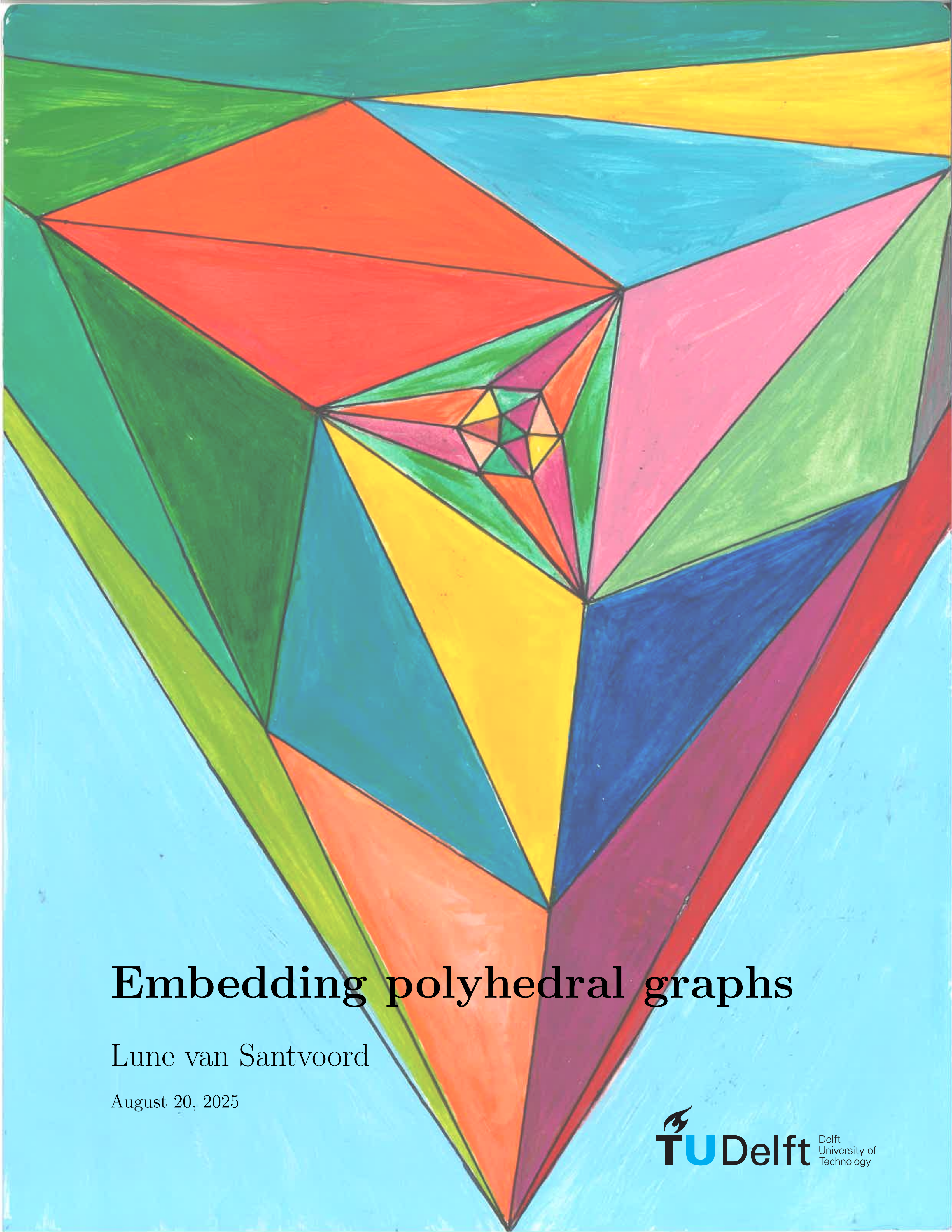


# Embedding polyhedral graphs

Lune van Santvoord

August 20, 2025

TU Delft Delft University of Technology

# Embedding polyhedral graphs

by

Lune van Santvoord

Bachelor's Thesis
Submitted to the Delft Institute of Applied Mathematics
Delft University of Technology

In partial fulfillment of the requirements of
the degree Bachelor of Science in Applied Mathematics
at the Delft University of Technology
to be defended publicly on Wednesday August 27, 2025 at 13:00 PM.

| | | |
|---|---|---|
| Student number: | 5872952 | |
| Project duration: | March 1, 2025 – August 27, 2025 | |
| Thesis committee: | Dr. F.M. de Oliveira Filho, | TU Delft, supervisor |
| | Dr. E. Lorist, | TU Delft |
| Cover drawing: | Luna van den Elzen | |

# Preface

*"Graph drawing is not a single well-defined problem but an art, namely the art of describing what a nice drawing of a graph means in the context of a particular application."*[1]

While I do agree with this quote, for me the first part suffices. This project reminded me again of the art and beauty of mathematics aside from its applications.

I would like to thank Dr. F.M. de Oliveira Filho for supervising me during this project. He provided me with initial stepping stones and advised me on making the right decisions. Additionally, I would like to thank Dr. E. Lorist for joining the Bachelor's Comittee. Finally, I would like to thank Luna, for creating a beautiful cover drawing, and the rest of my friends and family who challenged me to explain my thesis to them.

*Lune van Santvoord*
*Amsterdam, August 2025*

[1]From a chapter by R. Fleischer and C. Hirsch in a book by Kaufmann and Wagner (2003)

## Lay Summary

Graphs are structures consisting of objects and relations between those objects. A drawing of a graph is made by representing the objects as points and the relations as curves between their endpoints. When a graph has a drawing where these curves only cross in their endpoints, the graph is planar. One method of drawing a planar graph is by fixing an initial set of the objects in the two-dimensional plane and by letting the relations behave as ideal rubber bands, letting the rest of the vertices settle in equilibrium. This is called the rubber band representation. For a specific type of graph, Tutte proved that this representation has certain characteristics. This research, by adapting the problem corresponding to the rubber band representation, created several different drawings while maintaining the characteristics given by Tutte.

# Abstract

Graphs are mathematical models that contain information about objects (vertices) and relations between those objects (edges). A drawing, also called an embedding, of a graph is made by representing the vertices as points in $\mathbb{R}^2$ and the edges as curves between their endpoints. When these curves only intersect in their endpoints, the embedding is planar. Graphs that have such an embedding are planar graphs. There is no fixed set of rules when it comes to drawing a graph. So how to decide how a graph should be drawn? One method of drawing a graph is called the rubber band representation, where some vertices are initially fixed as a strictly convex polygon on the Euclidean plane and all remaining vertices are placed in the barycenter of their neighbours. For the rubber band representation of 3-connected planar graphs, further referred to as the Tutte embedding, Tutte's theorem states that all connected regions that are bounded by edges (called faces) are strictly convex, these faces do not contain vertices or edges. This research adapted the optimization problem corresponding to the Tutte embedding by changing the objective function, while making sure the embedding remained compliant with Tutte's theorem. In doing so, new methods for graph drawing can be examined and used in various areas, depending on the individual context and application of the drawing. Two types of objective functions were analysed. The first type, based on existing research, minimizes the difference between the length of the edges and their desired length. The second type was proposed by this research and minimizes the differences between the surface area of all faces within the embedding. The first type of objective functions yielded embeddings with a similar structure as the Tutte embedding, maintaining the symmetries of the graph, but with different proportions. The embeddings that were yielded by the second type of objective function were different. Some of which do not maintain the symmetric characteristics of the graph. A notable feature is that for some graphs, there exist several local minima with corresponding embeddings that are significantly different from each other. This report contains the analysis of the different objective functions and their results. The results show that there are multiple ways to draw 3-connected planar graphs compliant with Tutte's theorem. Moreover, it provides options for further analysis of the objective functions and paves the way for further research possibilities.

*Key words*: graph drawing, 3-connected planar graph, planar embedding, rubber band representation, Tutte's theorem, optimization

# Contents

# Introduction

Graphs are abstract mathematical objects that are unknown to most people (Kaufmann & Wagner, 2003). However, graphs appear not only in mathematics or computer science, they live under the surface across many areas. There are various kinds of information that can be represented by a graph. Think of railway systems, scheduling problems and biological networks such as the phylogenetic tree in Figure 0.1.

A graph is a structure containing objects and relations between these objects. The objects are called vertices and the relations between the vertices are called edges. It is possible for two objects to be related in more than one way or for one object to be related to itself. In Figure 0.1, the T-junctions and the open ends are the vertices of the graph, and the connections between these are the edges.

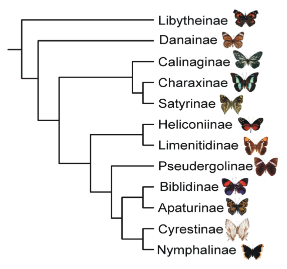


FIGURE 0.1. The phylogenetic tree of the Nymphalidae

Because graphs are so widely occurring, the study on them, called graph theory, is a major branch in mathematics. In addition, graphs have a very visual structure. Therefore, Kaufmann and Wagner (2003) emphasize the importance of the study of *graph drawing*; representing the vertices of a graph as points in $\mathbb{R}^2$, where related vertices are connected by a curve. These curves do not contain any other vertices. A drawing is also called an *embedding.* When none of the curves intersect, it is called a *planar embedding.* A graph that has such an embedding is called a *planar graph.*

Creating suitable ways of graph drawing has a positive effect on its applications in computer science, such as software engineering, data science, network design and graphical interfaces. But also on applications in other areas, such as graphical data analysis in scientific areas and visualization of information in general. (Kaufmann & Wagner, 2003)

Essentially, there are an infinite number of possible drawings of a single graph. To make a suitable drawing of a graph, one has to account for the intended structure and meaning. For example, the graph displayed in Figure 0.1 would be more difficult to interpret, with regard to the evolutionary structure it contains, if all vertices were placed randomly (preserving the correct connections). In addition, one could also consider the graph drawing problem from an aesthetic point of view, which would lead to more subjectiveness.

All in all, a drawing is dependent on the individual and subjective nature of the graph, which makes drawing in itself a difficult matter when considering the structure and meaning of the graph (Kaufmann & Wagner, 2003).

However, it also makes the subject interesting. There are no fixed rules when it comes to drawing a graph. The only importance is that the drawing is helpful to the viewer or user in regard to 'reading' the graph. Imagine trying to figure out a clustered map of a metro system in a large city. Therefore, creating and improving algorithms for the graph-drawing problem is a dynamic and demanding problem (Kaufmann & Wagner, 2003).

One specific way to draw a graph is to take a selection of the vertices and fix them and the edges between them. Now, imagine that all other edges behave as contracting rubber bands and let the rest of the vertices be pulled by their edges. Consequently, these vertices will settle in equilibrium, in fact, in the barycenter of their neighbours. Think of a dreamcatcher, where the crossings of the rope are the vertices and the pieces of rope between these crossings are the edges. This drawing is called the *rubber band representation*.

A graph is *connected* if there exists a sequence of edges between any two vertices. It is *3-connected* if it is connected and remains connected after removal of any two distinct vertices. In 1963, W.T. Tutte proved a theorem that states that for a 3-connected planar graph, the rubber band representation of this type of graph has its own characteristics. The rubber band representation will be further referred to as the *Tutte embedding*. The Tutte embedding of a graph can easily be obtained by solving a linear system that is derived from an optimization problem. Lovász (2009) showed that this optimization problem could be intuitively adapted by using a set of various rubber bands instead of one kind. Consequently, the question arises how to decide what kinds of rubber bands should be used and, for each edge, what the corresponding kind should be. Hence, the main question for this research reads as follows:

*How can the optimization problem, that corresponds to the Tutte embedding of a 3-connected planar graph, be adapted so that the embedding remains compliant with Tutte's theorem?*

To answer the research question, a new optimization problem is created. The optimization problem is tested with several objective functions. Chapter 3 elaborates on this. The results will be discussed in chapter 4. Chapter 1 first delves deeper into the Tutte embedding. In the second chapter is explained how the original optimization problem is changed. This chapter also includes some research on possible objective functions. Finally, the main research question will be answered based on the results. Note that this report focusses only on the mathematics and will not qualify the results as 'good' or 'bad', since this is a highly subjective matter and completely dependent on the users preferences.

