain the new graph that we wanted to add to the list, we ran the Algorithm 8 to get the string representation and then used a hash, function setting the string as the key, to optimize the identification of repeated graphs.

Algorithm 9 Construct_TG(m:integer)

Create $Complete\_list$ a list with $m-1$ empty lists;
Set the B12-tree as the first element of $Complete\_list[0]$;
Set the B111-tree as the first element of $Complete\_list[1]$;
for all w white vertex of B12-tree **do**
    Set Γ as the result of applying O2 to the B12-tree in the vertex w;
    Add Γ to $Complete\_list[1]$;
end for
for n in $[3, m]$ **do**
    for all g graph in $Complete\_list[n-2]$ **do**
        for all w white vertex of g **do**
            Set Γ as the result of applying O2 to g in the vertex w.
            Add Γ to $Complete\_list[n-1]$ if it's not already there.
        end for
    end for
    for all g graph in $Complete\_list[n-3]$ **do**
        for all w white vertex of g **do**
            Set Γ as the result of applying O1 to g in the vertex w.
            Add Γ to $Complete\_list[n-1]$ if it's not already there.
        end for
    end for
    for i in $[0, n-1]$ **do**
        if $n-i-4 \geq 0$ **then**
            for all pair (u, v) where u is a white vertex of a graph in $Complete\_list[i]$, v is a white vertex of a graph in $Complete\_list[n-i-4]$ **do**
                Set Γ as the output of applying $O1^*$ using the vertices u, v;
                Add Γ to $Complete\_list[n-1]$ if it's not already there.
            end for
        end if
    end for
end for

After running the program in Python for 11 white vertices we could count the number of graphs created and the number of graphs without isomorphism (labeled as total). The following table shows the results of this process. The numbers labeled as *created* is the total number of graphs created after applying the steps on Theorem 4.6.

n	Total	Created
2	1	1
3	3	3
4	6	11
5	18	37
6	51	150
7	167	573
8	551	2267
9	1954	8997
10	7066	36498
11	26486	149708

TABLE 6.1: Number of distinct graphs we got for each value n (the number of white vertices) and the number of graphs that were created to construct them all.

Even when the algorithms used on this process run in linear time, they depend on the number of vertices of the graphs. And as the number of white vertices grow, the number of black vertices grow as well. Also the number of graphs grow exponentially, that's why even when the algorithm is 'good and fast' we can't run it for a large amount of white vertices. The memory on the computer won't allow us.

6.3 More optimization

We have mentioned that there are some operations that lead us to the same graph. One example of this is figure 5.1 where the same operation on 3 different vertices of the same graph lead us to the same result.

This phenomenon occurs because of the autosymmetries of the graph that we are working on.

Definition 6.1. Given G a rooted trivalent-stratifold graph, we say that two vertices $u, v \in G$ are *symmetric* if there exists an automorphism $\phi : G \to G$ (as rooted weighted graphs) such that $\phi(u) = v$.

We can notice that given a rooted trivalent-stratifold graph, two vertices are symmetric if they have the same string representation. Because the string representation of any vertex depends solely on the descendants of it. Also it is necessary that the fathers of both vertices are symmetrical as well or that they are the same vertex. Both of this observations imply that the process of detecting symmetrical vertices can be iterated

recursively, being successful when the fathers of the two vertices coincide or having failed in other case.

We used this idea in order to detect symmetrical vertices while building the collection $\mathcal{G}$ to avoid the repetitions of applying the same operation to symmetrical vertices. And this reduced the number of generated graphs by around 20% as we can see on Table 6.2.

n	Created	Reduction
4	11	0,00%
5	32	13,51%
6	122	18,67%
7	467	18,50%
8	1781	21,44%
9	7099	21,10%
10	28852	20,95%
11	119168	20,40%

TABLE 6.2: Number of created graphs after considering symmetrically distinct white vertices.

One can try other ways of optimizing the algorithm but because the number of elements on the collections $\mathcal{G}_n$ grow exponentially the algorithm would still run in exponential time.

6.4 Search Engine for Trivalent-Stratifold Graphs

As a result of the implementation of the program on Python, Dr. Rodríguez-Viorato and I were able to draw every trivalent-stratifold graph with 10 or less white vertices. And uploaded the images on a search engine for future consultation.