CHAPTER 1

# Tutte's embedding

## 1. Definitions

We are looking at a *graph* $G = (V, E)$, where $V$ is the set of *vertices* and $E$ is the set of *edges* between those vertices.

For $u, v \in V$, let $u$ be a *neighbour* of $v$ if $uv \in E$. Then $N(v)$ denotes the set of neighbours of $v$, i.e., $N(v) = \{u \in V\colon\ uv \in E\}$. The *degree* of $v$, denoted $d(v)$, is the number of neighbours of $v$, that is, $d(v) = |N(v)|$.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there exists a bijection $f\colon\ V \to V'$ such that $uv \in E$ if and only if $f(u)f(v) \in E'$

A *drawing* of a graph $G$ is a representation of $G$, where the vertices of $G$ are points in $\mathbb{R}^2$ and the edges of $G$ are curves between their endpoints.

A *polygon* is a sequence $(x_1, ..., x_n)$ of distinct points in $\mathbb{R}^2$, such that the line segments between $x_1$ and $x_n$ and any two consecutive $x_is$ do not intersect, except at the endpoints. The line segments enclose a connected region in $\mathbb{R}^2$.

A set $S \subseteq \mathbb{R}^2$ is *convex* if for any two points $x, y \in S$, $S$ contains the line segment between $x$ and $y$. That is, for $0 \leq \lambda \leq 1$, we have $\lambda x + (1 - \lambda)y \in S$. The convex set $S$ is *strictly* convex if no three points of $S$ on the boundary are collinear.

A point $x$ in $\mathbb{R}^2$ is a *convex combination* of $x_1, ..., x_k$ in $\mathbb{R}^2$ if $x = \sum_{i=1}^{k} \alpha_i x_i$, where $\alpha_1, ..., \alpha_k$ are constants in $\mathbb{R}$ such that $\alpha_i \geq 0$ for all $1 \leq i \leq k$ and $\sum_{i=1}^{k} \alpha_i = 1$.

**1.1. 3-connected planar graphs.** In this paper, we are looking at a specific type of graph. A graph $G$ is *planar* if it has a drawing in $\mathbb{R}^2$ such that no edges cross in a point that is not a vertex of $G$. This drawing is called a *planar embedding* of $G$. We will focus on straight-line planar embeddings. Moreover, a *face* $F$ of a graph is a maximal connected region in $\mathbb{R}^2$ that contains no vertices or edges of $G$. The *boundary* of a face consists of the edges that enclose the region. The *outer face* is the only face that is unbounded.

A straight-line embedding is given by an injective function $f\colon\ V \to \mathbb{R}^2$; each edge $uv \in E$ is drawn as a line segment between $f(u)$ and $f(v)$. If the embedding is planar, these line segments will only intersect at their endpoints.

Define a *subdivision* of a graph $G$ as a graph where some edges of $G$ are replaced by a path. Note that the replacing edges and vertices are not in $G$. Kuratowski (1930) characterized planar graphs as those that do not contain a subdivision of the non-planar graphs $K_5$ and $K_{3,3}$, the complete graph on five vertices and the complete bipartite graph on six vertices. See Figure 1.1 for $K_5$ and $K_{3,3}$ and an example of a subdivision of each.

Theorem 1 (Kuratowski, 1930). *A graph $G$ is planar if and only if $G$ does not have a subgraph that is isomorphic to a subdivision of $K_5$ or $K_{3,3}$.*

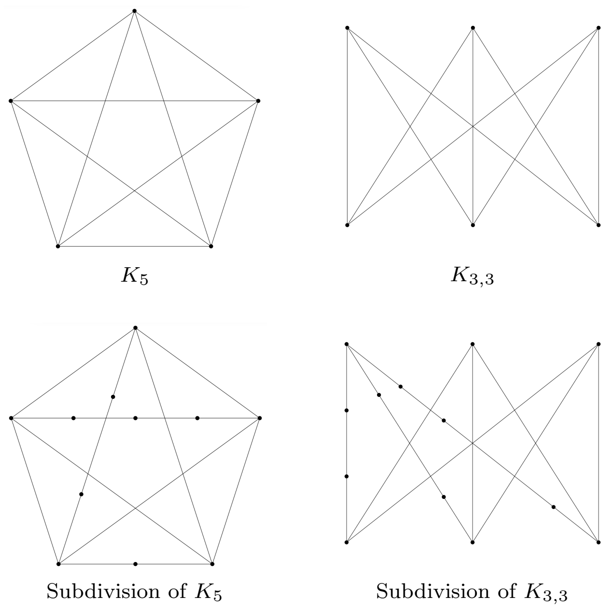


Figure 1.1

A non-empty graph $G = (V, E)$ is connected if there is a path between any two vertices in $G$. It is 3-connected if any two of its vertices can be joined by three independent paths. Two paths are independent if they share no vertices, except those at the endpoints. This definition implies that $G$ is 3-connected if it is connected and remains connected after the removal of any two distinct vertices. (Diestel, 2017)

The graphs used for this research are thus the 3-connected planar graphs. Now that the relevant definitions have been discussed, the next section will give a detailed explanation of the Tutte embedding as mentioned in the Introduction.

## 2. The Tutte embedding

**2.1. Physical analogy.** As mentioned in the Introduction, there are multiple ways to embed a graph. In fact, there are an infinite number of ways to draw a graph. Tutte introduced an embedding in 1962. The embedding is defined by a physical system.

Let $G = (V, E)$ be a connected graph. Note that $G$ is not necessarily 3-connected or planar. Let $V_0 \subseteq V$ be the vertices of some face of a planar embedding of $G$. This face will be further referred to as the outer face $F_0$. Fix these vertices on the two-dimensional plane, such that the bounded region inside the outer face forms a convex set in $\mathbb{R}^2$. Imagine that the edges behave as rubber bands that tighten themselves. Now release all the vertices that are not in $V_0$. This leads to the vertices settling in equilibrium inside the region bounded by the outer face. Therefore, this embedding is also known as the rubber band representation, but will further be referred to as the *Tutte embedding.* We also use this definition since the rubber band representation is a more general embedding that is also applicable to graphs that are not planar or 3-connected.

**2.2. Mathematical model.** We have seen how the Tutte embedding is defined physically. This embedding can be modeled mathematically such that the equilibrium positions are fairly easy to calculate as a system of linear equations.

Let $p\colon V \to \mathbb{R}^2$ be the function that assigns coordinates to the vertices in $V$. The edges behaving as rubber bands mathematically corresponds to the total energy of $G$ being minimized. The total energy is given by:

$$\mathcal{E}(p) = \sum_{uv\in E} \|p(u) - p(v)\|^2. \tag{1.1}$$

Let $p_0\colon V_0 \to \mathbb{R}^2$ be the function that fixes the vertices of the outer face. The equilibrium of the vertex positions, where the total energy is minimal, is the solution of the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \mathcal{E}(p) \\ \text{s.t.} \quad & p(u) = p_0(u) \ \text{ for all } \ u \in V_0. \end{aligned} \tag{1.2}$$

This problem indeed has a unique optimal solution, since the function $\mathcal{E}$ is strictly convex. Namely, for every $p, q\colon V \to \mathbb{R}^2$ and every $0 \leq \lambda \leq 1$, we have $\mathcal{E}(\lambda p + (1-\lambda)q) < \lambda\mathcal{E}(p) + (1-\lambda)\mathcal{E}(q)$. Moreover, in (1.2), if $\|p(u)\|$ for $u \notin V_0$ tends to infinity, so does $\mathcal{E}(p)$. Together with strict convexity, this implies that the optimal solution is unique.

To continue, we are searching for the extreme point of a strictly convex function. Thus, the optimal solution $p$ satisfies $\nabla\mathcal{E}(p) = 0$. Write $p(u) = (x_u, y_u)$. We now work out the partial derivative of $\mathcal{E}(p) = \sum_{uv\in E} \|(x_u, y_u) - (x_v, y_v)\|^2$ with respect to $x_u$:

$$\begin{aligned} \frac{\partial\mathcal{E}(p)}{\partial x_u} &= \sum_{vw\in E} \frac{\partial}{\partial x_u}\|(x_v, y_v) - (x_w, y_w)\|^2 \\ &= \sum_{vw\in E} \frac{\partial}{\partial x_u}((x_v - x_w)^2 + (y_v - y_w)^2) \\ &= \sum_{v\in N(u)} \frac{\partial}{\partial x_u}(x_u^2 + x_v^2 + 2x_u x_v) \\ &= \sum_{v\in N(u)} 2(x_u - x_v). \end{aligned}$$

Hence, if $\nabla\mathcal{E}(p) = 0$ we have

$$\begin{aligned} 0 &= \sum_{v\in N(u)} 2(x_u - x_v) \\ \iff \sum_{v\in N(u)} x_u &= \sum_{v\in N(u)} x_v \\ \iff \quad d(u)x_u &= \sum_{v\in N(u)} x_v \\ \iff \qquad x_u &= \frac{1}{d(u)} \sum_{v\in N(u)} x_v. \end{aligned}$$

for every $u \in V \setminus V_0$ and where $d(u)$ is the degree of the vertex $u \in V$ as defined in Section 1. The similar holds when calculating the partial derivative with respect to $y_u$. This means that optimal

$p$ is a solution to the following linear system:

$$
\begin{aligned}
p(u) &= p_0(u) && \text{for all } u \in V_0, \\
p(u) &= \frac{1}{d(u)} \sum_{v \in N(u)} p(v) && \text{for all } u \in V \setminus V_0.
\end{aligned} \tag{1.3}
$$

In other words, the position of every vertex not in $V_0$ is the barycenter of its neighbours. Define the *1-skeleton* of a polyhedron as the graph whose vertices are the vertices of the polyhedron and in which two vertices are adjacent if they belong to the same one-dimensional face of the polyhedron. In Figure 1.2, the Tutte embedding of the 1-skeleton of three polyhedra are presented. Notice how two embeddings of the same polyhedron are significantly different as a consequence of choosing outer faces with different lengths.

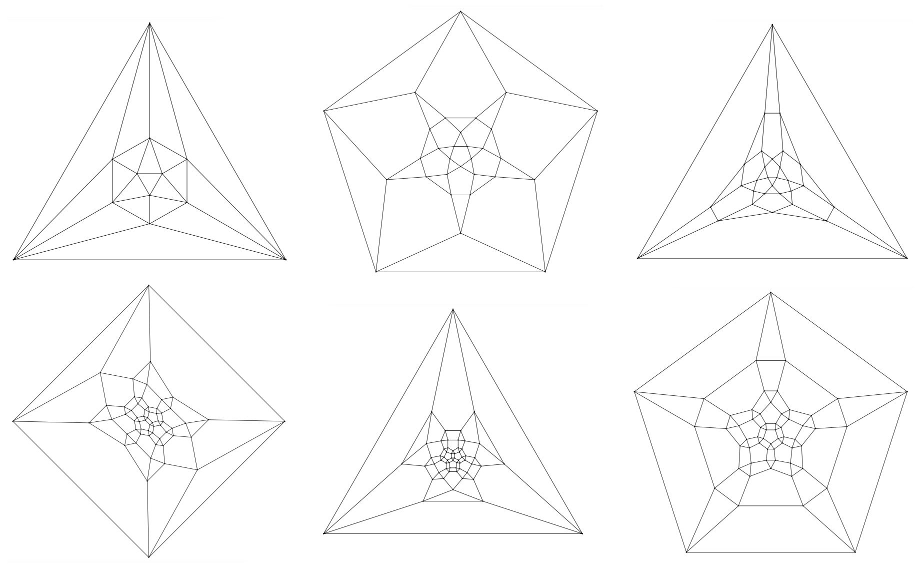


FIGURE 1.2. The Tutte embedding of three polyhedra. In the top right the 1-skeleton of the icosahedron. The top middle and top left are the 1-skeleton of the icosidodecahedron, with a different choice of outer face. On the bottom row the three embeddings of the 1-skeleton of the rhombicosidodecahedron, again with outer faces of a different length.

## 3. Tutte's theorem

Thus, the rubber band representation is a means of displaying all types of graphs in the two-dimensional plane. However, for the Tutte embedding specifically, Tutte (1963) proved that the embedding is actually planar. The theorem is stated below. [1]

[1]The proof is too elaborate to discuss in this paper and would distract from the current research, therefore it will not be treated. A more detailed version can be found in Lovász (2009).

Theorem 2 (Tutte, 1962). *Let $G = (V, E)$ be a 3-connected planar graph and $F_0$ be a face of some planar embedding of $G$. Let $E_0$ be the edges in the boundary of $F_0$, where $V_0$ are the endpoints of the edges in $E_0$. Now define the graph $C_0 = (V_0, E_0)$. If $p_0 \colon V_0 \to \mathbb{R}^2$ is a straight-line embedding of $C_0$ that maps $V_0$ to the vertices of a strictly convex polygon, then the Tutte embedding of $G$ is a straight-line embedding of $G$ in which $C_0$ forms the outer face and every inner face is a strictly convex polygon.*

In short, the theorem states that for the Tutte embedding of some 3-connected planar graph $G$ with outer face $F_0$, every face inside the outer face is a strictly convex polygon. This can also be seen in Figure 1.2. However, this implies that instead of the Tutte embedding, some other embedding could be taken, where all vertices not in $V_0$ are positioned inside the outer face, so that all inner faces are convex polygons.

This follows from the proof of Tutte's theorem (Lovász, 2009). The proof does not build on the fact that $p(u)$ is a solution to the linear system in equation (1.3). In fact, it suffices for $p$ to be a solution of a linear system where $p(u)$ is a convex combination with non-zero coefficients of $p(v)$ for $v$ in $N(u)$.

By changing the weights of the convex combination, different drawings can be obtained. This could be useful for certain applications or from the aesthetic point of view. However, if the linear system changes, it will not be a solution to $\nabla \mathcal{E}(p) = 0$. So, how does this affect the calculation of the total energy of the graph? The next chapter will investigate this question.

CHAPTER 2

# Changing the system

Before going into the problem at hand, a short note will be made on drawing planar graphs as a physical system. There are various types of graphs with different applications. Moreover, there are various ways of drawing them and various means to qualify a drawings of some graph (Kamada & Kawai, 1989; Kaufmann & Wagner, 2009). However, this report will treat only the part relevant to this research.

## 1. Drawing planar graphs as a physical system

Firstly, we are working with planar graphs. Planarity is a pleasant characteristic when it comes to drawing graphs. A planar embedding has no crossings, making it more accessible and comprehensible to the human eye (Purchase, 1997).

Secondly, the rubber band representation is a physical system of objects and rubber bands. Methods based on physical analogies are practical for several reasons: they are very intuitive and easily comprehensible and programmable (Brandes, 2001). Moreover, for graphs up to around fifty vertices, these methods often deliver satisfying results.

The optimization problem can be a general version. The function that is minimized can then be an adaptable component depending on the characteristics. Still, how such a physical model should be defined, depends on the context and characteristics of the graph, but this report will not elaborate on this.

For this research, the optimization problem places vertices in a convex combination of its neighbours. Subsequently, the function that minimizes the energy, called the objective function, is to be defined. Several definitions of the objective function will be treated in Chapter 3. But first, the optimization problem itself will be explained.

## 2. Vertices as a convex combination of their neighbours

Recall the 3-connected planar graph $G = (V, E)$, with an outer face containing the vertices $V_0$. In Chapter 1 was discussed how Tutte's theorem still holds when the vertices not in $V_0$ are placed as a convex combination of their neighbours instead of being the barycenter. More intuitively, this new representation can be considered as a variant on the Tutte embedding, where the edges are defined by different types of rubber bands.

Consequently, these coefficients can be interpreted as a certain pulling force of one vertex on another. We will define a new system that is similar to the barycentric linear system in (1.3). Let $\alpha_{uv}$ be the coefficient that corresponds to the force of the vertex $v$ that pulls on $u$. We assume $\sum_{v \in N(u)} \alpha_{uv} = 1$ and $\alpha_{uv} > 0$, to prevent collinearity of more than two vertices. We also assume $\alpha_{uv} \neq \alpha_{vu}$ to create a more general problem. Then the desired $p_\alpha$ will satisfy the following linear system:

$$\begin{aligned} p_\alpha(u) &= p_0(u) && \text{for all } u \in V_0, \\ p_\alpha(u) &= \sum_{v \in N(u)} \alpha_{uv} \cdot p_\alpha(v) && \text{for all } u \in V \setminus V_0. \end{aligned} \tag{2.1}$$

However, as mentioned in the first chapter, changing the images of the vertices of $V$ will change the function for the total energy, since it now also depends on $\alpha_{uv}$. Unfortunately, reversing the calculations of the barycentric linear system in the system in (2.1) does not provide an energy function that corresponds to the convex linear system. This is only possible when considering a less general version in which the alphas are symmetric (Lovász, 2009).

Therefore, the optimization problem as in (1.2) will be considered, where $p$ is replaced by $p_\alpha$ and the alphas are non-symmetric. Subsequently, instead of the energy function $\mathcal{E}(p)$, various functions will be considered. In Section 3, some possible energy functions will be discussed. In the meantime, a short note on finding the outer face.

**2.1. Choosing the outer face.** It has been made clear how the Tutte embedding works and how the embedding could be adapted from barycentric into convex. However, it is implied that the outer face has been given in advance.

When this is not the case, Lovász (2009) suggests using a depth-first search spanning tree in the graph. This tree has the property that for every edge of the graph, both of its endpoints lie on a path starting at the root of the tree. In short, this spanning tree $T$ can then be used to find an initial face, by finding a specific minimal path in $T$ that can be closed by an edge not in $T$. Then the initial face is given by attaching this edge to the path.

For this research, the input data that is used already contains all faces of the graph. This will be further explained in Section 3

## 3. Existing energy functions

The choice of a suitable energy function depends on the individual context of a graph and its purpose. The study on what is desirable is too broad for this report. Nevertheless, a few criteria are generally accepted. It is agreed upon that vertices should be well-distributed over the area and adjacent vertices should be close (Brandes, 2001). As mentioned earlier, minimizing the number of edge crossings is also considered a criteria, but is clearly not a problem in the planar case.

Kamada and Kawai (1989) consider these criteria for their optimization algorithm for drawing general graphs. They create an energy function that calculates the positions $p$ of the vertices by minimizing the difference between the desirable length of all edges and the distance between the vertices. Their function looks as follows:

$$\mathcal{E}_{KK}(p) = \sum_{u,v \in V} \frac{c}{d(u,v)^2} (\|p(u) - p(v)\| - l_{uv})^2, \tag{2.2}$$

where $c$ is a scaling constant and $d(u, v)$ the length of the shortest path between the vertices $u$ and $v$. The desired distance between $u$ and $v$ is represented by $l_{uv}$, it is calculated as the product of the desired edge length $L$ and the length of the shortest path between two vertices. When working on a display with restricted side length $L_0$, $L$ is calculated as

$$L_0 / \max_{u,v \in V} d(u,v).$$

Kamada and Kawai created an algorithm for general graphs. Therefore, in combination with the convex property of the inner faces, the function in (2.2) is well applicable to the problem in this research. In fact, it is similar to Tutte's energy function when considering an ideal length of zero, letting $c/d(u,v)^2 = 1$ and summing only over adjacent vertices.

Dividing by the squared length of the shortest path at the beginning of the equation, helps to control the influence that two (non-)adjacent vertices exert on each other. However, the desired distance $l_{uv}$ is not directly usable for this research, since the outer face is fixed and all vertex positions are a convex combination of their neighbours. This value could depend on several factors, such as the length and diameter of the outer face or the maximum length of the shortest path. The desired length could also be taken as a function of the mean edge length from the original Tutte embedding. More on the choice of this desired distance can be found in Chapter 3.

The objective function in (2.2) resembles the family of objective functions as described by Cohen (1997):

$$\mathcal{E}_k(p) = (\sum_{u,v\in V} l_{uv}^{2-k})^{-1} \cdot \sum_{u,v\in V} \frac{1}{l_{uv}^k}(\|p(u) - p(v)\| - l_{uv})^2, \tag{2.3}$$

where $k \in \{0, 1, 2\}$. For $k = 0$, no distinction is made between long and short distances when minimizing the difference between the final distance and the desired distance between two vertices. If $k = 1$, two vertices with a larger desired distance are allowed to deviate more from this distance in their final positions than two vertices with a smaller desired distance. When $k = 2$, only errors are penalized that are proportionally large compared to the desired distance. (Cohen, 1997) This last case is the most similar to $\mathcal{E}_{KK}$ depending on how exactly the desired distance is defined.

It would appear that, by penalizing longer distances between vertices, the energy function in (2.2) results in less cluttered embeddings that have fewer small angles (Brandes, 2001).

In the next chapter, it will be explained how the functions as described by Kamada and Kawai and Cohen will be used in this research.

CHAPTER 3

# A new optimization problem

In this chapter, the energy functions that are used for this research are discussed. Given a 3-connected planar graph $G = (V, E)$, a set $V_0 \in V$ and weights $\alpha$, recall that $p_\alpha\colon V \to \mathbb{R}^2$ is defined as in (2.1). We want to solve the following optimization problem on the variables $\alpha$, where $\mathcal{E}$ is some energy function:

$$\begin{aligned}
\text{minimize} \quad & \mathcal{E}(p_\alpha) & & \\
\text{s.t.} \quad & p_\alpha(u) &= p_0(u) \quad & \forall u \in V_0, \\
& p_\alpha(u) &= \sum_{v \in N(u)} \alpha_{uv} \cdot p_\alpha(v) \quad & \forall u \in V \setminus V_0, \\
& \alpha_{uv} &> 0 \quad & \forall u, v \in V, \\
& \sum_{v \in N(u)} \alpha_{uv} &= 1 \quad & \forall u \in V \setminus V_0.
\end{aligned} \tag{3.1}$$

The first two constraints form the linear system that needs to be solved.

The input is a 3-connected planar graph $G = (V, E)$ on $n$ vertices, $m$ edges and $l$ faces. An outer face $F_0$ is given. The first step is to fix the positions of the vertices $V_0$ of the outer face. The outer face will be embedded as a regular polygon around a central point $(x_0, y_0)$, with the vertices located at the boundary of a circle with radius $r$. The coordinates of the positions of the vertices of $V_0$ will be calculated as follows:

$$\begin{aligned}
x_i &= x_0 + r \cdot \cos(i \cdot \varphi), \\
y_i &= y_0 + r \cdot \sin(i \cdot \varphi),
\end{aligned}$$

for $1 \leq i \leq |V_0|$ and where $\varphi = 2\pi/|V_0|$ is the central angle between two adjacent vertices.

## 1. Optimization method

In Python, `scipy.optimize.minimize` package is used, in particular, the 'COBYLA' method. The solver requires two arguments, the function that has to be minimized and an initial guess in the form of a vector of length $q$, where $q$ is the number of independent variables.

All $\alpha$'s are placed in an $n$ x $n$-matrix $W$, such that for every edge $uv$ in $G$, the element $W_{ij}$ corresponds to $\alpha_{uv}$, where $u$ is the $i$-th vertex and $v$ the $j$-th vertex. These $\alpha$'s are the independent variables. Therefore, the initial guess vector $\mathbf{w}$ has length $q = 2 \cdot m$, where $m$ was the number of edges of $G$. $\mathbf{w}$ contains real numbers from the interval $[0, 1)$.

However, the alphas must satisfy $\alpha_{uv} > 0$ and $\sum_{v \in N(u)} \alpha_{uv} = 1$. To ensure this, they are 'transformed' in the following manner:

$$\tilde{\alpha}_{uv} = 1 + w_{uv}^2$$

$$\alpha_{uv} = \frac{\tilde{\alpha}_{uv}}{\sum_{v \in N(u)} \tilde{\alpha}_{uv}}$$

Subsequently, the linear system found in equation 3.1 is solved in the form of a matrix equation $A\mathbf{x} = \mathbf{b}$, where the elements of $A$ are defined as

$$A_{uv} = \begin{cases} -W_{uv}, & \text{if } v \in N(u) \\ 1, & \text{if } u = v \\ 0, & \text{else} \end{cases} \tag{3.2}$$

for all $u$ not in $V_0$. And $\mathbf{b}$ is the vector so that

$$b_u = \begin{cases} p_0(u), & \text{if } u \in V \setminus V_0 \\ 0, & \text{else} \end{cases} \tag{3.3}$$

for all $u$ in $V_0$. Consequently, the vector $x$ of length $n$ corresponds to the two-dimensional coordinates of the vertices.