This search engine is hosted at `http://trivalent-stratifolds.com` [22]. And has the following home screen:

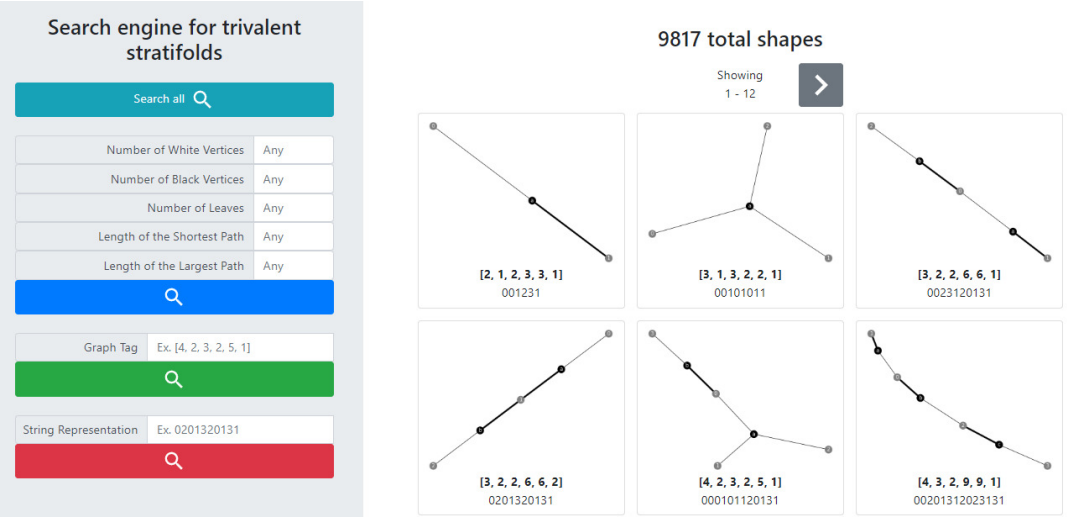


FIGURE 6.2: Home screen of the Search Engine for Trivalent-Stratifold Graphs

To have the general idea of the shape of the graph we conclude that the number of leaves, white and black vertices were important. Also, the length of the largest and shortest leaf paths of the graph. That's why include this information on the search engine.

On the search engine, we identify every graph with a unique tag, this tag is different from the string representation because for graphs with a great number of vertices this string can be really long and also it doesn't gives you a lot of information by itself without building the graph. On the other hand, the tag gives you the characteristics mentioned on the previous paragraph.

Given a graph G, denote $W(G), B(G), L(G)$ the sets of white vertices, black vertices and leaves of G, respectively.

The tag of G has the following structure:

$$tag(G) = [\ |W(G)|, |B(G)|, |L(G)|, \textit{length(shortest leaf path(G))}, \textit{length(largest leaf path(G))}, \textit{ID number(G)}]$$

where $|\cdot|$ is the size of the set $\cdot$.

On Chapter 5 Section 5.1, we mentioned that even when two graphs have the same values on the first five characteristics of the tag, the trivalent-stratifold graphs could be not isomorphic because of the distribution of weights on the edges. That's why we need the sixth value that is an ID number.

The ID number is assigned as follows. For every set

$$A = \{G : \begin{cases} G \text{ is a trivalent-stratifold graph,} \\ |W(G)| = a_1, \\ |B(G)| = a_2, \\ |L(G)| = a_3, \\ length(\text{shortest leaf path(G)}) = a_4 \text{ and} \\ length(\text{largest leaf path}(G)) = a_5 \end{cases}$$

where $a_i \in N$ for $1 \leq i \leq 5$. We will sort the elements of A using their string representation and the lexicographical order. Therefore, for two graphs $G_1, G_2 \in A$, $G_1 < G_2$ if the string representation of G_1 is minor than the string representation of G_2 lexicographically. Then the ID number of a graph is its position in the sorted set, where the first position corresponds to the smallest element of the set.

By this method, we can assure that the ID number has no ambiguities during the assignation and can be applied for greater sets of trivalent-stratifold graphs.