Now, the `scipy` solver will minimize over a function. This is the energy function of the optimization problem found in (3.1). The solver returns the optimized vector $\mathbf{w}$, which is used to determine the final and optimal positions of all vertices, based on the chosen function. Note that by the definition of $\alpha$ and the manner in which the linear system is solved, each iteration of the solver always satisfies Tutte's theorem.

For this research, as mentioned in the previous chapter, several functions will be considered and tested. The next section will elaborate on the decision on different types of energy functions.

## 2. The energy functions

In section 3 was explained how the energy function could be defined. In this section will be elaborated on the different energy functions that are used for this particular research.

**2.1. Euclidean distance between vertices.** This first energy function is based on the functions from Kamada and Kawai (1989) and Cohen (1997), as described in the previous chapter. For all vertex combinations, the function calculates the deviation from the desired distance of the distance between the final positions of the vertices. The function does not consider when both vertices are part of the outer face. For $k \in \{-1, -2, -3\}$, we have:

$$\mathcal{E}_{k,m}(p_\alpha) = \sum_{\substack{u,v \in V \\ u \vee v \in V \setminus V_0}} c(u,v)^k \cdot (\|p_\alpha(u) - p_\alpha(v)\|^2 - l_{uv}^2)^2,$$

where $c(u,v)$ is a function that controls the influence that two distinct vertices $u$ and $v$ exert on each other. $l_{uv}$ is the desired distance between $u$ and $v$. Note that the Euclidean distance, $\|p_\alpha(u) - p_\alpha(v)\|$, and $l_{uv}$ are squared. This is for computational reasons. Moreover, the fraction at the beginning of equation (2.3) has no additional value to this research, since it only scales the value of the energy function. In addition, for this research, the value of $\mathcal{E}_{k,m}$ is only important for comparisons. Therefore it is not necessary that it perfectly imitates a physical situation.

A specific case is when $k$ and $l_{uv}$ are equal to zero. Then minimizing $\mathcal{E}_{k,m}$ would yield the same embedding as the Tutte embedding, aside from the value of $\mathcal{E}$.

In lieu of $c(u, v)$, we could consider taking the length of the shortest path, $d(u, v)$, or the desired distance, $l_{uv}$ (Kamada & Kawai, 1989; Cohen, 1997). Subsequently, the amount of influence that two (non-)adjacent vertices exert on each other can be controlled by setting an integer value for the constant $k$ in $c(u, v)^k$. By considering a larger value, the deviation for vertices that are further away is weighed less heavy than for vertices that are closer together.

For the desired distance $l_{uv}$, the value as described in Section 3 could be considered, that is:

$$l_{uv} = \frac{m \cdot L_0}{max_{u,v \in V} d(u, v)} \cdot d(u, v), \tag{3.4}$$

where $L_0$ is taken as the *diameter* of the outer face and $m$ is a scaling constant. This diameter is calculated as the Euclidean distance between the two vertices of the outer face where the shortest path in the outer face is maximal.

Another method we suggest is to consider $l_{uv}$ as the square root of the mean length $M$ of the inner edges and to let the summation in $\mathcal{E}(p_\alpha)$ only go over the inner edges. Write $E_{in}$ for all edges of $E$ that are not part of the outer face. $M$ is calculated as follows:

$$M = \frac{1}{|E_{in}|} \sum_{\substack{u,v \in V \\ uv \in E_{in}}} \|p_\alpha(u) - p_\alpha(v)\|^2. \tag{3.5}$$

This energy function forces the lengths of the inner edges to differ as little as possible from each other.

To calculate the shortest distance $d(u, v)$, the *Bellman-Ford Algorithm* is used. The Bellman-Ford algorithm finds the shortest path between a starting vertex $u$ and all other vertices of the graph. The algorithm works both for unweighted and for weighted graphs. The input for the algorithm is the vertex $u$ and data on the weights of the edges, for example, in the form of an *adjacency matrix.* This is a matrix $D$, such that the entry $D_{uv}$ is some constant $d_{uv}$ whenever $u$ and $v$ in $V$ are adjacent and 0, otherwise. The diagonal of $D$ consists solely of zeros. $d_{uv} = 1$ for all edges $uv \in E$ if the graph is unweighted.

**2.2. The surface area of the inner faces.** Finally, a somewhat different method to calculate the positions is presented. Instead of minimizing the energy as a function of final and desired distances, we bring forward a function that minimizes the differences between the surface area of all inner faces.

Recall that we are working in $\mathbb{R}^2$. Since all faces are convex polygons, the areas can be calculated using the fact that a convex polygon $P_n = (x_1, ..., x_n)$ can be written as a union of $n - 2$ triangles, namely, $\Delta x_1x_2x_3$, $\Delta x_1x_3x_4$, ..., $\Delta x_1x_{n-1}x_n$. These triangles will be further referred to as *inner triangles.* See Figure 3.1 for an elementary example. The area $T$ of a triangle with edges of length $a$, $b$ and $c$, is calculated using Heron's formula:

$$T = \sqrt{s(s - a)(s - b)(s - c)},$$

where $s$ is the semiperimeter of the triangle, calculated as $\frac{1}{2}(a + b + c)$. and the length of an edge is calculated as the Euclidean distance between the two endpoints of that edge.
Subsequently, the area of a convex polygon $P_n = (x_1, ..., x_n)$ can be computed as the sum of the areas of the n-2 inner triangles of $P_n$:

$$A(P_n) = \sum_{i=1}^{n-2} T_i \tag{3.6}$$

Now, for a graph $G$, the faces $F_0, F_1, ..., F_l$ of G are defined as the faces of the original Tutte embedding, where $F_0$ is the outer face and $F_i$ for $1 \leq i \leq l$ are the inner faces. The mean surface area of the inner faces is calculated as the area of $F_0$ divided by $l$, the number of inner faces. The energy function is defined as follows:

$$\mathcal{E}_F(p_\alpha) = \sum_{i=1}^{l}(\frac{A(F_0)}{l} - A(F_i))^2.$$

It appears in the formula as if $\mathcal{E}_F$ does not depend on $p_\alpha$. This is not the case however, as the surface area of the inner faces depends on the positions of the vertices as opposed to each other.

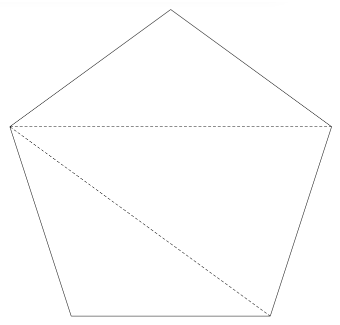


FIGURE 3.1. A pentagon as a union of three inner triangles

## 3. A selection of polytopes

In the next chapter, the results of the different energy functions, as discussed above, will be presented. But first, this section elaborates on the graphs that will be used as input.

Recall that in Tutte's theorem, the graphs are required to be planar and 3-connected. By a theorem of Steinitz (1922), these graphs are isomorphic to the 1-skeleton of a 3-polytope. A restatement of Steinitz' theorem, by Grünbaum, implies that these types of graphs can be converted into convex polyhedra (Duijvestijn & Federico, 1981). 3-connected planar graphs are therefore referred to as *polyhedral graphs*. Unfortunately, Steinitz' theorem has not been proven for polytopes, the generalization of polyhedra to any dimension.

Hence, the input graphs will be a selection of several polyhedral graphs. These polyhedral graphs are highly symmetric, under rotation and/or under reflection. It is important in graph drawing to maintain a symmetric structure when it exists. As symmetry is a valuable characteristic (Kamada & Kawai, 1989).

In Chapter 1, the Tutte embeddings of three polyhedra were presented. In the next chapter, the embeddings of some polyhedral graphs will be calculated as a consequence of the different energy functions. This will also be done for other polyhedral graphs.

CHAPTER 4

# Embedding polyhedral graphs

Each embedding of a graph has a value of the corresponding energy function. But, for different energy functions, it would not make sense to compare the values, since they're calculated in a different way. For completeness, the energy of the embedding will also be calculated as the energy of the Tutte embedding:

$$\mathcal{E}(p_\alpha) = \sum_{uv \in E} \|p_\alpha(u) - p_\alpha(v)\|^2.$$

However, this does not indicate anything about the quality of the embedding. In this chapter, we will see that a lower energy does not imply a 'better' embedding. Partly because, as was shortly mentioned in the introduction, the quality of an embedding is subject to a number of factors, such as individual context and structure of the graph, intended use and aesthetics. This research will not assess the quality of an embedding, it merely reports on the results of the various energy functions.

All energy functions are functions of $p_\alpha$. The value of the energy function also depends on the side length of the display. The vertices are placed inside a square display with fixed length $L_0 = 700$. If another value is taken for $L_0$, the energies corresponding to the different values of $L_0$ cannot be compared.

## 1. Minimizing $\mathcal{E}_{k,m}$ and $\mathcal{E}_d$

For $k \in \{-1, -2, -3\}$, recall the function

$$\mathcal{E}_{k,m}(p_\alpha) = \sum_{\substack{u,v \in V \\ u \vee v \in V \setminus V_0}} c(u,v)^k \cdot (\|p_\alpha(u) - p_\alpha(v)\|^2 - l_{uv}^2)^2. \tag{4.1}$$

This function is tested on the 1-skeleton of the icosahedron. Due to lack of computational power, it was not possible to test this function on more polyhedral graphs. The limit to the number of function evaluations of the COBYLA method was set at 100000.

**$l_{uv}$ as a function of $d(u,v)$.** First, we consider $c(u,v) = d(u,v)$ and

$$l_{uv} = \frac{m \cdot L_0}{max_{u,v \in V} d(u,v)} \cdot d(u,v).$$

The time complexity of $\mathcal{E}_{k,m}$ is $\mathcal{O}(n^2)$.[1] In Table A.1 in Appendix A, the energy values are presented for $m \in \{1, 0.75, 0.65, 0.5\}$.

The values of $\mathcal{E}_{k,m}$ are very large due to the value of $L_0$ and because every vertex combination is considered (except when both vertices are part of the outer face). These numbers an sich don't say much about the embedding. Figure A.1 in Appendix A shows the embeddings corresponding to the values in Table A.1.

[1]Note that this is only the time complexity of the energy function itself, not of the full program.

Apart from the embedding corresponding to $\mathcal{E}_{-1,1}$, all embeddings have the same structure as the Tutte embedding. For $m \in \{1, 0.75\}$, some vertices are very close to the outer face. For $k \in \{-1, -2\}$ and $m \in \{0.65, 0.5\}$; some edges are very short. These embeddings might therefore be more difficult to read with the naked eye.

For $\mathcal{E}_{-3,m}$ and $m \in \{0.65, 0.5\}$; the optimizer found an optimal solution. Hence, for $\mathcal{E}_{-3,0.65}$ and $\mathcal{E}_{-3,0.5}$, the program was run four more times. The energy values are presented in Table 4.1. In the table, it is shown that for both values of $m$, the difference between the values of the local minima is negligible ($< .001\%$).

| $\mathcal{E}_{k,m}$ | $\mathcal{E}$ | function evaluations | $\mathcal{E}_{k,m}$ | $\mathcal{E}$ | function evaluations |
|---|---|---|---|---|---|
| 52944642316 | 2114886 | 80688 | 69866593921 | 2045579 | 79809 |
| 52944642353 | 2114887 | 95038 | 69866593943 | 2045592 | 55902 |
| 52944642447 | 2114882 | 86448 | 69866593959 | 2045571 | 65620 |
| 52944642720 | 2114886 | 82693 | 69866594353 | 2045577 | 78724 |
| 52944642798 | 2114887 | 92936 | 69866595855 | 2045571 | 70064 |

TABLE 4.1. Values of local minima of $\mathcal{E}_{-3,0.65}$ (left) and $\mathcal{E}_{-3,0.5}$ (right).

In Figure 4.1, the embeddings corresponding to the minimum local minima of $\mathcal{E}_{-3,0.65}$ (left) and $\mathcal{E}_{-3,0.5}$ (right) are shown. The embeddings have the same structure as the Tutte embedding. Therefore, we could compare the value of $\mathcal{E}$ in the table with the energy value of the Tutte embedding of the icosahedron. For the Tutte embedding, the value of the total energy is $\mathcal{E} = 2004545$. All values of $\mathcal{E}$ in Table 4.1 are higher than the energy value of the Tutte embedding. This is due to the fact that the inner edges are longer than in the Tutte embedding. The local minima of $\mathcal{E}_{-3,0.5}$ are closest to the energy value of the Tutte embedding.

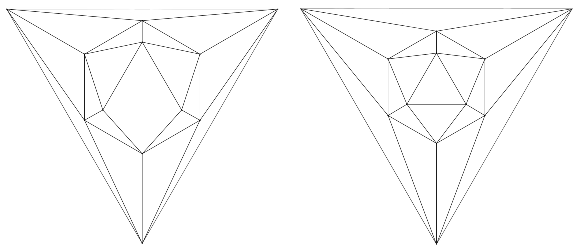


FIGURE 4.1. Embeddings of local minimum of $\mathcal{E}_{-3,0.65}$ (left) and $\mathcal{E}_{-3,0.5}$ (right)

**$l_{uv}$ as the mean length of the inner edges.** Now, we consider the desired distance $l_{uv}$ as the square root of the mean length $M$ of all inner edges $E_{in}$, multiplied by scaling constant $d$. That is, $l_{uv} = \sqrt{d \cdot M}$, where

$$M = \frac{1}{|E_{in}|} \sum_{\substack{u,v \in V \\ uv \in E_{in}}} \|p_\alpha(u) - p_\alpha(v)\|^2.$$

In this case, we let the summation in $\mathcal{E}_{k,m}$ only consider the inner edges, so for all $u, v \in V$ we set $c(u,v) = 1$ (or $k = 0$). The new function, denoted as $\mathcal{E}_d(p_\alpha)$, looks as follows:

$$\mathcal{E}_d(p_\alpha) = \sum_{\substack{u,v \in V \\ u \vee v \in V \setminus V_0}} (\|p_\alpha(u) - p_\alpha(v)\|^2 - d \cdot M)^2. \tag{4.2}$$

Denote the degree of an arbitrary vertex as $w$. $\mathcal{E}_d$ has a time complexity of $\mathcal{O}(w \cdot n + m)$. The program was run for $d = (0.3, 0.4, ..., 1)$. The energy values are presented in Table 4.2.

| d | $\mathcal{E}_d$ | $\mathcal{E}$ | function evaluations |
|---|---|---|---|
| 1.0 | 20946040380 | 2313247 | 100000 |
| 0.9 | 21497488571 | 2306995 | " |
| 0.8 | 23091036766 | 2289290 | " |
| 0.7 | 25573529434 | 2241381 | " |
| 0.6 | 28656604484 | 2235747 | " |
| 0.5 | 32929407096 | 2189918 | " |
| 0.4 | 37541978132 | 2148955 | " |
| 0.3 | 42725555006 | 2127882 | " |

TABLE 4.2. Energy values corresponding to $\mathcal{E}_d$ for $d = (0.3, 0.4, ..., 1)$.

See Figure A.2 in Appendix A for the embeddings corresponding to the values in the table above. For $d > 0.5$, some edges are very short compared to the rest of the edges. These embeddings do not maintain the symmetry by rotation of 120 degrees, only by reflection in one altitude. For $d \in \{0.3, 0.4\}$, the structure of the embeddings is similar to the Tutte embedding.

The embeddings of the energy functions $\mathcal{E}_{k,m}$ (except for $\mathcal{E}_{-1,1}$) and $\mathcal{E}_d$ for $d \in \{0.3, 0.4\}$ are very similar to the Tutte embedding, in the sense that they have the same structure and are symmetric under rotation of 120 degrees. The embeddings of $\mathcal{E}_{-3,m}$ for $m \in \{0.65, 0.5\}$ seem slightly unnatural. The face in the middle is large compared to the faces adjacent to it. The embeddings of $\mathcal{E}_d$, for $d \in \{0.3, 0.4\}$ come across as more natural.

However, whether one embedding is better than the other, completely depends on its application. It could also depend on the computational power of the user. The time complexity of $\mathcal{E}_d$ is $\mathcal{O}(w \cdot n + m)$, which is usually[2] faster than $\mathcal{O}(n^2)$, the time complexity of $\mathcal{E}_{k,m}$. However, $\mathcal{E}_d$ doesn't reach a local minimum in 100000 function evaluations, while it is unclear in what amount these values differ from a local minimum. It should therefore be tested how many function evaluations are needed to reach a local minimum, or in what amount the COBYLA method makes a difference after 100000 evaluations.

## 2. Minimizing $\mathcal{E}_F$

The second function is as discussed in Section 2.2. Let $G$ a polyhedral graph with faces $F_0, ..., F_l$, $l \in \mathbb{N}$. The area of a face $F_i$ for $i \in \{1, ..., l\}$ is given by $A(F_i)$ as defined in (3.6). The energy is

[2]considering the polyhedral graphs

calculated as follows:

$$\mathcal{E}_F(p_\alpha) = \sum_{i=1}^{l} (\frac{A(F_0)}{l} - A(F_i))^2. \tag{4.3}$$

The time complexity of $\mathcal{E}_F$ is $\mathcal{O}(l \cdot n + l)$. The polyhedral graphs that will be used as input are the 1-skeleton of the icosahedron, the cuboctahedron, the dodecahedron and the rhombic dodecahedron. Note that from now on, the 1-skeleton is always implied when a polyhedron is mentioned.

**2.1. Icosahedron.** The Tutte embedding of the icosahedron was presented in Figure 1.2. The icosahedron has 12 vertices, 30 edges and 20 faces. In Table 4.3, the values of twenty local minima of $\mathcal{E}_F$ are presented. The table shows that for the icosahedron, the energy function often reaches a minimum value below two. All twenty optimizations were successful.

| $\mathcal{E}_F$ | $\mathcal{E}$ | function evaluations | $\mathcal{E}_F$ | $\mathcal{E}$ | function evaluations |
|---|---|---|---|---|---|
| ◊ 0.244957 | 2555576 | 20542 | * 1.752328 | 2342381 | 28901 |
| □ 0.381891 | 2476721 | 12380 | * 1.805990 | 2342320 | 20803 |
| ◊ 0.407046 | 2555572 | 11322 | * 1.846196 | 2320984 | 21664 |
| □ 0.560016 | 2476703 | 8364 | □ 2.103554 | 2440271 | 38487 |
| * 0.781168 | 2321010 | 21192 | □ 2.498724 | 2440259 | 30731 |
| ◊ 1.013991 | 2590656 | 27284 | □ 2.623326 | 2440256 | 28999 |
| * 1.157301 | 2320993 | 16457 | □ 2.931048 | 2440250 | 27942 |
| * 1.249863 | 2320994 | 18774 | * 3.143116 | 2362445 | 24540 |
| * 1.299521 | 2321009 | 19612 | □ 3.747607 | 2440230 | 33813 |
| * 1.623418 | 2321006 | 22612 | □ 4.121620 | 2440225 | 35961 |

TABLE 4.3. Energy values of twenty local minima of $\mathcal{E}_F$ for the icosahedron.

The results of the embedding can be subdivided in three categories regarding the embedding. The first category, represented by a lozenge, is characterized by having one outer vertex for which the middle inner neighbour is further away than the other two inner neighbours, such that the outer vertex and its three inner neighbours form a diamond shape. The second category is represented by a square. This category contains an irregular triangle that has an inner triangle. The asterisk represents the third and last category. This category has a tilted regular triangle in the middle and all other faces are placed such that the embedding is approximately symmetric by a 120-degree rotation. The other two categories are not symmetric by rotation and none of the categories are symmetric by reflection in one of the altitudes.

Of each category, an embedding is presented in Figure 4.2. On the left, the characterizing outer vertex is in the top right corner. The middle embedding has two triangles in the middle of the embedding and two triangles adjacent to the right middle triangle. These four form the characterizing triangle with an inner triangle. The right embedding has a symmetric structure with a tilted regular triangle in the middle.

The two leftmost embeddings have the lowest values of $\mathcal{E}_F$ and the four most optimal embeddings are in these categories. However, in these embeddings it is not apparent that the original graph has a symmetric structure, while it was mentioned that embeddings of a symmetric graph should maintain symmetry. In the third embedding and the corresponding category, this symmetry is respected. This shows that a more optimal solution $p$ does not necessarily give a more desired embedding.

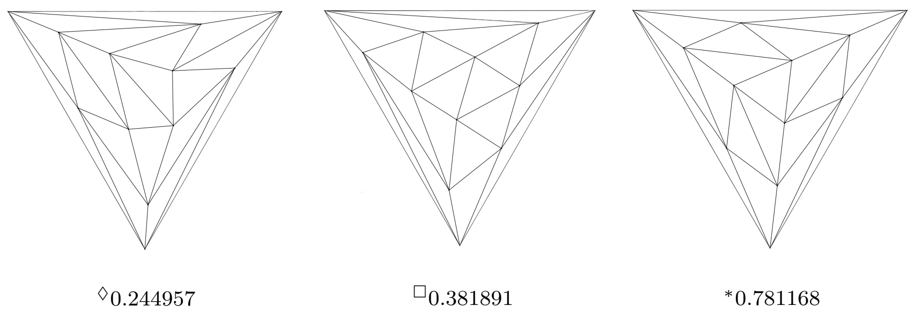


FIGURE 4.2. Three embeddings of the icosahedron, with corresponding local minimum values of $\mathcal{E}_F$

**2.2. Cuboctahedron.** A cuboctahedron has 12 vertices, 24 edges and 14 faces, of which 8 have length three and the remaining 6 have length four. For both face lengths, the program was run once with 1000000 function evaluations and five times with 100000 evaluations. Unfortunately, none returned a success. The energy values are presented in Table 4.4.

| $\mathcal{E}_F$ | $\mathcal{E}$ | function evaluations |
|---|---|---|
| 1245663 | 1809625 | 1000000 |
| 2665713 | 1806329 | 100000 |
| 2718840 | 1805798 | ” |
| 2829952 | 1805914 | ” |
| 2941403 | 1805188 | ” |
| 3042463 | 1805290 | ” |

| $\mathcal{E}_F$ | $\mathcal{E}$ | function evaluations |
|---|---|---|
| 119371736 | 2297361 | 1000000 |
| 126766437 | 2283895 | 100000 |
| 126918985 | 2283659 | ” |
| 127126078 | 2283367 | ” |
| 127828289 | 2282085 | ” |
| 128011536 | 2281632 | ” |

TABLE 4.4. Minimization results of $\mathcal{E}_F$ for the cuboctahedron with $|F_0| = 3$ (left) and $|F_0| = 4$ (right).

In the left table, the result of 1000000 function evaluations is less than half of the lowest result of 100000 evaluations. In the right table, this difference is less than six percent. This shows that increasing the number of function evaluations tenfold, does not necessarily result in a significant improvement in minimizing the value of the energy function.

Contrary to the icosahedron, for each length of the outer face, all embeddings of the cuboctahedron corresponding to the values in Table 4.4 are approximately symmetric under rotation by $360/|F_0|$ degrees. In Figures 4.3 and 4.4, the embeddings corresponding to the 1000000 function evaluations are presented.

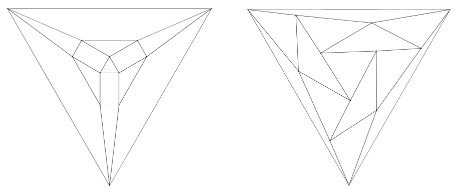


FIGURE 4.3. Embeddings of Tutte (left) and the lowest value of $\mathcal{E}_F$ (right).