On the search engine, one can look for all the graphs that share a specific characteristic. And not only does it show you the elements, it also tells you the size of the set. For example, all the trivalent-stratifold graphs with 4 white vertices. The interface also tells you that there are only 6 graphs with this characteristic. As shown on figure 6.3

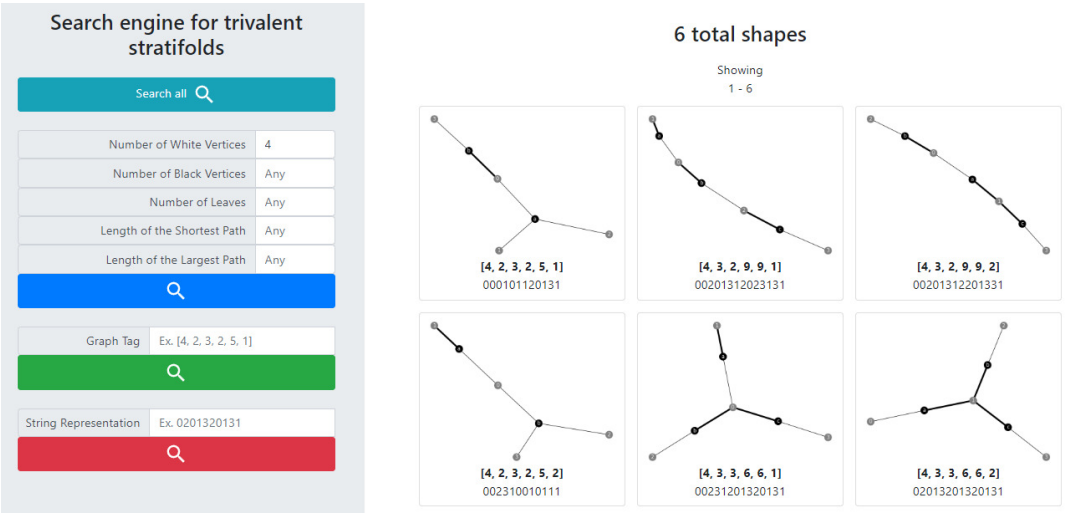


FIGURE 6.3: Example of use of the Search Engine

The search engine also allows the user to look for a trivalent-stratifold graph by its tag or by its string representation. And by selecting any image one can see all the information associated to that graph and even download the image.

FIGURE 6.4: Example of a trivalent-stratifold graph with its associated information.

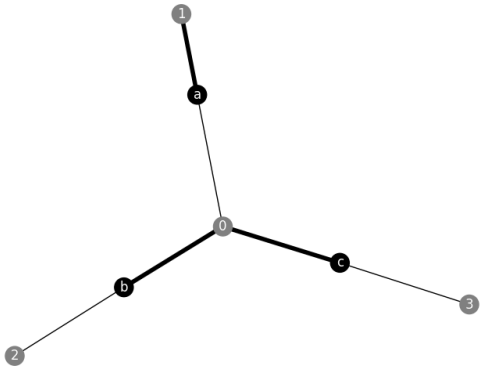


FIGURE 6.5: $[4, 3, 3, 6, 6, 1]$

We will now proceed to describe an image of a trivalent-stratifold graph showed on the search engine.
On the image we can see gray vertices representing the white vertices and black vertices representing the black vertices of the graph. The bold edges are the edges with weight 2 and the other ones have weight 1.

The number on the gray vertices represents the order in which this vertices were added to the graph. The analogous occurs with the letters on the black vertices. Remember that every graph is obtained as the result of an iterative process. And with this numbers and letters one can retrieve one sequence of operations which output is the graph in the screen.

Chapter 7

Conclusions & Future Work

By the implementation of the *String Representation* to identify the trivalent-stratifold graphs we can assure that this work has a satisfactory conclusion.

As stated on Chapter 1 our main goal was to be able to classify the trivalent 2-stratifolds with trivial fundamental group. By the identification of every trivalent 2-stratifold with a trivalent-stratifold graph and then with a string representation we were able to reach that goal. We not only classify them, we also counted them (for few vertices) and gave them a useful nomenclature to identify each of them.

Also the implementation of the search engine, would be helpful for the ones who are interested on this subject and want to work with a specific set of this trivalent 2-stratifolds with trivial fundamental group. More graphs can be added in the future with the help of a more powerful computer.

More over, if there is a way of building the trivalent-stratifold graphs with n white vertices, without using the ones with less white vertices. That would be a great optimization on the application of the algorithms described on Chapter 6 and more graphs would be classified.