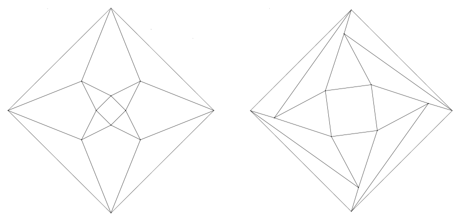


FIGURE 4.4. Embeddings of Tutte (left) and the lowest value of $\mathcal{E}_F$ (right).

**2.3. Dodecahedron.** A dodecahedron has has 20 vertices, 30 edges and 12 faces of length five. Due to long runtime, the program was run only twice. Both minimizations delivered a local minimum. The energy values are presented in Table 4.5 and the embedding corresponding to the lowest value of $\mathcal{E}_F$ is shown in Figure 4.5, together with the Tutte embedding of the dodecahedron.

| $\mathcal{E}_F$ | $\mathcal{E}$ | function evaluations |
|---|---|---|
| 0.755094 | 1293957 | 14421 |
| 1.248657 | 1321090 | 14707 |

Table 4.5. Local minima of $\mathcal{E}_F$ for the dodecahedron

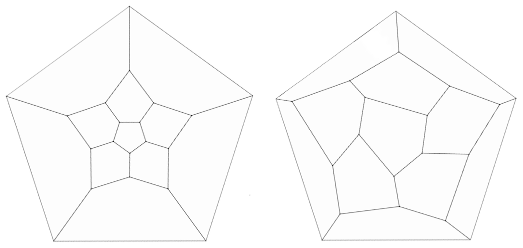


Figure 4.5. Tutte embedding (left) and the embedding corresponding to the lowest value of $\mathcal{E}_F$ (right).

The embedding of $\mathcal{E}_F$ in Table 4.5 is not symmetric. However, it could be that there are other local minima that have a more symmetric structure. It is also possible to add a component to the function that decreases the difference in length of certain edges, for example the five inner edges that are connected to the outer face.

**2.4. Rhombic dodecahedron.** This graph has 14 vertices, 24 edges and 12 faces of length four. The energies of five local minima of $\mathcal{E}_F$ are displayed in Table 4.6.

| $\mathcal{E}_F$ | $\mathcal{E}$ | function evaluations |
|---|---|---|
| 0.062918 | 1630478 | 5405 |
| 0.493646 | 1665179 | 4291 |
| 0.574753 | 1599025 | 7012 |
| 0.949214 | 1614787 | 4212 |
| 1.204026 | 1602275 | 5926 |

Table 4.6. Energy values of five local minima of $\mathcal{E}_F$ for the rhombic dodecahedron.

The values of $\mathcal{E}_F$ are quite close to zero. However, in Figure 4.6, it can be seen that none of the embeddings are symmetric. They do have some symmetric characteristics, for example, the embedding corresponding to $\mathcal{E}_F = 0.574753$ is almost symmetric in the y-axis. By adding constraints on the $x$ and/or $y$ coordinates of specific vertices or on the distance between two vertices, the embedding could be forced to be (more) symmetric. For example, a constraint could be added that enforces vertices with equivalent $y$-coordinates in the Tutte embedding, to maintain this characteristic in the new embedding that is created by minimizing $\mathcal{E}_F$.

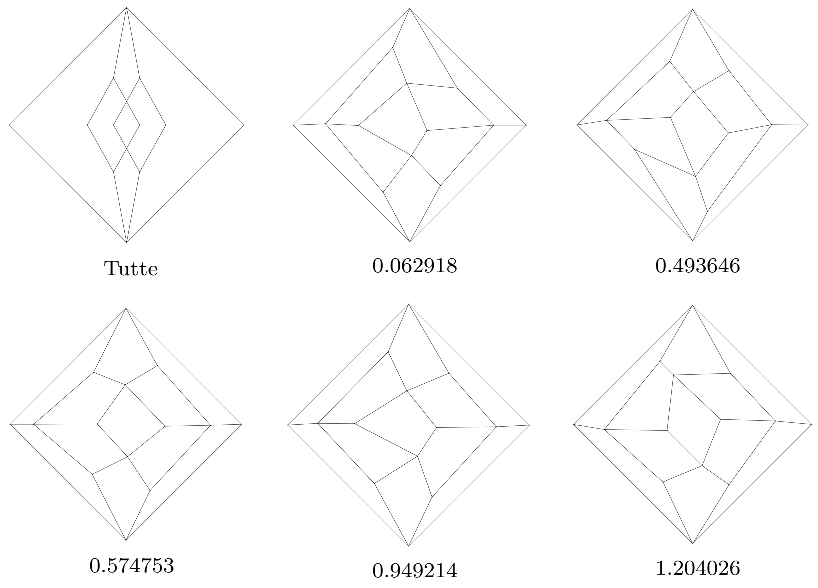


Figure 4.6. The Tutte embedding and five embeddings corresponding to the local minima in Table 4.6.

We have seen several embeddings corresponding to different energy functions. The energy functions $\mathcal{E}_{k,m}$ and $\mathcal{E}_d$ yielded embeddings that have the same structure as the Tutte embedding. The embeddings of the icosahedron are mostly symmetric by a 120-degree rotation and reflection in the altitudes. Depending on the input graph and choice of outer face, these functions will most likely maintain the structure of the Tutte embedding and its symmetries.

$\mathcal{E}_F$ yields different embeddings than the Tutte embedding, some of which are symmetric by rotation of 120 degrees, but not by reflection, depending on the input. However, by adding extra constraints, it is possible to manipulate the embedding to satisfy the users needs while maintaining compliance with Tutte's theorem. One could also add factors that distinguishes between face lengths or between whether a face is adjacent to the outer face. For example, in the embeddings of the icosahedron and the cuboctahedron with outer face length three, the three faces adjacent to the outer face are more elongated than all other faces. Consequently, they are a bit more difficult to distinguish with the naked eye.

In addition, it turns out that the lowest energy value does not always yield the best embedding. Moreover, due to the way the optimization problem was encoded, the number of function evaluations could be limited while still yielding an embedding that is compliant with Tutte's theorem. This could also be less time-consuming if an approximation to a local minimum is satisfactory.

# Conclusion

The Tutte embedding is a way to draw a 3-connected planar graph, where every vertex is positioned as the barycenter of its neighbours. Such an embedding can be made easily by solving a linear system that is obtained from an optimization problem, as discussed in Chapter1. Tutte proved that every face of the embedding is drawn as a convex polygon. The main goal of this research was to change the optimization problem corresponding to the Tutte embedding, so that the embedding of a local minimum of the objective function remains compliant with Tutte's theorem.

To achieve this, a new optimization problem was created in such a way that every iteration would be compliant with Tutte's theorem. For the optimization problem, three different objective functions were tested: $\mathcal{E}_{k,m}$, $\mathcal{E}_d$ and $\mathcal{E}_F$, given in Chapter 4. The first two resulted in embeddings that have a similar structure as the Tutte embedding, maintaining the symmetric qualities of the polyhedral graph. By changing the values for $k$, $m$ and $d$, or by changing $l_{uv}$ or $c(u,v)$, the embedding can be influenced based on personal or contextual preferences.

The embeddings of $\mathcal{E}_F$ had a completely different structure than the Tutte embedding. Unfortunately not always maintaining the symmetric properties of the polyhedral graph, but these properties could be enforced by adjusting the energy function or the constraints. Some of the embeddings did have symmetric characteristics, which gives an interesting view on the possibilities regarding the subject at hand.

A distinction between the energy functions is that $\mathcal{E}_F$ is more generally deployable. The other two functions are subject to a desired edge length, which causes a possible difference in optimal values of $k$, $m$ and $d$, depending on the polyhedral graph. These functions therefore need an extra check after creating an embedding.

Returning to the main goal of this research. Whether the quality of an embedding is sufficient, depends completely on the context, application or other factors. But this research showed that there are various ways of drawing 3-connected planar graphs while maintaining compliance with Tutte's theorem. One could create embeddings using one of the given functions, but is also free to add extra constraints or to create a more personal energy function according to ones needs.

# Discussion

A huge challenge for this research was the lack of computational power. This precluded a larger data set on which more analysis could have been executed. Three elements would have contributed to this research. Firstly, having more local minima of an energy function creates a more detailed image of the function and gives more insight in the embeddings. Secondly, a faster runtime gives a possibility to run the code more often with a different number of function evaluations. Then, per polyhedral graph, it could be investigated around what number of evaluations, the minimization stagnates and if this correlates, for example, with the amount of vertices, edges or faces. Lastly, the functions that depend on a desired edge length could have been tested on more polyhedral graphs.

Although the results that were created for this research were sufficient to answer the main question, there are a lot of graphs that are left untested. Think of more complex polyhedra such as the icosidodecahedron, the truncated dodecahedron and rhombicosidodecahedron (Figure 1.2). Besides graphs, other variations on the problem were also mentioned. It is possible to add extra constraints to influence positions of vertices, to make a distinction between face lengths in the function $\mathcal{E}_F$ or to some other extent. It is also possible to change the energy functions by changing the desired edge length $l_{uv}$ in $\mathcal{E}_{k,m}$, the desired surface area of a face in $\mathcal{E}_F$ or by adding scaling factors that depend on the variables or graph components.

When it comes to further research, this research opened some doors. One could study the amount of symmetry in the embeddings. Moreover, an interesting dimension could be the interaction between an energy function and characteristics of the graph, such as the length of the outer face, the number of inner faces, vertex degrees and more. Another option is to investigate what the influence would be of fixing the outer face as an irregular strictly convex polygon. Furthermore, this research could be combined with a specific sector or type of application of graph embeddings. Subsequently, the embeddings can be qualified based on contextual, aesthetical or other properties.

# Bibliography

Brandes, U. (2001). Drawing on physical analogies. In Lecture notes in computer science (pp. 71–86). https://doi.org/10.1007/3-540-44969-8_4

Cohen, J. D. (1997). Drawing graphs to convey proximity. *ACM Transactions on Computer-Human Interaction, 4*(3), 197–229. https://doi.org/10.1145/264645.264657

Diestel, R. (2017). *Graph theory.* Graduate Texts in Mathematics 173, (5th ed.), Springer, Berlin.

Duijvestijn, A. J. W., & Federico, P. J. (1981). The number of polyhedral (3-connected planar) graphs. *Mathematics of Computation, 37*(156), 523-532. https://doi.org/10.1090/s0025-5718-1981-0628713-3

Kamada, T., & Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters, 31*(1), 7–15. https://doi.org/10.1016/0020-0190(89)90102-6

Kaufmann, M., & Wagner, D. (Eds.) (2003). Drawing graphs: methods and models. In Springer eBooks (p. 312). https://doi.org/10.1007/3-540-44969-8

Kuratowski, C. (1930). *Sur le problème des courbes gauches en Topologie.* Fundamenta Mathematicae, 15(1), 271–283. https://doi.org/10.4064/FM-15-1-271-283

Lovász, L. (2009). *Geometric Representations of Graphs.* [PDF] https://webspace.science.uu.nl/~neder003/2MMD30/lecturenotes/L3/geomrep_lovasz.pdf

Purchase, H. (1997). Which aesthetic has the greatest effect on human understanding? In *Lecture notes in computer science* (pp. 248–261). https://doi.org/10.1007/3-540-63938-1_67

Steinitz, E. (1916). Polyeder und Raumeinteilungen, von Ernst Steinitz. In *Encyklopädie der mathematischen Wissenschaften.* 1-13

Tutte, W.T. (1963). How to draw a graph. *Proceedings of the London Mathematical Society, 13*, 743–767.

APPENDIX A

# Results of $\mathcal{E}_{k,m}$ and $\mathcal{E}_d$

| k | m | $\mathcal{E}_{k,m}$ | $\mathcal{E}$ | function evaluations | success |
|---|---|---|---|---|---|
| -1 | 1 | 298919546745 | 2679819 | 100000 | false |
| -1 | 0.75 | 125389869053 | 2288921 | 100000 | false |
| -1 | 0.65 | 122006938239 | 2148199 | 100000 | false |
| -1 | 0.5 | 142832885063 | 2050500 | 100000 | false |
| -2 | 1 | 142461811712 | 2598785 | 100000 | false |
| -2 | 0.75 | 72296680865 | 2232266 | 100000 | false |
| -2 | 0.65 | 75173359253 | 2128887 | 100000 | false |
| -2 | 0.5 | 95099678129 | 2047445 | 100000 | false |
| -3 | 1 | 77325010327 | 2546311 | 100000 | false |
| -3 | 0.75 | 47681695487 | 2196581 | 100000 | false |
| -3 | 0.65 | 52944642353 | 2114887 | 95038 | true |
| -3 | 0.5 | 69866593959 | 2045571 | 65620 | true |

TABLE A.1. Results of minimizing $\mathcal{E}_{k,m}$ for $k \in \{-1,-2,-3\}$ and $m \in \{1, 0.75, 0.65, 0.5\}$

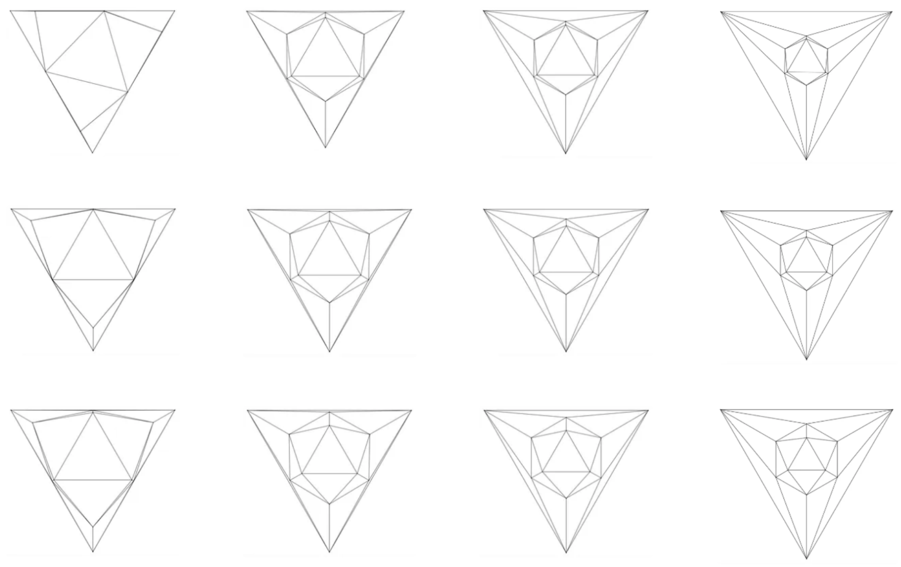


Figure A.1. Embeddings corresponding to the values in Table A.1. Row values from top to bottom corresponding to k = -1,-2,-3; Column values from left to right corresponding to m = 1, 0.75, 0.65, 0.5.

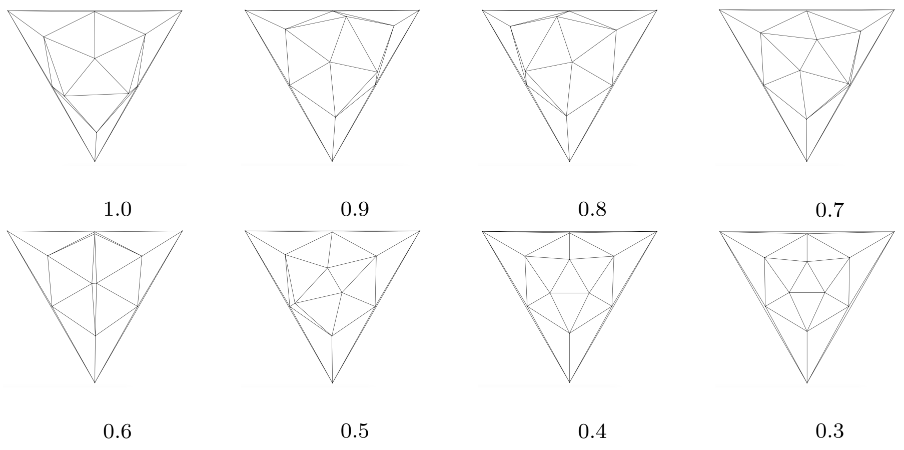


Figure A.2. Embeddings corresponding to the values in Table 4.2.

APPENDIX B

# Python code: Tutte embedding

```
import CanvasApp as CA

from math import sin, cos, sqrt, trunc

import numpy as np

import scipy as sp

import scipy.linalg

from tkinter import *

from random import randint



class Graph:

    def __init__(self, nverts, medges):

        self.nverts = nverts

        self.medges = medges

        self.neighbors = [ set() for i in range(nverts) ]

        self.faces = []



    def add_edge(self, u, v):

        self.neighbors[u].add(v)

        self.neighbors[v].add(u)
```

```
def add_face(self, corners):
    face = set(corners)
    self.faces.append(face)

def face_lengths(self):
    lengths = set()
    for face in self.faces:
        if len(face) not in lengths:
            lengths.add(len(face))

    return lengths

def sort_face(self, face):
    cycle = []
    current = face.pop()
    cycle.append(current)
    while len(face) != 0:
        f = face.pop()
        if f in self.neighbors[current]:
            cycle.append(f)
            current = f
        else:
            face.add(f)

    return cycle

def choose_outer_face(self, length):
    for face in self.faces:
            if len(face) == length:

                sorted_face = self.sort_face(face)

                return sorted_face

    def make_dict(self):
        face_dict = dict()
        for f in self.faces:
            l = len(f)
            if l in face_dict:
                face_dict[l].append(f)
            else:
                face_dict[l] = [f]

        return face_dict


def read_graph(filename):
    with open(filename, ’r’) as infile:
        lines = [ s for s in infile ]

    n = int(lines[0])
    m = 0
    G = Graph(n, m)

    for s in lines[1:]:
        if s[0] == ’F’:
            face = set()
            fields = s.split()

            for i in range(1, len(fields)):
                face.add(int(fields[i]))

            G.add_face(face)

        if s[0] == 'E':
            m += 1
            fields = s.split()

            G.add_edge(int(fields[1]), int(fields[2]))

    G.medges = m

    return G


class TutteEmbedding:
    def __init__(self, G, F0):
        self.G = G
        self.n = G.nverts

        width = height = 800
        radius = 350
        ox = width / 2
        oy = height / 2
        # Positions of all vertices in F0
        self.F0_positions = {}
        F0 = list(F0)

        pi = 3.14159265
        theta0 = pi / 2
        for i, v in zip(range(len(F0)), F0):
            self.F0_positions[v] = (ox + radius*cos(theta0 + i*2*pi/len(F0)),
                                    oy + radius*sin(theta0 + i*2*pi/len(F0)))
        self.F0 = set(F0)


    def standard_vector(self):

        return np.ones(self.G.nverts ** 2)


    def compute_alphas(self, w):
        n = self.n
        W = np.zeros((n, n))

        for u in range(n):
            for v in range(n):
                W[u, v] = w[u * n + v]

        return W
```

```
def compute_positions(self, W):
    F0 = self.F0
    G = self.G
    n = self.n

    # Create system.
    # Coordinates of u are in positions 2*u and 2*u + 1.
    A = np.zeros((2 * n, 2 * n))
    b = np.zeros(2 * n)

    for u in range(n):
        if u not in F0:
            total_weight = 0
            for v in G.neighbors[u]:
                A[2*u, 2*v] = -W[u][v]
                A[2*u + 1, 2*v + 1] = -W[u][v]

                total_weight += W[u][v]

            A[2*u, :] /= total_weight
            A[2*u + 1, :] /= total_weight

        else:
            (x, y) = self.F0_positions[u]
            b[2*u] = x
            b[2*u + 1] = y
            A[2*u, 2*u] = A[2*u + 1, 2*u + 1] = 1

        # Compute LU decomposition and compute positions
        self.lu_dec = sp.linalg.lu_factor(A)
        sol = sp.linalg.lu_solve(self.lu_dec, b)

        # Assemble array with coordinates of each vertex
        x = [ (sol[2 * u], sol[2 * u + 1]) for u in range(n) ]

        return x, A



def main():
    files = ['cuboctahedron', 'dodecahedron', 'icosahedron',
             'rhombic_dodecahedron', 'icosidodecahedron',
             'rhombicosidodecahedron']

    f = 5
    name = files[f]
    graph = name + '.dat'
    G = read_graph(graph)

    faces_dict = G.make_dict()
    print("Face lengths: ", set(faces_dict.keys()))
    print(faces_dict)
    l, f = input("Type length and face number: ", ).split()
    f0 = faces_dict.get(int(l))[int(f)]
    F0 = G.sort_face(f0)
    print('G is a ', name)

    print('G has ', G.nverts, ' vertices and', G.medges, ' edges')

    print('F0 = ', F0)


    # The embedding

    embedding = TutteEmbedding(G, F0)


# Original Tutte Embedding and energy

    x, A = embedding.compute_positions(np.ones((G.nverts, G.nverts)))

    E = 0

    for u in range(G.nverts):

        for v in rG.neighbors[u]:

            if u < v:

                norm_sq =  (x[u][0] - x[v][0]) ** 2 + (x[u][1] - x[v][1]) ** 2


                E += norm_sq

    print(E)


# make drawing

    root = Tk()

    title = str(name) + '; F0 = ' + str(F0) + '; Tutte = ' + str(trunc(E))

    root.title(title)

    app = CA.App(title, G.nverts, G.neighbors, x, root)

    root.mainloop()


main()
```