This work can be extended to use the string representation to identify graphs that come from 2-stratifolds with trivial fundamental group, not necessarily trivalent. It is important that the 2-stratifold have trivial fundamental group because that's the property that give us a graph that is a tree. But for a different fundamental group it would be possible to look for different ways of implementing the observations here described.

Bibliography

[1] Charles Livingston and Allison H. Moore. Knotinfo: Table of knot invariants. URL: `knotinfo.math.indiana.edu`, Current Year.

[2] James R. Munkres. *Topology*. Prentice Hall, Inc., Upper Saddle River, NJ, 2000. Second edition of [MR0464128].

[3] Jean Gallier and Dianna Xu. *A guide to the classification theorem for compact surfaces*, volume 9 of *Geometry and Computing*. Springer, Heidelberg, 2013.

[4] Paul Bendich, Ellen Gasparovic, John Harer, and Christopher Tralie. *Scaffoldings and Spines: Organizing High-Dimensional Data Using Cover Trees, Local Principal Component Analysis, and Persistent Homology*, pages 93–114. 07 2018.

[5] Lum P., Singh G., Lehman A., and et al. Extracting insights from the shape of complex data using topology. *Sci Rep 3*, 1236, 2013.

[6] John Matson. Frothy physics: The math of foams, Jul 2013. https://www.scientificamerican.com/article/frothy-physics-the-math-foams/.

[7] J. C. Gómez-Larrañaga, F. González-Acuña, and W. Heil. 2-dimensional stratifolds. In *A mathematical tribute to Professor José María Montesinos Amilibia*, pages 395–405. Dep. Geom. Topol. Fac. Cien. Mat. UCM, Madrid, 2016.

[8] Allen Hatcher. *Algebraic topology*. Cambridge University Press, Cambridge, 2002.

[9] Gary Chartrand and Ping Zhang. *Chromatic graph theory*. Discrete Mathematics and its Applications (Boca Raton). CRC Press, Boca Raton, FL, 2009.

[10] Béla Bollobás. *Graph theory*, volume 63 of *Graduate Texts in Mathematics*. Springer-Verlag, New York-Berlin, 1979. An introductory course.

[11] J. C. Gómez-Larrañaga, F. González-Acuña, and W. Heil. 2-dimensional stratifolds homotopy equivalent to S^2. *Topology Appl.*, 209:56–62, 2016.

[12] Walter D. Neumann. A calculus for plumbing applied to the topology of complex surface singularities and degenerating complex curves. *Trans. Amer. Math. Soc.*, 268(2):299–344, 1981.

[13] Covering space. URL: `https://en.wikipedia.org/wiki/Coveringspace#Definition`. From Wikipedia, edited on 13 August 2022.

[14] J. C. Gomez-Larrañaga, F. González-Acuña, and W. Heil. Classification of simply-connected trivalent 2-dimensional stratifolds. *Topology Proc.*, 52:329–340, 2018.

[15] J. C. Gomez-Larrañaga, F. González-Acuña, and W. Heil. Models of simply-connected trivalent 2-dimensional stratifolds. *Bol. Soc. Mat. Mex.*, 26:1301—-1312, 2020.

[16] Myriam Hernández-Ketchul and Jesús Rodríguez-Viorato. String representation of trivalent 2-stratifolds with trivial fundamental group. *Boletín de la Sociedad Mexicana de Matemáticas, manuscrito BSMM-D-21-00044R4 en proceso de publicar.*, 2021.

[17] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1975. Second printing, Addison-Wesley Series in Computer Science and Information Processing.

[18] D.E. Knuth. *Art of Computer Programming, Volume 4, Fascicle 4,The: Generating All Trees–History of Combinatorial Generation.* Pearson Education, 2013.

[19] Douglas M. Campbell and David Radford. Tree isomorphism algorithms: speed vs. clarity. *Math. Mag.*, 64(4):252–261, 1991.

[20] Myriam Hernández-Ketchul and Jesús Rodríguez-Viorato. Trivalent stratifold repository. https://github.com/MyHerket/Trivalent_Stratifolds_Graphs, 2020. GitHub repository.

[21] Yair Hernández. Stratifolds. URL: `https://github.com/yair-hdz/stratifolds`, 2018. GitHub repository.

[22] Diego Estrada-Talamantes and Myriam Hernández-Ketchul. Search engine for trivalent stratifolds. URL: `http://trivalent-stratifolds.com/`, 2021.