APPENDIX C

# Python code: Convex embeddings

```
import CanvasApp as CA

from math import sin, cos, sqrt, trunc

import numpy as np

import scipy as sp

import scipy.linalg

import sys

from tkinter import *

from random import randint


class Graph:

    def __init__(self, nverts, medges):

        self.nverts = nverts

        self.medges = medges

        self.faces = []

        self.sorted_faces = []

        self.neighbors = [ set() for i in range(nverts) ]

        self.adj_matrix = [[0] * nverts for i in range(nverts)]

        self.BF = [[] * nverts for i in range(nverts)]



    def add_edge(self, u, v):
        self.neighbors[u].add(v)
        self.neighbors[v].add(u)
        self.adj_matrix[u][v] = 1
        self.adj_matrix[v][u] = 1

    def add_face(self, corners):
        face = set(corners)
        self.faces.append(face)

    def sort_face(self, fac):
        cycle = []
        face = fac.copy()
        current = face.pop()
        cycle.append(current)
        while len(face) != 0:
            f = face.pop()
            if f in self.neighbors[current]:
                cycle.append(f)
                current = f
            else:
                face.add(f)

        return cycle

    def choose_outer_face(self, length):
        for face in self.faces:
            if len(face) == length:
                sorted_face = self.sort_face(face)

        return sorted_face

    def sort_faces(self, F0):
        self.sorted_faces.append(F0)

        for f in self.faces:
            if f != set(F0):
                face = self.sort_face(f)
                self.sorted_faces.append(face)

        return self.sorted_faces

    def make_dict(self):
        face_dict = dict()
        for f in self.faces:
            l = len(f)
            if l in face_dict:
                face_dict[l].append(f)
            else:
                face_dict[l] = [f]

        return face_dict



# returns list of shortest distances to all vertices from vertex u
    def bellman_ford(self, source, matrix):
        distances = [float('inf')] * self.nverts
        distances[source] = 0

        for i in range(self.nverts - 1):
            for u in range(self.nverts):
                for v in self.neighbors[u]:
                    if distances[v] > distances[u] + matrix[u][v]:
                        distances[v] = distances[u] + matrix[u][v]

        return distances



    def BF_array(self, matrix):
        return [self.bellman_ford(u, matrix) for u in range(self.nverts)]



def read_graph(filename):
    with open(filename, 'r') as infile:
        lines = [ s for s in infile ]

    n = int(lines[0])
    m = 0
    G = Graph(n, m)

    for s in lines[1:]:
        if s[0] == 'F':
            face = set()
            fields = s.split()
            for i in range(1, len(fields)):
                face.add(int(fields[i]))

            G.add_face(face)

        if s[0] == 'E':
            m += 1
            fields = s.split()

            G.add_edge(int(fields[1]), int(fields[2]))

    G.medges = m

    G.BF = G.BF_array(G.adj_matrix)

    return G


class TutteEmbedding:
    def __init__(self, G, F0, fun, sorted_faces):
        self.G = G
        self.n = G.nverts
        self.fun = fun
        self.F0 = F0
        self.sorted_faces = sorted_faces

    # Positions of the vertices in F0
```

```
        self.F0_positions = {}

        radius = 350
        ox = oy = 400
        pi = 3.14159265
        phi = 2 * pi / len(F0)

        for i, v in zip(range(len(F0)), F0):
            self.F0_positions[v] = (ox + radius * cos(pi/2 + i*phi),
                                    oy + radius * sin(pi/2 + i*phi))
        self.f0 = set(F0)


    def standard_vector(self):

        return np.ones(self.G.nverts ** 2)


    def random_vector(self):
        x = np.zeros(self.G.nverts ** 2)
        for i in range(self.G.nverts ** 2):
                x[i] = np.random.random()

        return x


    def compute_alphas(self, w):
        n = self.n
        W = np.zeros((n, n))

        for u in range(n):
            for v in range(n):
                W[u, v] = 1 + w[u * n + v] ** 2

        return W


    def compute_positions(self, W):
        f0 = self.f0
        G = self.G
        n = self.n

        # Create system.
        # Coordinates of u are in positions 2*u and 2*u + 1.
        A = np.zeros((2 * n, 2 * n))
        b = np.zeros(2 * n)

        for u in range(n):
            if u not in f0:
                total_weight = 0

                for v in G.neighbors[u]:
                    A[2*u, 2*v] = -W[u][v]
                    A[2*u + 1, 2*v + 1] = -W[u][v]

                    total_weight += W[u][v]
                A[2*u, :] /= total_weight
                A[2*u + 1, :] /= total_weight

            else:
                (x, y) = self.F0_positions[u]
                b[2*u] = x
                b[2*u + 1] = y

            A[2*u, 2*u] = A[2*u + 1, 2*u + 1] = 1

        # Compute LU decomposition and compute positions
        self.lu_dec = sp.linalg.lu_factor(A)
        sol = sp.linalg.lu_solve(self.lu_dec, b)

        # Assemble array with coordinates of each vertex
        x = [ (sol[2 * u], sol[2 * u + 1]) for u in range(n) ]

        return x, A


    def triangle_area(self, u, v, w):
        a = np.sqrt((u[0] - v[0]) ** 2 + (u[1] - v[1]) ** 2)
        b = np.sqrt((v[0] - w[0]) ** 2 + (v[1] - w[1]) ** 2)
        c = np.sqrt((w[0] - u[0]) ** 2 + (w[1] - u[1]) ** 2)
        s = 0.5 * (a + b + c)

        area = np.sqrt(s*(s-a)*(s-b)*(s-c))
        return area



    def compute_areas(self, faces, positions):
        # returns list of surface area of all faces
        pos = positions

        S = []
        for face in faces:
            s = 0
            u = face[0]

            for i in range(1, len(face) - 1):
                v, w = face[i], face[i+1]
                s += self.triangle_area(pos[u], pos[v], pos[w])

            S.append(s)

        return S

    def energy(self, pos, A):
        global counter
        counter += 1
        sys.stdout.write(f"\r{counter}")
        sys.stdout.flush()

        '''Different energy functions'''
```

```
fun = self.fun

G = self.G

n = self.n

f0 = self.f0


E = 0


if fun == 'KK':

    BF = G.BF

    max_BF = 0

# calculates max length of shortest path

    for u in range(n):

        for i in BF[u]:

            if i > max_BF:

                max_BF = i


    L0 = 700

    L = 0.5 * L0 / max_BF # desired edge length


    for u in range(n):

        for v in range(n):

            if u < v and (u not in f0 or v not in f0):

                dist_uv = BF[u][v]

                l_uv = L * dist_uv

                l_uv_sq = l_uv ** 2

                norm_sq =  (pos[u][0] - pos[v][0]) ** 2

                            + (pos[u][1] - pos[v][1]) ** 2
                E += (dist_uv ** -3) * ((norm_sq - l_uv_sq) ** 2)



if fun == 'neighdist':

    total_dist = k = 0

    edge_lengths = []


    for u in range(G.nverts):

        for v in G.neighbors[u]:

            if u < v and (u not in f0 or v not in f0):

                norm_sq =  (pos[u][0] - pos[v][0]) ** 2

                            + (pos[u][1] - pos[v][1]) ** 2


                edge_lengths.append(norm_sq)

                total_dist += norm_sq

                k += 1


    mean_sq_dist = 0.7 * total_dist/k


    for i in range(len(edge_lengths)):


        E += ((mean_sq_dist) - edge_lengths[i]) ** 2



if fun == 'surFace':

    faces = G.faces # list of sets


    S = self.compute_areas(G.sorted_faces, pos)
```

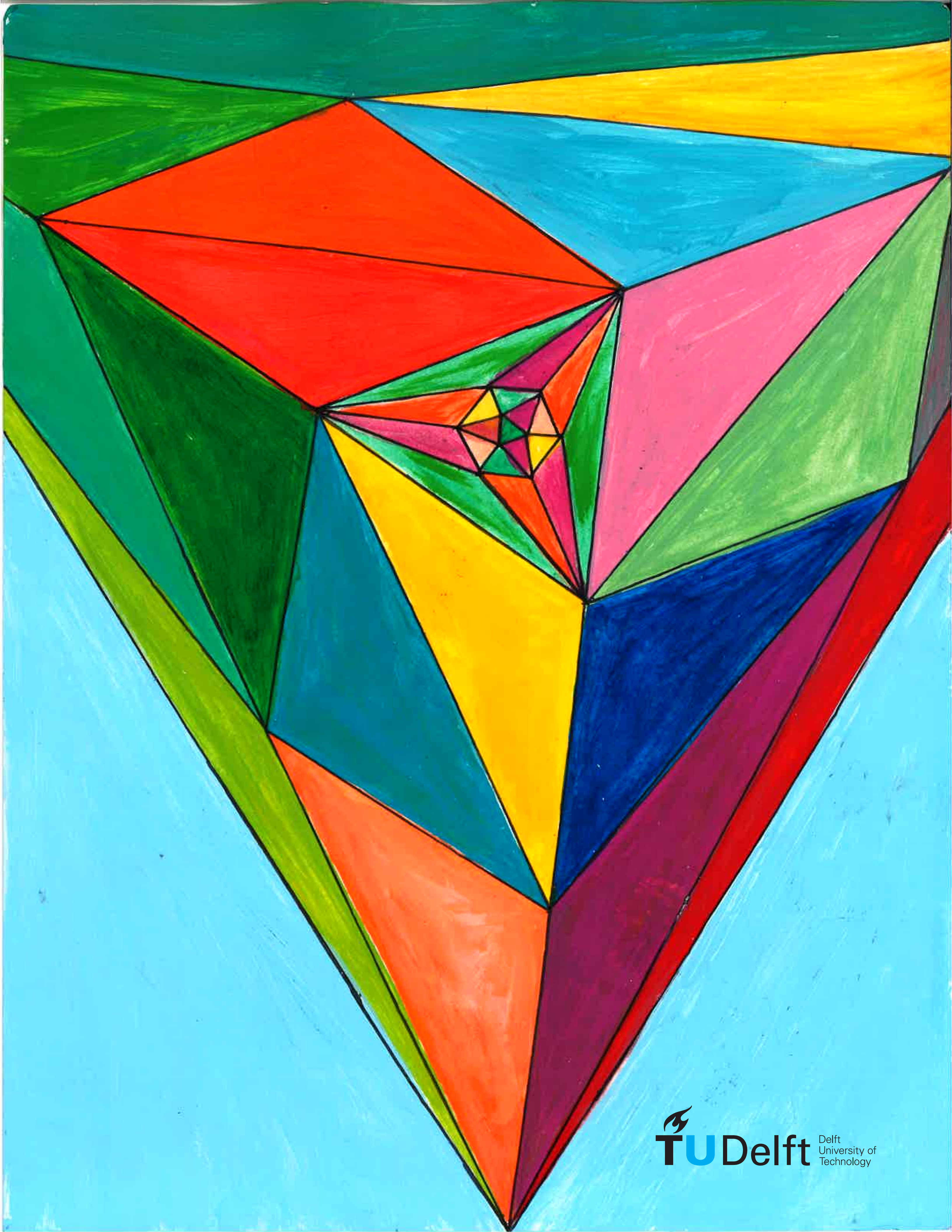
```
            mean_area = S[0] / (len(faces) - 1)

            for i in range(1, len(faces)):

                E += (mean_area - S[i]) ** 2


        return E


    def __call__(self, w):
        W = self.compute_alphas(w)
        pos, A = self.compute_positions(W)

        energy = self.energy(pos, A)

        return energy


def create_model(files, f):
    name = files[f]
    graph = name + '.dat'
    G = read_graph(graph)

    print('G is a', name)
    print('G has', G.nverts, 'vertices,', G.medges, 'edges and',
            len(G.faces), 'faces.')
    faces_dict = G.make_dict()
    print("Face lengths: ", set(faces_dict.keys()))
    print(faces_dict)
    l, f = input("Type length and face index: ", ).split()
    f0 = faces_dict.get(int(l))[int(f)]
    F0 = G.sort_face(f0) # list
    sorted_faces = G.sort_faces(F0)

    funs = ['KK', 'neighdist', 'surFace']
    print(funs)
    function = str(input('Choose a function over which to minimize: ',))

    print('F0 = ', F0)
    print('minimized: ', function)

    return G, F0, name, function, sorted_faces


def optimizer(G, embedding):
# Optimized embedding
    opt = {'disp':True, 'maxiter':100000}
    vec = embedding.random_vector()
    sol = sp.optimize.minimize(embedding, vec, method='COBYLA', options=opt)
    weights = embedding.compute_alphas(sol.x)
    x, A = embedding.compute_positions(weights)

# calculate original energy to compare
    E = 0
    for u in range(G.nverts):
        for v in G.neighbors[u]:
            if u < v:
                norm_sq =  (x[u][0] - x[v][0]) ** 2 + (x[u][1] - x[v][1]) ** 2

                E += norm_sq
    print('E = ', E)

    return sol, weights, x, A, E



def main():

    files = ['cuboctahedron', 'dodecahedron',
             'icosahedron', 'rhombic_dodecahedron']

    f = 2

    G, F0, name, function, sorted_faces = create_model(files, f)

# call optimizer
    embedding = TutteEmbedding(G, F0, function, sorted_faces)
    sol, weights, x, A, E = optimizer(G, embedding)

# possible print statements
    print('sol =', sol)
    print('weights =', weights)
    print('positions =', x)
```

```
# make drawing
    root = Tk()
    title = str(name) + '; F0 = ' + str(F0) + '; ' + str(function)
            + ' = ' + str(trunc(sol.fun)) + ', E = ' + str(trunc(E))
    root.title(title)
    app = CA.App(title, G.nverts, G.neighbors, x, root)
    root.mainloop()

counter = 0
main()
```

APPENDIX D

# Python code: Drawing

```
### file: CanvasApp.py ###


from tkinter import *


class App:

    def __init__(self, title, n, neighbors, positions, master):

        self.n = n

        self.neighbors = neighbors

        self.title = title


    # Canvas

        self.width = self.height = 800

        self.radius = 350


        outer = Frame(master)

        outer.pack()


        left = Frame(outer)

        left.pack(side = LEFT)


        self.canvas = Canvas(left, width = self.width, height = self.height)
        self.canvas.pack(side = LEFT)
        self.canvas.config(background="white")

    # make the drawing.
        self.make_drawing(positions)



    def make_drawing(self, positions):
        self.edges = []
        x = positions

        # Clear canvas.
        for i in self.canvas.find_all():
            self.canvas.delete(i)

        for u in range(self.n):
            r = 1 # 5
            self.canvas.create_oval(x[u][0]-r, x[u][1]-r, x[u][0]+r,
                                    x[u][1]+r, fill="black")
            for v in self.neighbors[u]:
                if u < v:
                    h = self.canvas.create_line(x[u][0], x[u][1],
                                                x[v][0], x[v][1])
                    self.edges.append((h, u, v))
```
,



TU Delft Delft University of Technology